MASTER EN SCIENCES INFORMATIQUES

L'application des mécanismes d' information retrieval pour la construction automatique des systèmes hypertextes

D'Haeyere, Vincent

Award date:
1995

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix
Institut d'Informatique
Rue Grandgagnage, 21
B 5000 - Namur (Belgium)

L'application des mécanismes d'*information retrieval* pour la construction automatique des systèmes hypertextes

Vincent D'Haeyere

Promoteur : Mme M. Noirhomme

Mémoire présenté en vue
de l'obtention du grade de
Licencié et Maître en Informatique

Année académique 1994-1995

Résumé

Dans ce mémoire, nous avons étudié l'application des mécanismes d'*information retrieval* (IR) à la création automatique des systèmes hypertextes. Après une présentation de ces systèmes, nous avons exploré les concepts manipulés en IR et leur application à la représentation et à la classification des documents. Nous avons également présenté les différentes approches qui peuvent conduire à la construction automatique de liens hypertextes. Nous avons construit un programme prototype pour réaliser une classification des messages échangés sur Usenet. Cette réalisation a permis de montrer que les liens obtenus sont, dans l'ensemble, assez satisfaisants mais qu'un certain nombre de difficultés subsistent dans l'application stricte d'une création automatique. Nous suggérons des recherches complémentaires.

Asbtract

In this Master Thesis we have studied the application of information retrieval (IR) mechanisms to the automatic creation of hypertext systems. After a general presentation of those systems, the basic concepts of IR for the representation of documents and their classification have been presented. We also have shown the various approaches to the links creation problem. We have realised a prototype program designed to automatically classify Usenet messages as found on Internet. This has shown that the results are satisfying but that some problems remain for the totally automatic creation of hypertext systems. We suggest further researches to solve or reduce those problems.

Remerciements

Au terme de ce mémoire, je tiens à remercier les personnes qui m'ont aidé et soutenu dans mon entreprise.

Je tiens à remercier Madame M. Noirhomme pour les conseils et orientations qu'elle m'a prodigués.

Ma reconnaissance va aussi à Monsieur G. Kerger qui m'a permis d'effectuer mon stage au sein du Centre de Ressources Multimédia du Centre de Recherche Public Henri-Tudor, Luxembourg et qui m'a orienté dans mes recherches.

Je tiens à exprimer ma gratitude à tous ceux et celles qui, d'une manière ou d'une autre, ont contribué à l'élaboration de ce mémoire.

Je ne pourrais conclure sans remercier mes proches pour les encouragements et le soutien sans lesquels rien n'aurait été pareil.

Table des matières

Introduction

Sans cesse, dans l'environnement qui nous entoure, nous entendons parler de l'information. *Information at your finger tips*, la société de l'information, les autoroutes de l'information et de la télécommunication. Cela semble inéluctable, la société future sera la société de l'information.

Cependant, un problème se pose : nous souffrons d'une surcharge d'informations. L'information vient de la presse, de la littérature, de la télévision, des embryons d'autoroute de l'information... Tous ces média nous fournissent de l'information mais aucun ne nous fournit l'*information*, celle que nous recherchons expressément ou celle qui nous est indispensable à l'accomplissement d'une tâche.

« *Organisons cette information !* » Oui, mais comment l'organiser ? Comment gérer l'ensemble des média ? Comment faire pour que les informations que nous cherchons soient aisément et rapidement accessibles ? Comment rendre la consultation d'informations la plus agréable possible ?

Depuis de nombreuses années, l'*information retrieval* a eu pour objectif d'organiser des collections de documents. Par le développement de techniques spécifiques d'indexation, l'*information retrieval* a su organiser de vastes collections de documents telles que les bibliothèques. Au départ purement manuelle, il est apparu que

l'organisation des documents pouvait être également effectuée par les ordinateurs. Les premiers systèmes documentaires informatisés sont apparus initialement dans les bibliothèques et centres de documentation avant de se répandre dans les entreprises.

Parallèlement à ces développements de l'*information retrieval*, les développements informatiques dans la structuration et la présentation de l'information se sont matérialisés sous la forme de systèmes hypertextes. Ces outils, qui, selon certains auteurs, forment une nouvelle classe de systèmes complexes de gestion de l'information, ont acquis une importance remarquable. Les applications hypertextes sont, en général, faciles à apprendre, agréables à utiliser et peuvent organiser de grandes quantités de données, comme une encyclopédie par exemple.

Tout comme la recherche et l'organisation d'informations, la structuration et la présentation de celles-ci au travers de systèmes hypertextes sont problématiques pour de grandes quantités. Comment un auteur peut-il organiser correctement une application hypertexte ? Comment déterminer les liens à créer entre les documents ?

Certains chercheurs estiment qu'une manière de procéder à la création des applications hypertextes est d'automatiser le processus. Pour cela, diverses approches sont envisagées. Elles consistent à analyser les textes des documents et d'établir des liens, soit entre ceux-ci, soit entre des passages de ceux-ci qui semblent suffisamment similaires. Une des approches utilise les techniques d'*information retrieval* pour construire ces applications hypertextes. Cette application technique a retenu notre attention et est à l'origine de ce mémoire.

Nous structurerons notre exposé de la manière suivante. Dans un premier chapitre, nous explorerons les différentes facettes des systèmes hypertextes. Nous en évoquerons les composants de base et les applications privilégiées. Nous en présenterons une architecture pour, finalement, présenter leurs limitations.

Dans un deuxième chapitre, nous nous intéresserons aux techniques et concepts de base de l'*information retrieval*. Les méthodes pour représenter un document, pour le classifier et pour rechercher l'information seront décrites.

Le troisième chapitre sera consacré à la création des systèmes hypertextes. Nous décrirons les différentes méthodes envisagées pour la création de ces systèmes, tels les systèmes auteurs et les approches automatisées pour la conversion de documents structurés ou non. Nous mettrons en évidence l'adéquation des techniques d'*information retrieval* à la création des systèmes hypertextes.

Dans le quatrième et dernier chapitre, nous présenterons le programme prototype que nous avons développé. Comme exemple d'application, nous nous sommes servis des messages échangés sur les groupes de discussion d'Internet pour construire un système hypertexte. Nous évaluerons ce prototype et l'approche choisie en analysant les liens découverts entre les messages.

Chapitre 1 :
Les systèmes hypertextes

1. Introduction

Les systèmes hypertextes émergent comme une nouvelle classe de systèmes complexes de gestion de l'information [BALASU., 1994]. Ces outils souples fournissent un moyen d'accès commode à de grandes masses d'informations.

Afin de cerner les enjeux des systèmes hypertextes et de présenter le cadre dans lequel ce travail a été élaboré, nous commencerons par un bref historique. Nous présenterons une définition des concepts de base puis nous envisagerons les types d'applications où l'usage d'un système hypertexte peut s'avérer utile. Nous continuerons par une présentation de l'architecture des systèmes hypertextes et une énumération des critères d'utilisabilité des systèmes hypertextes. Enfin, nous évoquerons quelques points faibles de ces systèmes. La création des systèmes hypertextes sera abordée dans un chapitre ultérieur.

2. Historique

D'un point de vue historique, les auteurs [NIELSEN, 1990A] et [BALASU., 1994] sont unanimes pour attribuer l'idée fondatrice des systèmes hypertextes à Vannevar Bush [BUSH, 1945], publié dans un article désormais célèbre « *As we may think* » paru dans *The Atlantic Monthly*. Reprenons certains passages de cet article (extrait de [NIELSEN, 1990A] et [BALASU., 1994]).

« The *memex* device is a device in which an « *individual stores his books, records and communications, and which is mechanised so that it may be consulted with exceeding speed and flexibility. It is an enlarged intimate supplement to his memory* » » [BALASU., 1994]

« Bush proposed « *associative indexing, the basic idea of which is a provision whereby any item may be caused at will to select immediately and automatically another. This is the essential feature of memex. The process of tying two items together is the important thing.* » » [BALASU., 1994]

« Bush envisaged that « *most of the Memex content are purchased on microfilm ready for insertion. Books of all sorts, pictures, current periodicals, newspapers, are thus obtained and dropped into place. Business correspondence takes the same path.* » » [NIELSEN, 1990A, p. 30]

Les deux idées principales des systèmes hypertextes sont exposées : la première est qu'un individu enregistre toutes les informations qu'il désire sur un support particulier (microfilm); la deuxième idée est qu'un mécanisme permet de faire des accès rapides grâce à des liens préétablis ou construits par l'individu lui-même.

Ce n'est pas Bush qui propose le mot « *hypertexte* », mais bien Ted Nelson, en 1965, le père du projet *Xanadu*. Ce projet, qui n'a pas encore abouti et n'aboutira pas dans un avenir proche - peut-être jamais -, consiste à créer un *repository* de tous les documents produits par l'homme (*docuverse - universe of documents*). Il suppose, entre autres, que les documents soient gérés sur des sites différents, que toutes les versions soient conservées et que nous puissions accéder au moindre octet des

documents en question. Cette vision futuriste restera sans doute à jamais de la science-fiction, même si une implémentation partielle existe.

Le développement des systèmes hypertextes continue dans les années 60 et 70. Le projet *Hypertext Editing System* est développé à l'Université de Brown par Andries Van Dam. Ce premier système fonctionnel tournait sur un *mainframe* IBM/360. Il a été utilisé par le Houston Manned Spacecraft Center pour les missions Appolo. Le projet FRESS (*File Retrieval and Editing System*) vient aussi de l'Université de Brown et dérive directement d'*Hypertext Editing System*. Ces deux systèmes disposent des fonctionnalités de base des systèmes hypertextes - les liens et le passage entre documents - mais dans une interface textuelle.

En 1978, le MIT Architecture Machine Group développe le premier système hypermédia *Aspen Movie Map*. Cette application permet de simuler un voyage dans la ville de Aspen. Elle utilisait des vidéo disques pour enregistrer les photographies des rues. L'aspect hypermédia découle du fait que chaque photographie permettait de répondre aux sollicitations de l'utilisateur actionnant un joystick pour simuler le comportement d'un conducteur ou celui d'une personne entrant dans un bâtiment de la ville d'Aspen.

En 1985, le *Symbolic Document Examiner* est développé pour les utilisateurs des stations de travail Symbolics. Cette interface hypertexte a été conçue pour accéder à la documentation *online* des stations de travail (l'équivalent de 8.000 pages de manuels imprimés, sous la forme de 10.000 noeuds, 23.000 liens pour une occupation de 10 Méga). Le stade de prototype de laboratoire est à cet instant dépassé pour aboutir à un outil efficace pour les utilisateurs.

A partir de 1985, nous pouvons dire que les systèmes hypertextes se développent commercialement. Parmi les plus connus se trouvent *NoteCards* de Xerox, *Intermédia* de l'Université de Brown, *Guide* de Office Workstation Limited et *Hypercard* de Apple.

La reconnaissance de l'intérêt de l'hypertexte est acquise par la création de conférences par l'Association for Computing Machinery (Hypertext'87 et suivantes).

Synthétiquement, on retrouve l'historique dans le tableau suivant (Tableau 1):

1945	Vannevar Bush propose Memex
1965	Ted Nelson utilise le mot *hypertexte*
1967	Hypertext Editing System and FRESS
1978	Aspen Movie Map, premier vidéo disque hypermédia du MIT Architecture Machine Group
1985	Symbolics Document Examiner
1985	Intermédia de l'Université de Brown
1986	Guide est commercialisé par OWL, premier système hypertexte largement disponible
1987	Apple introduit Hypercard
1987	Première conférence de l'A.C.M. sur le sujet - Hypertext'87

Tableau 1 : histoire de l'hypertexte (adapté de [NIELSEN, 1990A, p. 29])

3. Définition des concepts de base

Nielsen [NIELSEN, 1990A, p. 1] nous donne une définition de l'hypertexte :

« Une façon aisée de définir un système hypertexte, c'est de le comparer avec un texte traditionnel, comme un livre. Les textes traditionnels sont par essence essentiellement séquentiels, c'est-à-dire qu'il existe une séquence linéaire qui définit l'ordre dans lequel le texte doit être lu. [...] Un hypertexte est non séquentiel. Il n'y a pas un ordre qui détermine la séquence de lecture du texte. »

Alors que dans un texte traditionnel nous lisons les pages dans l'ordre (page 1, page 2, page 3), dans un système hypertexte, le lecteur peut lire le texte dans l'ordre qu'il souhaite. Le schéma du même auteur représente les choix possibles du lecteur (Figure 1).

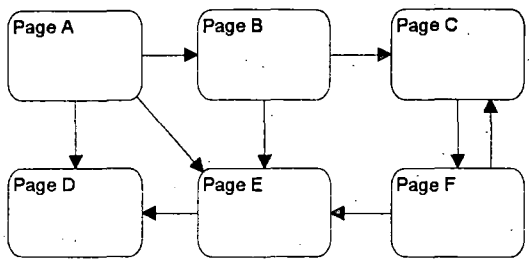


Figure 1 : vue simplifiée d'une petite structure hypertexte, utilisant 6 noeuds et 9 liens [NIELSEN, 1990A, p. 1]

C'est le lecteur qui choisira quelle page il désire lire à la suite de la page qu'il est en train de consulter. Si le lecteur lit la page A, il peut enchaîner sa lecture, selon son choix, sur la lecture vers les pages B, E ou D.

La lecture d'un hypertexte ne se fait pas de manière traditionnelle, mais nous parlerons de navigation (*browsing*) pour mettre en évidence le fait que le lecteur détermine activement l'ordre dans lequel il désire enchaîner la lecture des pages.

Les désavantages des textes traditionnels sont bien connus. Reprenons ceux énumérés par [COOK, 1988] (repris de [BALASU., 1994]).

- comparé au moyen électronique, même si on trouve régulièrement des ouvrages multivolumes, le texte imprimé reste limité dans le nombre d'informations qu'il contient. Il est malaisé de chercher une information dans de multiples volumes;
- ils sont difficiles ou impossibles à mettre à jour périodiquement;
- la recherche d'informations est majoritairement lexicale, c'est-à-dire par l'utilisation des tables des matières et index. Les références croisées sont en général peu nombreuses. Les index reflètent le choix des auteurs et sont limités dans l'information qu'ils apportent;
- l'information ne peut pas être arrangée dynamiquement afin de satisfaire les besoins d'un utilisateur;
- l'information est dispersée dans un grand nombre de volumes et leurs recherche et stockage sont problématiques.

L'hypertexte offre une gestion souple de l'information. Selon [BALASU., 1994], les systèmes hypertextes présenteraient les avantages suivants :

- l'hypertexte permet une navigation aisée dans l'information;

- les supports informatiques permettent de conserver un grand nombre d'informations;
- il peut fournir une indication visuelle sur les informations spécifiées par un utilisateur;
- les utilisateurs sont désireux de pouvoir conserver une trace de leurs recherches dans des ouvrages de référence et d'utiliser les informations recueillies.

Un système hypertexte est constitué de pièces d'information inter-reliées (texte ou autre). L'information peut, soit être reprise sur un écran d'ordinateur, soit dans une fenêtre déroulante, ou encore dans une partie de l'écran ou d'une fenêtre. Chaque unité d'information est appelée un **noeud.**

Quelle que soit la taille de ce noeud, il peut exister un certain nombre de **liens** vers d'autres noeuds. En général, le nombre de liens n'est pas déterminé à l'avance mais dépend étroitement du contenu informationnel du noeud et de l'interrelation qui existe entre les noeuds du système hypertexte.

Un lien hypertexte relie donc deux noeuds. Il est dirigé, c'est-à-dire qu'un lien relie deux noeuds dans lesquels se distinguent le noeud de départ et le noeud de destination. Afin que le système hypertexte soit complet, un lien comporte toujours un noeud de départ et un noeud d'arrivée. Sinon des liens pendants pourraient exister dans le système. Toutefois, il se peut que, pour certaines applications, la complétude du réseau ne soit pas garantie. Il faut alors que le développeur prenne des mesures adéquates pour gérer ces circonstances.

L'ensemble des noeuds et des liens peut être représenté sous la forme d'un réseau. La structure de ce réseau est toujours interne au système hypertexte et rarement visible par l'utilisateur. Un des rares systèmes à présenter une vue du réseau est *NoteCards*, développé par Halasz au Xerox PARC. Cette vue doit être dynamique; elle ne reproduit qu'une partie du réseau (car celui-ci est trop important pour être reproduit complètement de façon intelligible sur un écran d'ordinateur). Cette représentation doit changer en fonction du noeud courant.

Dans la plupart des systèmes hypertextes existants, les liens sont unidirectionnels, c'est-à-dire que le système met en évidence les noeuds d'arrivée

possibles, mais n'indique pas les noeuds pour lesquels ce noeud est la destination d'un lien. Le système *Intermédia* développé à l'Université de Brown est un contre-exemple. *Intermédia* supporte les liens bidirectionnels ce qui se révèle d'ailleurs adéquat dans certaines applications. L'exemple repris par Nielsen est celui d'une application sur la poésie chinoise, qui reprend des anthologies et leurs poèmes. Pour chaque poème, il existe une liste d'anthologies.

La définition traditionnelle d'hypertexte implique que le système ne travaille qu'avec du texte. Certains utilisent le terme *hypermédia* pour désigner des systèmes qui utilisent aussi bien le texte, les graphiques, les images ou encore les animations. Cette distinction n'a plus beaucoup de sens aujourd'hui car la plupart des systèmes existants mélangent allègrement ces médias. Nielsen [NIELSEN, 1990A, p. 5] propose d'abandonner cette distinction et d'utiliser les termes *hypertexte* et *hypermédia* de façon interchangeable, en préférant le terme *hypertexte*. Il justifie ce choix en affirmant qu'il est superflu d'utiliser un terme spécifique aux systèmes constitués uniquement de textes.

Dans la suite de ce mémoire, nous serons amené à faire cette distinction pour mettre en évidence le fait que le prototype développé ne porte que sur des systèmes hypertextes dont les noeuds seront uniquement des textes traditionnels (cfr. Chapitre 4).

4. Les applications des systèmes hypertextes

Nielsen [NIELSEN, 1990A, pp. 43-82] ainsi que [BALASU., 1994, chap. 7] établissent un « catalogue » d'applications qui conviennent à une transposition dans un système hypertexte.

Toutes les applications ne doivent pas être transformées en systèmes hypertextes. Shneiderman [SHNEID., 1989] (repris de [NIELSEN, 1990A, p. 43]) a proposé les trois règles d'or de l'hypertexte. Une application est adaptée à l'hypertexte pour autant que :

- que les informations soient organisées en de nombreux fragments;
- que les fragments soient liés entre eux;
- que l'utilisateur n'ait besoin que d'une petite partie de l'information à la fois.

Ces trois critères se retrouvent dans les applications citées ci-après (Tableau 2).

Type d'applications	Exemples
Applications informatiques	documentations en ligne assistance à l'utilisateur ateliers de génie logiciel systèmes d'exploitation
Applications d'affaire	manuel de réparation et autres dictionnaires et ouvrages de références applications d'audit applications de démonstration catalogues de produits et publicité
Applications intellectuelles	organisation d'idées support au *brainstorming* journalisme recherche
Applications éducatives	apprentissage des langues étrangères apprentissage de la littérature classique musées
Applications de loisirs	guides touristiques bibliothèques fictions interactives

Tableau 2 : les applications privilégiées des systèmes hypertextes (synthèse personnelle de [NIELSEN, 1990A, pp. 43-82]

5. Architecture des systèmes hypertextes

Selon [BALASU., 1994], un système hypertexte est composé des éléments suivants :

- une interface graphique, qui permet la navigation, affiche éventuellement une partie du réseau hypertexte, guide l'utilisateur et permet à celui-ci d'activer les liens;
- un système auteur muni d'outils pour créer et gérer l'information du système hypertexte;
- des mécanismes traditionnels d'*Information Retrieval*, comme la recherche sur les mots-clés, sur l'auteur, ...;

- un moteur hypermédia pour la gestion et l'affichage de l'information et des liens;
- un dispositif de stockage afin de conserver l'information et les liens.

Nous allons d'abord examiner un modèle d'un système hypertexte. Dans un même temps, nous en présenterons les composants, c'est-à-dire les noeuds et les liens.

5.1. Une architecture en couche

Selon [CAMPBELL, 1988] (repris de [NIELSEN, 1990A, pp. 101-104]), un système hypertexte peut être découpé en couches (Figure 2).

- le niveau de présentation : l'interface utilisateur;
- le niveau *Hypertext Abstract Machine* qui gère les noeuds et les liens;
- le niveau base de données : stockage, partage de données et accès au réseau.

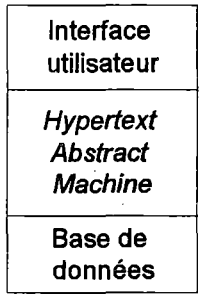


Figure 2 : modèle de Campbell et Goodman [CAMPBELL, 1988]

Détaillons chacune de ces couches :

5.1.1. Le niveau base de données

Il s'agit de la couche de bas niveau qui s'occupe du stockage des informations nécessaires au fonctionnement du système hypertexte. Ces informations sont stockées sur l'ordinateur même (disque dur ou CD ROM) ou sur une autre machine connectée au réseau. Peu importe la manière dont l'information est conservée, l'accès rapide à toutes les informations doit être garanti.

Le niveau base de données doit également gérer la concurrence d'accès, la sécurité et autres tâches typiques des bases de données.

Pour ce niveau, les noeuds et les liens sont des données comme les autres, sans signification particulière. Ce sont seulement des objets que la base de données doit manipuler comme tous les autres objets.

Nielsen [NIELSEN, 1990A, p. 102] affirme « *But in any case the hypertext field would do well in taking advantage of the extensive work and experience in the traditional database field for the design and implementation of the database level.* ». Il ne faut cependant pas confondre une base d'informations hypertextes et une base de données traditionnelle. Une base d'informations hypertextes ne possède pas de structures régulières, ni de définitions centralisées comme une base de données des employés dans une entreprise. De plus, les liens entre noeuds sont basés sur la sémantique, pas uniquement sur une définition structurelle.

Selon [BALASU., 1994], les caractéristiques d'une base de données relationnelle sont inadéquates pour les systèmes hypertextes. Ceux-ci font partie d'une classe d'applications manipulant de grandes quantités d'informations, souvent complexes, comme, par exemple, les applications de *Computer Aided Design* ou de *Computer Aided Manufacturing*, les applications de gestion documentaire, ou encore les *Geographical Information Systems* pour lesquelles les bases de données relationnelles sont inappropriées car elles n'offrent pas de structure assez riche.

Certains chercheurs estiment qu'une base de données orientée objet (BD OO) serait adaptée aux besoins d'un système hypermédia. Leurs avantages seraient d'être plus expressives dans la modélisation et plus souples que les BD relationnelles.

Les caractéristiques souhaitées pour les bases de données supportant les systèmes hypertextes sont la standardisation d'accès, la distribution sur différentes machines, le support d'accès concurrent, le maintien de l'intégrité des

données ou encore l'usage de structure dynamique. Et, de plus, le support pour les objets multimédia, tels que le son, l'image ou les animations vidéo.

5.1.2. Le niveau Hypertext Abstract Machine (HAM)

Le niveau HAM se trouve en sandwich entre la base de données et l'interface utilisateur. C'est à ce niveau que le système détermine la nature même des noeuds et des liens et qu'il maintient les relations entre eux. Il connaît aussi la forme des liens et des noeuds, et les attributs dont ils disposent.

a) Les noeuds [NIELSEN, 1990A, pp.105-106]

Les noeuds sont les éléments principaux des systèmes hypertextes. La principale distinction peut être faite entre les systèmes dits « *frame-based* » et les systèmes basés sur les fenêtres (*window-based*).

Un cadre (*frame*) constitue une unité informationnelle qui prend un certain espace sur l'écran de l'ordinateur. Peu importe la quantité d'informations de ce noeud. Une carte dans le système *HyperCard* d'Apple est constitué d'un cadre. Le défaut de ce système consiste en la grande quantité d'informations d'un noeud. Il s'avérera sans doute nécessaire de scinder l'information sur plusieurs pages. Par contre, l'avantage est que tout ce qui tient à la navigation peut se trouver dans le cadre ainsi défini.

Un système basé sur un système de fenêtrage affiche l'information dans une fenêtre munie d'un moyen de défilement (*scrolling*). L'information est présentée dans sa totalité dans la fenêtre mais il est peut-être nécessaire de faire défiler celle-ci, car la fenêtre n'affiche qu'une partie du contenu informationnel. L'avantage est qu'une information, aussi importante soit elle, est regroupée dans une seule et même unité spatiale, la fenêtre. Ceci évite que l'utilisateur soit désorienté par plusieurs pages pour une même information. Un grand désavantage tient au fait que l'auteur n'a pas le contrôle total de l'apparence de l'information puisque l'utilisateur peut faire défiler la fenêtre à sa guise. Un

exemple est le système d'aide de *Microsoft Windows* qui utilise un système de fenêtre.

Cette distinction est assez radicale. La plupart des systèmes modernes combinent des aspects « *frame-based* » et « *window-based* ». Par exemple, *HyperCard* possède comme élément graphique une fenêtre déroulante.

b) Les liens [NIELSEN, 1990A, pp.106-111]

Les autres éléments fondamentaux des systèmes hypertextes sont les liens. Le lien se présente sous la forme de l'information qui joue à la fois le rôle de contenu informationnel et de lieu d'activation du lien. La plupart des noeuds sont explicites, c'est-à-dire qu'ils ont été établis par l'auteur entre le noeud de départ et le noeud de destination. Certains systèmes utilisent des liens implicites qui découlent de certaines propriétés de l'information.

La majorité des systèmes hypertextes utilisent des liens unidirectionnels munis d'un noeud de départ et d'un noeud de destination. Dans les systèmes « *frame-based* » se trouvent seulement des liens entre cadres. Il est parfois intéressant de pouvoir faire des liens avec un point précis du cadre.

Une autre décision à prendre est de, soit mettre les liens en évidence, soit les représenter discrètement . Cela dépend en fait du nombre de liens trouvés par noeud. Si le nombre de liens est fort élevé, l'écran risque d'être surchargé et illisible.

Les liens peuvent être typés, c'est-à-dire munis de propriétés pouvant les distinguer des autres noeuds, par exemple, le nom de l'auteur du lien ou la date de création du lien. Les liens peuvent être classifiés selon une taxonomie suivant le type de relation qu'il représente (lien d'abstraction, de formalisation, de simplification, ...).

Nous avons défini les liens comme liant un noeud de départ à un noeud de destination. Il existe des systèmes dans lesquels certains liens possèdent plusieurs noeuds de destination. C'est l'utilisateur qui choisit alors le lien de

destination désiré parmi les choix possibles. Cette particularité apparaît dans le système d'aide de *Microsoft Windows* par exemple.

Le niveau *Hypertext Abstract Machine* est le lieu idéal pour définir des moteurs hypertextes, c'est-à-dire des composants logiciels qui permettent de gérer le comportement d'applications hypertextes quelconques. C'est le cas d'*HyperCard* ou encore du système d'aide de *Microsoft Windows* qui sont des outils génériques pour afficher respectivement des piles *HyperCard* ou des fichiers d'aide d'applications *Windows*.

5.1.3. Le niveau interface utilisateur

Ce niveau s'occupe de la présentation à l'utilisateur des informations contenues dans le système hypertexte. Il gère aussi le dialogue avec l'utilisateur en définissant les actions qui permettent de changer de noeuds, ou encore en affichant une vue partielle du réseau hypertexte.

Si les liens sont typés et munis d'attributs, le niveau interface utilisateur peut permettre ou empêcher l'utilisateur de voir des liens ou des informations selon les spécifications établies par l'utilisateur lui-même.

Selon [CONKLIN, 1987] (repris de [BALASU., 1994]), la désorientation et la surcharge cognitive sont les deux challenges de l'hypertexte. Il affirme même que ces deux problèmes « *may ultimately limit the usefullness of hypertext.* »

Le problème de la désorientation (« *getting lost in space* ») provient de ce que l'utilisateur désire savoir où il se situe dans le réseau hypertexte, d'où il vient, ou encore comment atteindre un autre endroit. Dans celui-ci, ce phénomène est accentué par le nombre d'informations à disposition.

La surcharge cognitive se trouve aussi bien chez le créateur du système hypertexte que chez l'utilisateur. Le créateur doit organiser l'information et garder une trace des noeuds et des liens entre eux. Quant à l'utilisateur, il doit prendre sans cesse des décisions sur ce qu'il désire consulter, sur quels liens

abandonner ou suivre, ... C'est un problème sérieux, surtout si le nombre de noeuds et de liens est élevé.

L'interface utilisateur prend le plus souvent la forme d'une interface graphique. Ces outils de navigation (*browsers*) sont les vitrines de l'information du système hypertexte. Ils affichent l'information selon une présentation conviviale qui tend à réduire la désorientation en fournissant, par exemple, un plan du réseau hypertexte. Ces outils permettent aussi d'afficher une partie seulement du système hypertexte ou encore un noeud isolé. L'information ainsi limitée réduit la surcharge cognitive de l'utilisateur.

La présentation de l'information peut aussi être dynamique, c'est-à-dire que celle-ci correspond aux spécifications de l'utilisateur. Par exemple, montrer tous les liens de tel auteur ou ceux qui sont de tel type. Une autre façon de présenter l'information est de fournir une indication de la pertinence du document de destination d'un lien en fonction de critères spécifiés par l'utilisateur ou déduite de son comportement précédent.

Un autre mécanisme pour faciliter la navigation est l'usage de chemins. Un chemin (*path*) permet à l'auteur de déterminer un ordre de présentation des noeuds afin qu'ils ne soient pas consultés dans un ordre impropre à une bonne compréhension ou à la logique de l'application. Ce mécanisme réduit à la fois la désorientation et la surcharge cognitive. Ce même mécanisme pourrait être utilisé par l'utilisateur pour enregistrer une séquence de documents qu'il vient de consulter afin de pouvoir les visionner à une date ultérieure. La pose de marques spécifiques (*bookmarks*) peut s'avérer bénéfique pour l'utilisateur.

Le mécanisme de retour en arrière ou la liste historique des consultations sont des moyens complémentaires pour guider l'utilisateur, même si leur efficacité est discutable [NIELSEN, 1990A] (repris de [BALASU., 1994]).

On peut encore citer la vision d'une partie du système hypertexte selon la *vue en oeil de poisson* (*fisheye view*), l'usage de technique de parcours et de zoom sur le réseau hypertexte, l'usage de menus ou de fenêtres imbriquées, ...

6. L'utilisabilité des systèmes hypertextes

Selon [WRIGHT, 1991] (repris de [BALASU., 1994]), l'évaluation d'un système hypertexte doit se faire sur les critères suivants :

- l'adéquation de l'interface au contenu du système hypertexte. L'auteur doit faire des compromis entre le contenu du système, les fonctionnalités, l'affichage et le contrôle, tout en gardant à l'esprit que l'utilité du système est déterminée par la facilité de navigation, la population des utilisateurs-cibles et le but du système;

- l'acceptation du système par les utilisateurs. Dès que les utilisateurs ont employé un système particulier, ils s'attendent à trouver des fonctionnalités similaires dans d'autres systèmes. Ils peuvent aussi avoir un comportement plus « pointu » qui ne correspond pas nécessairement au design de l'interface ou du système utilisé.

- l'adaptabilité de l'interface selon la tâche de l'utilisateur. L'interface du système doit être adaptée à la tâche en cours de l'utilisateur, comme, par exemple, l'annotation de documents;

- l'expertise de l'utilisateur en tant qu'utilisateur d'informations;

- les coûts de production et de dissémination des systèmes hypertextes.

Nielsen [NIELSEN, 1991] (repris de [BALASU., 1994]) fournit 4 catégories d'évaluation des systèmes hypertextes. Ces catégories sont l'utilité du système pour la tâche de l'utilisateur, l'intégrité des informations du système (information à jour, facilité de maintenance), l'utilisabilité du système en terme de facilité d'apprentissage du système, de facilité d'utilisation et de gestion d'erreurs, et la dernière catégorie porte sur l'esthétique du système.

L'utilisabilité des systèmes hypertextes est en général associée aux 5 critères suivants [NIELSEN, 1990A, p. 143] :

- **facilité d'apprentissage :** le système doit être facile à utiliser et permettre à l'utilisateur de « rentrer » directement dans l'application. Il doit aussi favoriser la création d'applications claires et structurées;

- **efficacité d'utilisation :** l'utilisateur à la recherche d'une information précise trouvera rapidement celle-ci si elle est présente ou pas. L'auteur pourra créer rapidement une application et y apporter des modifications;

- **facilité à mémoriser l'utilisation du système :** l'utilisateur peut facilement retenir le fonctionnement du système tandis que l'auteur peut aisément maintenir une application hypertexte;

- **génération de peu d'erreurs à l'utilisation :** le moteur hypertexte doit permettre à l'utilisateur de retourner sur ses pas. Pour l'auteur, le système doit l'avertir des liens pendants et l'auteur doit s'assurer de la correction du contenu de l'information;

- **utilisation agréable :** le système doit être agréable à utiliser, au moins autant que la même information dans une forme plus traditionnelle. Les utilisateurs ne se sentiront pas frustrés car ils auront le sentiment de contrôler le système et d'y circuler à leur guise.

Il n'y a pas à proprement parler de mesures pour ces différents critères. Il s'agit surtout d'appréciations qualitatives basées sur son expérience en tant qu'utilisateur.

Nielsen [NIELSEN, 1990A, pp. 144-147] décompose l'utilisabilité du système hypertexte en un certain nombre de facteurs. Il résume ces facteurs dans le schéma suivant (Figure 3).

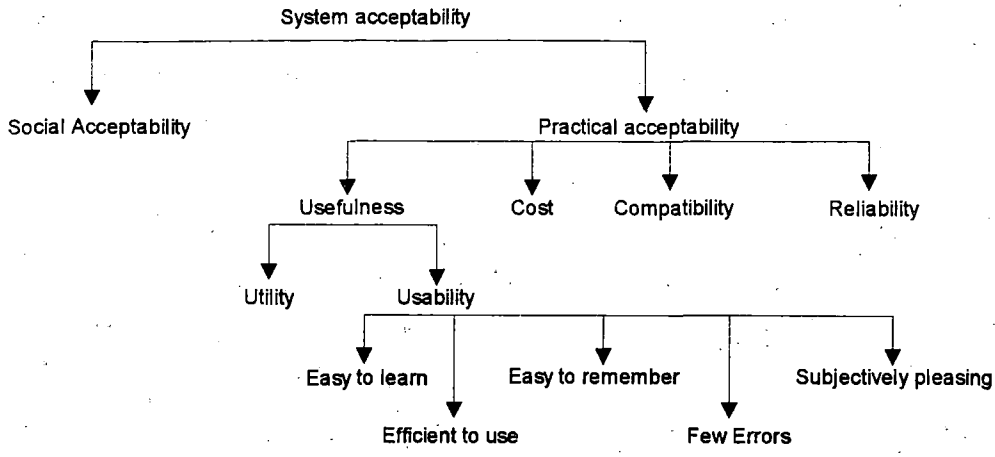


Figure 3 : les paramètres associés à l'utilisabilité du système hypertexte[NIELSEN, 1990A, p. 145]

Dans les systèmes hypertextes, l'utilisabilité peut être raffinée en considérant qu'une application hypertexte est constituée de l'utilisabilité du moteur hypertexte, de l'utilisabilité du contenu et de la structure de la base d'informations hypertextes. Bien que l'utilisateur ne distingue pas ces deux éléments, cela reste intéressant pour le concepteur qui peut alors déterminer « à qui la faute ? ».

7. Les limites des systèmes hypertextes

Tout d'abord reprenons les remarques qui ont été formulées à propos de l'utilisateur qui se sent désorienté dans l'hypertexte. Ce problème n'est toujours pas résolu et a alimenté la recherche dans le domaine.

La surcharge cognitive est toujours d'actualité. Des applications hypertextes qui sont mal conçues, pas structurées, dont les explications sont peu claires, ont peu de chance d'être utilisées et, si l'utilisateur les emploie, la surcharge cognitive est évidente.

Autre remarque : à coté des mécanismes de navigation, les systèmes hypertextes doivent fournir des systèmes d'*information retrieval*, comme la recherche basée sur des mots clefs ou des concepts. Ces aspects sont importants pour des utilisateurs

chevronnés qui recherchent une information et qui ne veulent pas « perdre » leur temps en des navigations inutiles. L'implémentation dans les systèmes est fortement variable.

Il n'y a pas non plus de standard pour l'échange d'applications hypertextes entre systèmes hétérogènes. Chaque système utilise ses conventions pour les noeuds et pour les liens (bi- ou unidirectionnel, typé ou pas), ... De plus, l'interface est largement dépendante de la plate-forme utilisée. Il y a bien quelques comités qui tentent de proposer un standard. Mais ne viennent-ils pas trop tard ?

Une autre limitation des systèmes hypertextes est l'absence de méthodes fiables pour transformer des documents traditionnels en un système hypertexte le plus adéquat possible, c'est-à-dire offrant des liens sémantiques entre les noeuds et pas uniquement des liens structurels (précédants/suivants). C'est dans le domaine de la construction automatique de systèmes hypertextes au moyen des techniques d'*information retrieval* que ce mémoire sera développé.

Chapitre 2 :
Les mécanismes d'*information retrieval*

1. Introduction

D'un point de vue historique, l'*information retrieval* vient de la gestion documentaire, telle celle pratiquée dans les bibliothèques ou les collections de documents. Ce domaine, intégré dans le domaine plus vaste des systèmes d'information et des technologies associées, s'est révélé utile aussi bien pour les documentalistes que pour les analystes en charge du design d'un système d'information [VICKERY, 1970, p. v]. L'*information retrieval* consiste à créer des systèmes de classification à destination des bibliothèques ou d'élaborer des techniques pour la gestion manuelle de grandes quantités de données.

Plus récemment, l'*information retrieval* a évolué vers les applications informatisées. Ces applications consistent, pour la plupart, à effectuer des comparaisons entre une collection de documents et une requête afin d'obtenir un ensemble de documents les plus pertinents possibles par rapport à la requête. Un autre type d'application est celui de la construction automatique de résumés d'articles.

Ce chapitre va s'attacher à décrire les mécanismes mis en oeuvre dans les systèmes d'*information retrieval*. Nous explorerons le fonctionnement de ces systèmes et les techniques à utiliser [SALTON, 1983] et [vanRIJS, 1979].

2. Les concepts d'*Information Retrieval*

Afin d'éclairer la lecture des sections suivantes, nous allons présenter une série de définitions informelles des concepts manipulés dans les systèmes d'*information retrieval*.

Les documents

Les documents sont composés de mots, de phrases et de paragraphes. Ils sont découpés éventuellement en chapitres , sections ou sous-sections. Un document ne contient pas que du texte; il peut aussi contenir des images, des tableaux ou des graphiques. Cependant, dans la suite de ce mémoire, seuls seront considérés les documents composés exclusivement de textes.

Les requêtes

Les requêtes sont en général émises par des utilisateurs qui cherchent de l'information dans un système contenant une collection de documents. En réponse à ces requêtes, les utilisateurs désirent obtenir une liste de documents satisfaisant la requête, c'est-à-dire qu'ils offrent une ressemblance manifeste avec celle-ci.

Le rappel (recall)

Le rappel est défini comme le nombre de documents restitués par le système en réponse à une requête.

La précision

La précision est définie comme le nombre de documents réellement pertinents obtenus du système en réponse à une requête.

L'indexation

L'indexation d'un document consiste à choisir les mots clés appropriés pour représenter celui-ci.

Les mots clés ou termes d'index

Les mots clés sont choisis afin de représenter un document. Les mots clés doivent être suffisamment spécifiques pour avoir une idée du contenu précis du document.

Les racines (stems)

Les racines sont des mots dont la terminaison a été retirée. Ce procédé permet de réduire des mots tels que *computing, computerize, compute* à la racine commune *compute.*

La pondération

La pondération est utilisée pour quantifier l'importance d'un mot clé dans la représentation d'un document. Dit d'une autre manière selon la théorie de l'information chère à Shanon, la pondération est l'entropie d'information d'un symbole au sein d'un alphabet dont les symboles ne sont pas équiprobables.

Le représentant de document

Les représentants de document sont en général des vecteurs de mots clés, éventuellement munis d'une pondération. Un représentant de document est donc un condensé du document. Il sera souvent manipulé en lieu et place du document dont il est issu. Le profil du document peut aussi être muni

d'informations bibliographiques, telles que le nom de l'auteur, l'éditeur ou la date de parution.

Les classes de documents (clusters)

Les classes de documents regroupent une série de documents selon des critères établis par le système ou déterminés au préalable par un utilisateur.

Les représentants de classes de documents (clusters' representatives)

A l'image des représentants de document, le représentant d'une classe de documents est un vecteur de mots clés constitué à partir des vecteurs de mots clés des documents repris dans la classe. Le représentant de la classe de documents est utilisé en lieu et place des documents dans un certain nombre de cas. Il synthétise en quelque sorte les documents de la classe. Les mots clés peuvent également être pondérés.

3. Le représentant de document

Pour décrire un document en *information retrieval*, il existe plusieurs techniques. Citons entre autres les méthodes basées sur les *n*-gram [KIMBRELL, 1988] ou celles qui génèrent une « signature » du document à l'image de la signature cryptographique d'un document [CROFT, 1988]. Mais la méthode la plus répandue - et dont l'utilité et l'efficacité ne sont plus à démontrer - est celle qui consiste à attribuer des mots clés à un document. Cette attribution peut être effectuée, soit manuellement, soit automatiquement par un programme.

L'indexation doit rencontrer trois objectifs [SALTON, 1983, p. 54] :

- permettre à un utilisateur de localiser des documents selon le sujet traité;
- relier des documents entre eux et les relier au sujet traité ou à un sujet proche;
- permettre d'établir la pertinence d'un document particulier par rapport à une demande précise.

L'indexation est caractérisée par l'exhaustivité de l'index et la spécificité de celui-ci.

L'exhaustivité de l'index rend compte de la précision avec laquelle tous les concepts et notions repris dans le document sont présents dans l'index. Plus l'exhaustivité est élevée, plus le nombre de documents pertinents par rapport à une requête sera élevé.

La spécificité de l'index fait référence à la spécificité des termes utilisés comme mots clés dans le représentant du document. Plus ces termes sont précis et spécifiques, moins nombreux sont les documents présentés suite à une requête.

Il y a donc un équilibre à trouver entre spécificité et exhaustivité. C'est en rapport direct avec la précision et le rappel attendu du système d'*information retrieval*. Par exemple, une exhaustivité peu élevée conduit à un bon niveau de rappel et à un bas niveau de précision. Tandis qu'une exhaustivité faible conduit à un niveau de rappel bas et une précision élevée.

L'inverse est vrai pour la spécificité de l'index, c'est-à-dire qu'une spécificité élevée conduit à une haute précision et un rappel faible et qu'une spécificité réduite conduit à une faible précision et un niveau de rappel élevé. Il doit donc y avoir un niveau optimal pour l'exhaustivité et la spécificité de la méthode d'indexation.

Nous allons envisager brièvement l'indexation manuelle puis nous développerons l'indexation automatique des documents.

3.1. L'indexation manuelle

Dans l'indexation manuelle, une personne, connaissant de préférence le sujet et le domaine auquel le document fait référence, doit assigner des mots clés à un document. Il est fait usage d'une liste de termes d'indexation, de règles d'indexation et de feuilles de travail spéciales afin d'enregistrer cette information.

Le choix du vocabulaire que l'indexeur utilise est, soit libre, soit contrôlé, c'est-à-dire qu'il devra utiliser les mots clefs se trouvant dans une liste établie au préalable. Salton affirme que dans la plupart des situations, un vocabulaire contrôlé est utilisé.

L'avantage du vocabulaire libre est évident : l'indexeur peut attribuer aux documents les mots clés les plus appropriés. Le désavantage est bien sûr que nous puissions assister à une prolifération des termes utilisés et que ceux choisis soient trop spécifiques. Une recherche sur un mot clé ne donnerait finalement que quelques documents alors que d'autres, pertinents, existent peut-être dans la collection de documents.

Le vocabulaire d'indexation contrôlé a un avantage : il limite le nombre de mots clés différents utilisés dans un système, en éliminant les mots inappropriés (fautes d'orthographe, synonymes, ...). Il permet aussi de regrouper des mots clés potentiels sous une catégorie générique, augmentant les chances de trouver un nombre important de documents pertinents. Son désavantage est qu'il ne s'adapte pas (sans concertation avec les personnes qui ont établi cette liste) à l'évolution de la technique et à la signification des termes utilisés. Cependant, il est nécessaire d'être familiarisé avec le processus d'indexation pour pouvoir formuler les requêtes adéquates les plus efficaces possibles.

Finalement, distinguons encore l'usage de mots uniques comme mots clés de l'usage de mots composés ou de termes dans leur contexte (exemple : *automatic information retrieval*). Les systèmes manuels utilisent de préférence des termes dans leur contexte tandis que les systèmes automatiques utilisent des termes isolés de leur contexte.

3.2. L'indexation automatique

3.2.1. L'indexation non pondérée

Outre l'attribution manuelle des mots clés comme représentants de document, il existe des techniques d'attribution automatique des mots clés. Cette attribution est le résultat d'une analyse automatique du texte.

Celle-ci a pour but, à partir du texte complet d'un document, d'un résumé (*abstract*), d'un titre ou d'une série de mots, de choisir les termes les plus pertinents pour représenter le texte original.

L'idée fondamentale de l'indexation automatique revient à Luhn [LUHN, 1958] : « *It is here proposed that the frequency of word occurence in an article furnishes a useful measurement of word significance. It is further proposed that the relative position within a set of words having given values of significance furnish a useful measurement for determining the significance of sentences. The significance factor of a sentence will therefore be based on a combination of these two measures.* ».

L'hypothèse est donc que la fréquence d'apparition d'un mot dans un texte est représentative de l'importance de ce mot et du concept dans le document. En combinant cette mesure de l'importance des mots, il est également possible de tirer une mesure de l'importance d'une phrase dans un texte et de déterminer un ensemble de mots clés qui formeront le représentant du document.

Une autre hypothèse formulée est que l'adéquation d'un mot à servir d'index est proportionnelle à sa fréquence d'apparition suivant la règle suivante : si le mot est très fréquent, alors l'utilisation de ce mot comme terme d'index n'apporte rien à la représentation du document. Il en est de même si le mot est peu fréquent dans le document. Les mots clés considérés sont ceux qui ont une fréquence comprise entre une borne inférieure et supérieure à déterminer.

Le problème associé à ces bornes inférieures et supérieures est qu'il n'existe pas de valeurs "miracles" qui conviennent à toutes les collections de documents. Elles doivent être établies par expérimentation ou tâtonnements successifs.

Quelles sont les étapes dans la génération automatique de ces mots clefs ?

- l'élimination des mots fréquents ("stop words");
- le retrait des terminaisons communes pour aboutir à des racines communes à plusieurs mots. Par exemple, les mots *computer, computing* et *computerize* seraient réduits à la racine commune *compute;*
- la détection des racines identiques et l'agrégation des fréquences d'occurrence pour les mots ayant une racine commune.

L'application d'une borne supérieure à la fréquence des mots est ici l'élimination des mots fréquents. L'élimination de ces mots est effectuée car ils ont un contenu informationnel très faible, voire nul. L'intérêt de cette élimination est aussi la réduction assez importante de la taille des documents considérés, ce qui réduit l'occupation du représentant du document en mémoire et réduit le temps nécessaire au traitement du document. A titre d'illustration, voici un extrait de la liste des mots fréquents utilisée dans notre prototype (Tableau 1).

alone	amount	anywhere	automatically	becoming
along	an	apart	available	been
already	and	appear	away	before
also	another	appropriate	awfully	beforehand
although	any	are	b	behind
always	anybody	around	back	being
am	anyhow	as	be	below
among	anyone	aside	became	beside
amongst	anything	associated	because	besides
amoungst	anyway	at	become	best

Tableau 1 : extrait de la listes des mots fréquents utilisé dans le prototype

Le retrait des suffixes est un problème assez complexe en soi. Porter [PORTER, 1980] a étudié ce problème et a proposé un algorithme pour lequel il donne peu de justifications pour les différentes étapes et les règles à appliquer.

Nous devons ajouter que l'algorithme de retrait de suffixes est dépendant de la langue utilisée dans les documents. Celui de Porter est spécifique à l'anglais. Avant de considérer l'utilisation de ce type d'algorithme dans un système pour

une autre langue, il faut se demander s'il est possible de construire un algorithme analogue pour cette langue et si oui, quelle en sera l'efficacité. Intuitivement, nous pensons que l'application de ce procédé est envisageable pour la langue française, mais nous voyons certaines difficultés dans les langues allemande et néerlandaise par exemple (pour rester dans des langues européennes anglo-saxonne ou latine). Une difficulté des langues néerlandaise et allemande est que beaucoup de mots sont composés de mots simples. La question est de savoir jusqu'où peut-on retirer les terminaisons sans nuire au sens de la racine ?

Frakes [FRAKES, 1992] propose une implémentation de l'algorithme de Porter. Voici quelques exemples de règles extraits de cette implémentation utilisée dans notre programme (Tableau 2).

Suffixe	**Remplacé par**
sses	ss
icate	ic
active	(supprimé)
biliti	ble

Tableau 2 : exemples de règles de substitution des suffixes pour l'obtention des racines

Pour retirer les suffixes des mots, il est nécessaire de déterminer des conditions « minimales » pour qu'un mot puisse être traité de la sorte. Par exemple, éliminons le suffixe « *ual* ». Pour le mot « *factual* », le retrait de la terminaison a un sens. Tandis que pour le mot « *equal* », la racine obtenue n'a plus aucun sens. Il est nécessaire de déterminer des règles qui permettent de retirer les suffixes tout en gardant une racine sensée.

Une hypothèse supplémentaire au sujet des racines communes est que, si plusieurs mots distincts ont la même racine, les mots se réfèrent alors aux mêmes concepts. Evidemment, il s'agit d'une simplification assez grossière mais qui ne diminue en rien l'efficacité de l'*information retrieval* [vanRIJS, 1989, chap. 2].

Appliqué à un document, le résultat de l'algorithme de réduction des mots est un ensemble de racines correspondantes. Une racine sera utilisée comme mot clé dans le représentant du document pour autant qu'un des mots réduits à cette racine apparaisse avec une fréquence supérieure à un seuil donné.

Voici un petit texte auquel a été appliqué l'algorithme de réduction à des racines :

```
HYPERTEXT AND LITERATURE

Hello.
I am currently a senior finishing a B.A. in English at Emory
University in Atlanta, GA.  I am planning on continuing into
Graduate School after this.  I am looking for Grad Schools with
programs involving the study of Literature, using, within the context
of, or involving hypertext and hypermedia.  The program does not
necessarily have to be part of a humanities or liberal arts program.
It could be communications, publications, etc.  If anyone out there
knows of such programs or is affiliated with one, please write to me
with information on how I may contact the program.
                              Thank You,
                              dmallon@emory.edu
```

Le Tableau 3 reprend les racines obtenues.

affili	art	atlanta	commun
contact	context	continu	dmallon
edu	emori	english	finish
ga	graduat	hello	human
hypermedia	hypertext	inform	involv
know	liber	literatur	look
necessarili	plan	program	public
school	senior	studi	thank
univers	write		

Tableau 3 : racines obtenues pour le texte précédant

On remarque que le nombre de mots a été fortement réduit. On est passé de 104 mots à 34 racines, ce qui est un gain appréciable. Le gain est fortement influencé par la taille du texte et par le vocabulaire utilisé.

Il y a bien sûr un débat pour savoir si l'indexation automatique est qualitativement supérieure ou inférieure à l'indexation manuelle. Au-delà de ce débat, il existe des résultats expérimentaux qui tendent à prouver que l'indexation automatique telle que décrite ci-dessus a une efficacité suffisante et qu'en tout cas, un processus plus élaboré d'indexation automatique n'obtient

qu'une amélioration marginale non significative. C'est la manière la plus simple de générer des représentants de document en *information retrieval.*

Salton [SALTON, 1970] a résumé ses conclusions à propos de l'indexation automatique : « ... *on the average the simplest indexing procedures which identify a given document or query by a set of terms, weighted or unweighted, obtained from document or query text are also the most effective.* ».

3.2.2. L'indexation pondérée

La procédure ci-dessus décrit l'attribution de mots clés à des documents selon un processus automatique. Celui-ci peut être adapté de façon à fournir une valeur numérique : la pondération. Celle-ci qui rendra compte de l'entropie d'information d'un terme au sein d'un document ou d'une collection de documents.

Dans un document, tous les mots n'ont pas le même poids en terme d'informations apportées au document. Par exemple, les articles, les pronoms et les adverbes ont un contenu informationnel faible par rapport à des verbes ou à des noms. Dans un système où interviendrait un indexeur professionnel, celui-ci pourrait facilement donner un *rang* à un terme d'index. Il est plus difficile de réaliser ce classement par un processus automatique.

Sparck Jones, Salton et Yang ont relié les facteurs d'exhausitivité, de spécificité à des statistiques portant sur la collection de documents. Ils supposent que l'exhaustivité d'un index est liée au nombre de termes utilisés comme mots clés pour un document et que la spécificité d'un terme utilisé dans un mot clé est proportionnelle au nombre d'occurrences de ce terme dans la collection de documents.

Ce qui est primordial dans cette relation, c'est la distribution des termes d'index dans la collection de documents. Donc, en attribuant aux mots clés des informations sur la distribution de ces termes dans les documents, le problème

lié au contrôle de l'exhaustivité et de la spécificité des termes d'index est en quelque sorte résolu.

Selon l'idée originale de Luhn, la puissance discriminatoire d'un terme d'index pour un document est fonction de sa fréquence d'apparition dans ce document. Luhn proposait que les termes d'index soient sélectionnés sur base de leur fréquence.

Cette information sur la fréquence peut servir à sélectionner un terme comme mot clé, mais également à lui attribuer un poids. C'est d'ailleurs une méthode souvent employée pour pondérer un terme d'index. Cela peut sembler contredire l'hypothèse de Luhn, supposant l'existence d'une borne supérieure de fréquence au-delà de laquelle la puissance discriminatoire du terme diminue fortement. L'hypothèse de Luhn peut être rendue valide en affirmant que la borne supérieure en question est justement la fréquence maximale d'un terme dans le document.

Une autre pondération possible est celle d'un des termes d'index, non plus en fonction de sa fréquence dans un document mais bien en fonction de sa fréquence dans la collection de documents. Cette dernière manière de procéder a été expérimentée par Sparck Jones.

Dans une collection de N documents et pour n apparitions d'un terme, celui-ci se voit attribuer un poids de $log(N/n)+1$. L'utilisation de cette pondération permet d'atteindre une meilleure efficacité lors d'une requête par rapport à un système où les termes d'index ne sont pas pondérés.

La différence entre ces deux méthodes d'indexation peut être résumée de la manière suivante : la pondération par la fréquence au sein du document met l'accent sur une description du contenu du document tandis que la pondération par la fréquence dans la collection met l'accent sur la discrimination des documents dans la collection.

Salton cite un critère pour déterminer si un terme d'index est bien choisi. Un bon terme d'index est celui qui rend plus dissemblable le document dans la

collection de documents. Inversement, un mauvais terme est celui qui rend similaire les documents de la collection.

Les termes apparaissant avec une fréquence élevée sont des mots clés peu adéquats. Ceux apparaissant avec une fréquence moyenne, surtout si les distributions de ces mots dans les documents sont irrégulières, sont de bons termes d'index. Les mots distribués irrégulièrement sont également de bons termes d'index. Finalement, les mots très rares ont une utilité assez faible. Dès lors, un classement de l'utilité des mots comme termes d'index selon leur fréquence d'apparition (Figure 1) peut être établi.

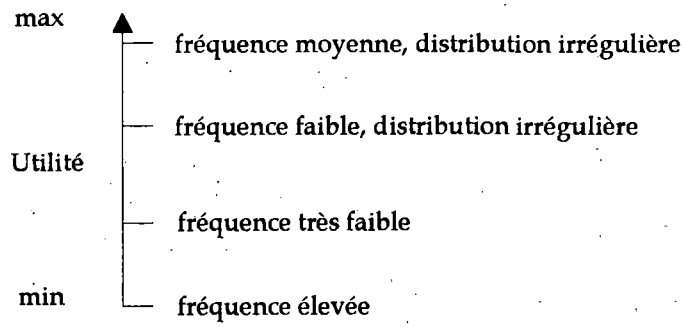


Figure 1 : utilité des termes selon leur fréquence d'apparition dans la collection de documents

La similarité est quantifiée par une valeur de discrimination du terme. Si ce terme d'index n'est pas utilisé, cette valeur mesure l'augmentation ou la diminution de la similarité des documents. Un bon terme d'index est un mot clé qui, s'il est retiré du représentant du document, augmente la similarité des documents dans la collection. Autre façon de le voir, un bon terme d'index est un mot clé, qui, s'il est ajouté au représentant du document, augmente la dissimilarité des documents dans la collection.

Yu and Salton [YU, 1977] ont résumé les points principaux de l'efficacité de l'*information retrieval* muni d'information sur les fréquences d'apparition. Quelques points de leur conclusion :

- l'utilisation de la pondération par l'inverse de la fréquence d'occurrence dans le document est plus efficace que l'utilisation d'une autre pondération;

- une technique d'élimination des suffixes pour les termes apparaissant rarement augmente le taux de rappel du système;
- la combinaison des termes de fréquence élevée en phrases augmente la précision du système.

3.3. L'usage de thesauri

L'efficacité des systèmes d'*information retrieval* peut être accrue en utilisant des thesauri dans le processus de sélection des termes d'index. Il en existe deux sortes.

Les premiers sont ceux qui relient des mots interchangeables, c'est-à-dire que ces termes sont repris dans une classe de mots sémantiquement interchangeables.

Les autres reprennent des relations hiérarchiques entre les termes d'index. Les relations exprimées dans cette sorte de thesauri sont d'ordre sémantiques, c'est-à-dire qu'elles définissent des synonymes, des termes plus généraux ou des relations particulières entre mots.

Des méthodes automatiques pour créer des thesauri ont été développées. Elles se basent sur des statistiques pour identifier les termes équivalents. Un terme *a* est interchangeable avec un terme *b* si l'utilisateur accepte que le système restitue des documents contenant le mot *b* alors que la requête portait sur le terme *a*.

Une manière de trouver les termes équivalents est de regarder les documents dans lesquels les termes apparaissent. Si les mêmes termes apparaissent dans les mêmes documents, alors il y a une chance pour qu'ils aient trait au même sujet et qu'ils puissent donc être substitués. Il est donc possible de définir des classes de mots qui semblent interchangeables.

Les deux stratégies pour exploiter ces thesauri sont :

- remplacer chaque mot clé dans un document par le terme générique de la classe à laquelle il appartient;
- remplacer chaque mot clé par tous les mots clés de la classe à laquelle il appartient.

Sparck Jones [SPARK, 1971] a rapporté un grand nombre d'expériences dans lesquelles de telles techniques ont accru l'efficacité de l'*information retrieval*. Mais d'autre part, Minker *et al.* [MINKER, 1972] ont montré que, au contraire, ces méthodes peuvent, dans certains cas, réduire l'efficacité de la recherche.

4. La classification automatique des documents

4.1. Introduction

La classification en *information retrieval* est importante car elle répond au besoin de regrouper les documents de manière à rendre les traitements plus efficaces, ou encore de construire automatiquement un thesaurus. L'adéquation d'une classification ne pourra être vérifiée que lors des différentes recherches et des performances de celles-ci.

Les méthodes de classification en *information retrieval* portent sur deux choses :

- le regroupement des mots clefs;
- le regroupement des documents.

Notre propos portera ici sur la classification des documents.

Salton [SALTON, 1968A] explicite le besoin de classification des documents : « *Clearly in practice it is not possible to match each analysed document with each analysed search request because the time consumed by such operation would be excessive.* ».

La classification de documents peut être définie comme étant le regroupement de documents au sein de classes; les documents perdent - jusqu'à un certain point - leur identité. De plus, la classification d'un document implique que celui-ci est identifié par la classe à laquelle il appartient et que les traitements des documents au sein d'une classe seront identiques tant qu'un document spécifique n'est pas traité. Les documents sont regroupés, non pas parce qu'ils sont

nécessairement liés, mais plutôt parce qu'il est vraisemblable que ces documents seront recherchés ensemble.

La recherche dans une telle situation se passera de la manière suivante : une recherche sur les représentants de groupe déterminera si le groupe de documents est pertinent par rapport à la requête et, dans un deuxième temps, au sein des groupes considérés comme pertinents, une recherche sur chaque document déterminera si les documents de cette classe sont pertinents ou non.

Il existe des classes exclusives ou des classes qui se recouvrent partiellement. Cette distinction a des répercussions théoriques et pratiques importantes. Dans les classes qui se recouvrent partiellement, l'efficacité de la recherche est fortement réduite. D'autre part, une telle solution est sémantiquement plus riche car elle permet au document d'être classé selon un nombre plus significatif de ses attributs.

4.2. La classification statistique

Dans un premier temps, nous présenterons les mesures qui permettent d'établir une classification de documents. Nous évoquerons les différentes propriétés des méthodes de *clustering*. Trois méthodes seront ensuite présentées.

4.2.1. Similarité et dissimilarité

La classification statistique se base sur l'existence d'une mesure de la similarité des documents, c'est-à-dire une mesure qui indique dans quelle proportion deux documents sont semblables.

Avant de présenter les différentes mesures rencontrées dans la littérature, nous proposons d'utiliser les notations suivantes :

X et Y sont deux ensembles de mots clés (vecteurs) représentant deux documents.

|X| et |Y| représentent le nombre de mots clés des deux documents

$X \cap Y$ est l'ensemble des mots clés communs aux deux documents

$X \cup Y$ est l'ensemble des mots clés pour les deux documents

$|X \cap Y|$ est le nombre de mots clés communs aux deux documents

$|X \cup Y|$ est le nombre de mots clés pour l'ensemble des deux documents

Une première mesure rend compte du nombre de mots clés en commun. Cette première mesure est donnée par $|X \cap Y|$.

Cette mesure est cependant mal choisie car elle n'est pas normalisée, c'est-à-dire qu'elle ne fait pas référence à la taille des documents ou à la taille des vecteurs de mots clés de ces documents. Il faut donc introduire une normalisation qui permette de définir en terme relatif la similarité des documents.

Voici quelques coefficients de similarité rencontrés dans la littérature :

- Coefficient de Dice : $2\frac{|X \cap Y|}{|X| + |Y|}$
- Coefficient de Jaccard : $\frac{|X \cap Y|}{|X \cup Y|}$
- Coefficient cosinus : $\frac{|X \cap Y|}{\sqrt{|X|} \cdot \sqrt{|Y|}}$
- Coefficient de recouvrement : $\frac{|X \cap Y|}{\min(|X|, |Y|)}$

Une autre forme de mesure est la dissimilarité qui peut être définie de manière mathématique. Il est à remarquer que la mesure de dissimilarité peut être transformée en mesure de similarité de la façon suivante

$$similarité = \frac{1}{1 + dissimilarité}$$

mais que l'inverse n'est pas toujours vrai.

La définition mathématique de la dissimilarité est donnée ci-dessous :

Soit P, l'ensemble des objets à classer. D, la fonction de mesure de la dissimilarité de PxP qui a pour résultat un nombre réel positif ou nul.

$$D(X, Y) >= 0 \text{ pour tout } X, Y \in P$$
$$D(X, X) = 0 \text{ pour tout } X, Y \in P$$
$$D(X, Y) = D(Y, X) \text{ pour tout } X, Y \in P$$

Elle peut être vue comme une fonction de distance. Et de fait, beaucoup de mesures de dissimilarité satisfont l'inégalité du triangle.

$$D(X, Y) \leq D(X, Z) + D(Y, Z)$$

Un exemple qui satisfait la définition mathématique ainsi que l'inégalité du triangle est la fonction :

$$\frac{|(X \cup Y) - (X \cap Y)|}{|X| + |Y|}$$

Cette fonction est équivalente à 1 - *coefficient de Dice* = $1 - 2\frac{|X \cap Y|}{|X| + |Y|}$

D'un point de vue conceptuel, les documents peuvent être représentés par des vecteurs binaires de mots clés dans un espace euclidien à *n* dimensions où *n* est le nombre de mots clés distincts pour les deux documents. Un vecteur binaire est

un vecteur où la pondération donnée à un terme est égale à 1 si le terme est présent dans le document, 0 sinon.

Alors la fonction

$$\frac{(X,Y)}{\|X\|.\|Y\|}$$

peut être interprétée comme étant le cosinus de l'angle qui sépare les vecteurs des documents X et Y dans cet espace euclidien. (X, Y) est *l'inner product* et $\| \ \|$ est la longueur du vecteur.

Cette représentation est également généralisable aux mots clés pondérés. Si l'espace est euclidien, qu'un vecteur de mots clés pondérés (x_1, ..., x_n) et (y_1, ..., y_n) est déterminé respectivement pour les documents X et Y, alors la formule du cosinus est définie sous cette forme :

$$\frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x^2}\sqrt{\sum_{i=1}^{n} y^2}}$$

4.2.2. La classification (*clustering*)

Une hypothèse fondamentale des méthodes de classifications basées sur le *clustering* est que les documents d'une même classe sont relatifs à la même requête. D'une façon similaire, les documents pertinents sont plus similaires entre eux qu'entre un document pertinent et un document non pertinent. C'est cette séparation entre documents pertinents et non pertinents qui est exploitée dans la classification.

Il est admis que ce genre d'organisation peut conduire à une efficacité plus grande de la recherche par rapport à une recherche strictement séquentielle. L'objet de la création automatique de mots clés est de créer une distance plus importante entre documents, ce qui permet de rendre plus probable le fait de

trouver des documents pertinents et inversement, moins probable de trouver des documents non pertinents.

Pour choisir une méthode de classification, il faut se référer à trois critères théoriques :

- sa stabilité dans le temps face à l'ajout de documents;
- sa stabilité vis-à-vis d'erreurs minimes dans la description des documents;
- la méthode ne doit pas être influencée par l'ordre initial des documents.

Bien que ces qualités soient désirables, il faut reconnaître que les méthodes développées jusqu'à présent ne satisfont pas l'ensemble de ces trois critères et qu'il s'agit plutôt de méthodes *ad-hoc* plus efficaces.

Ainsi, l'efficacité de la méthode de classification est un souci majeur. Elle peut se mesurer à deux niveaux : la performance de la recherche et l'utilisation de la mémoire de masse. Elle est également une propriété de l'algorithme de *clustering*. Alors qu'il faudrait idéalement différencier la *méthode* de l'*algorithme* de classification, le fait est que cette distinction, dans le champs de l'*information retrieval*, n'est pas opportune car, bien souvent, la méthode de classification est définie par son algorithme.

Il y a deux approches pour le *clustering* :

- les méthodes qui se basent sur une mesure de similarité entre documents;
- les méthodes qui se basent sur une description des documents, par exemple le vecteur de mots clés.

Nous présenterons également une autre méthode de classification, la méthode du lien unique.

4.2.3. L'approche « similarité entre documents »

Un bon exemple de classification basée sur une mesure de similarité est l'approche « théorie des graphes ». Pour un ensemble de documents à classer, pour chaque paire, le coefficient de similarité est calculé. Le système établit un graphe dont les sommets sont les documents. Entre chaque paire de documents dont la similarité est suffisamment élevée, c'est-à-dire au-dessus d'une borne fixée *a priori*, se dessine une arrête.

Le résultat se présente sous la forme suivante (Figure 2):

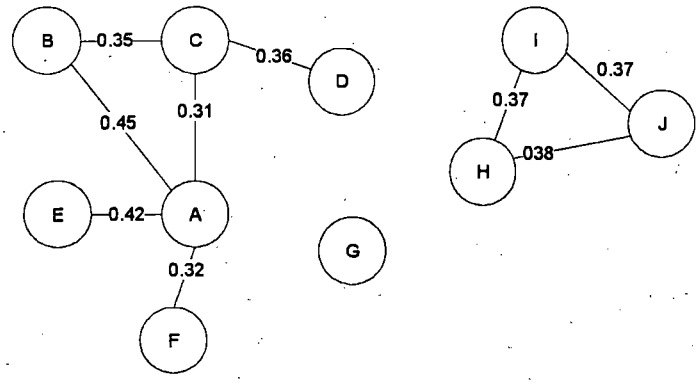


Figure 2 : graphe de dix documents

Le *cluster* est défini en fonction de sa représentation graphique. Par exemple, une chaîne est une séquence d'objets connectés entre eux. Dans le graphe de la Figure 2, les documents A, C et D forment une chaîne.

Ou encore, une composante connexe est un ensemble de documents dont chaque élément est connecté à un autre élément de l'ensemble. Dans le graphe de la Figure 2, les documents A, B, C, D, E et F forment une composante connexe, tout comme les documents H, I et J.

Toujours dans la Figure 2, une autre approche consiste à repérer des sous-ensembles qui satisfont des critères de cohérence (documents A, B, C, D, E et F; H,I et J) et d'isolation (H, I et J; G).

Cette approche « théorie des graphes » a été abandonnée car elle n'est pas assez efficace en temps calcul. Si la collection de documents comporte n documents, il serait nécessaire d'effectuer n^2 mesures de similarité entre documents. Le résultat serait une matrice de taille nxn. De plus, il faudrait exécuter des algorithmes pour repérer les composantes connexes, les chaînes ou les groupes isolés. Cette approche de la « théorie des graphes » a été assez rapidement écartée.

4.2.4. L'approche « description des documents »

Bon nombre de méthodes pour créer des *clusters* ont été développées dans un souci d'efficacité. L'efficacité est apportée en ne calculant plus une mesure de similarité entre documents mais en utilisant directement les représentants de document pour aboutir à la classification finale. Une autre caractéristique est que les méthodes d'*information retrieval* ne tentent pas de découvrir une éventuelle classification mais impose plutôt une classification « convenable ».

Il existe de multiples algorithmes heuristiques. Tous se basent sur l'existence d'un représentant par *cluster*. C'est un objet qui décrit et représente les objets du *cluster*. Il doit être proche, en moyenne, du *cluster* selon une mesure spécifique à l'algorithme. Dans la Figure 3, deux *clusters* sont présents. Chaque classe ainsi déterminée comprend un document « virtuel » R1 ou R2 qui est une sorte de résumé de l'ensemble des documents.

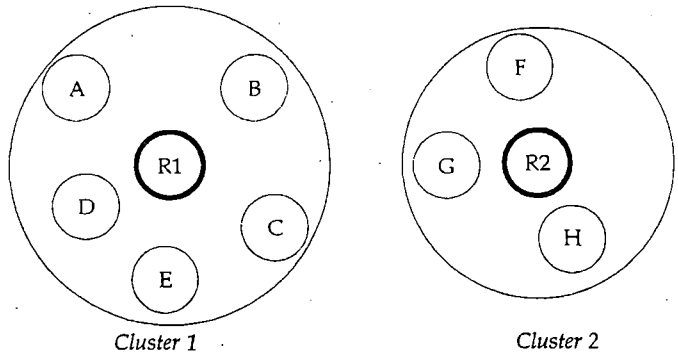


Figure 3 : deux clusters pour huit documents

Ce représentant de classe est, en général, un vecteur de mots clés pondérés. Les pondérations sont calculées pour être une sorte de centre de gravité du *cluster*.

Les méthodes de *clustering* utilisent des paramètres empiriques, tels que le nombre de *clusters* souhaités, la taille minimale et maximale des *clusters*, le contrôle du recouvrement entre classes, ...

Tous ces algorithmes sont itératifs, c'est-à-dire qu'une première classification est établie et qu'elle est améliorée au cours des itérations successives. Bien que la plupart des algorithmes soient construits pour établir un seul niveau de classification, il est possible de les modifier pour fournir des classifications hiérarchiques. Une telle modification consisterait alors à créer des classes de *clusters* ayant des propriétés similaires.

Un algorithme célèbre, développé dans le cadre du projet SMART dont G. Salton est un des parrains, est l'algorithme de *clustering* de Rocchio. Une description complète de l'algorithme de Rocchio se trouve dans [SALTON, 1968B]. Le processus peut être décomposé en quatre étapes :

- la sélection de documents, selon certains critères, qui serviront comme représentants des *clusters*. Les représentants des *clusters* initiaux répondront à des critères garantissant qu'un certain nombre de documents similaires existent pour chacun d'eux dans la collection de documents, ceci afin d'éviter de sélectionner des *clusters* qui resteront vides;
- les documents restants sont assignés à un des représentants ainsi désignés ou alors à un *cluster* poubelle s'il ne s'associe à aucun des représentants

désignés. La règle d'association est basée sur une fonction de similarité qui doit retourner une valeur minimale pour que le document soit associé au *cluster;*

- la troisième étape est itérative et permet d'ajuster les paramètres initiaux afin que la classification établie respecte les spécifications préalables (taille des clusters, ...);
- la quatrième et dernière étape consiste à "nettoyer" la classification obtenue, entre autres en assignant de force les documents du *cluster* poubelle à un autre *cluster*. La réduction ou la suppression du recouvrement de classe fait aussi partie de cette étape.

La plupart des algorithmes tentent de réduire le nombre de passes qui doivent être effectuées pour classifier les documents. Il y a quelques algorithmes qui travaillent en une seule passe. Leur logique est la suivante :

- la description des objets est traitée en série. Le premier objet devient d'office le représentant du premier *cluster;*
- les objets suivants sont comparés aux représentants de *clusters* déja existants.
- l'objet est assigné à un *cluster* ou plus selon une condition sur la fonction de correspondance des documents et des représentants;
- quand un objet est assigné à un *cluster,* le représentant du *cluster* est recalculé;
- si un objet ne rentre pas dans un *cluster* quelconque, alors il devient le représentant d'un nouveau *cluster.*

Un troisième type d'algorithmes travaille de la manière suivante :

- une partition et un ensemble de représentants de *cluster* sont déterminés (par un programme ou un opérateur);
- le traitement ultérieur consiste à allouer les documents restants au *cluster;*
- après cette réallocation, le représentant du *cluster* est recalculé et il remplace l'ancien *si* il est meilleur (selon une mesure moyenne) que le précédent par rapport à l'ensemble des documents.

Les algorithmes qui ont été présentés ici (il y en a d'autres) ont des temps d'exécution de l'ordre de $n \log n$ où n est le nombre de documents à classer. Ce temps d'exécution est à comparer avec celui des méthodes basées sur une mesure de la similarité deux à deux qui a un temps d'exécution de l'ordre de n^2.

Cependant, ces méthodes ont des inconvénients : la classification finale dépend de l'ordre dans lequel les objets sont présentés et les effets d'erreurs dans la description des objets sont imprévisibles.

4.2.5. La méthode du lien unique

Une toute autre méthode de classification permet d'aboutir directement à une hiérarchie. C'est la méthode dite du *lien unique*. Celle-ci utilise en entrée le coefficient de dissimilarité et fournit en sortie une hiérarchie (*dendrogram*) avec une valeur numérique pour chaque niveau.

Cette hiérarchie peut être représentée sous la forme d'un dendrogramme ou d'un arbre tel qu'illustré dans la Figure 4. A chaque niveau de la hiérarchie, un certain nombre de classes peuvent être identifiées. Les classes des niveaux supérieurs englobent les classes des niveaux inférieurs.

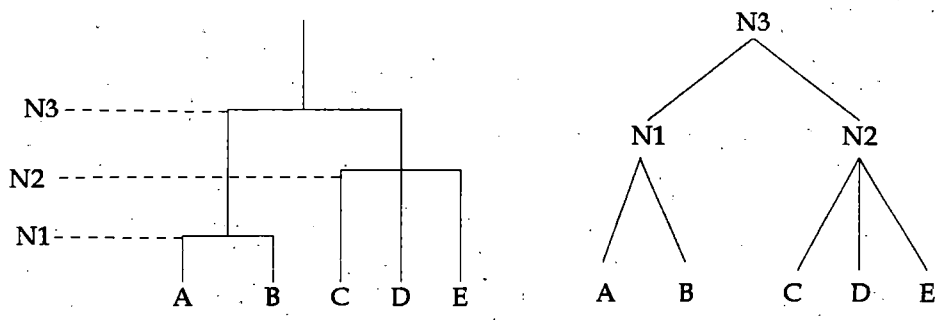


Figure 4 : un dendrogramme à 3 niveaux pour 5 documents et la même hiérarchie sous forme d'arbre

Une telle classification peut être déduite du coefficient de dissimilarité entre documents. Un graphe, le dendrogramme, est alors créé à partir du coefficient de dissimilarité en établissant une arrête entre deux documents si le coefficient est inférieur ou égal à un des niveaux pour lequel le graphe est construit (N=0.1, 0.2, 0.3, 0.4).

Un *cluster* à lien unique est défini comme étant une composante connexe du graphe à un certain niveau. Au niveau 1, nous observons dans la Figure 4 le *cluster* formé par les documents A et B. Au niveau 2, nous trouvons deux clusters formés par les documents {A, B} et {C, D, E}. Au niveau 3, seul le *cluster* formé par les documents {A, B, C, D, E} est observable.

La hiérarchie est établie en faisant varier le niveau de dissimilarité attendu, jusqu'à ce que tous les objets soient repris dans un seul *cluster*. Pour qu'un objet soit repris dans une composante connexe, il lui suffit d'être relié à un seul autre document, d'où le nom de la méthode.

La manière décrite pour construire les classes est inefficace, car elle correspond au calcul d'une matrice de coefficients pour chaque niveau. Mais il existe des méthodes de constructions plus efficaces.

L'arbre qui est produit par la méthode du lien unique est fortement lié à un autre arbre, l'arbre de couverture minimale (qui est également calculé grâce au coefficient de dissimilarité). Dans cet arbre, les noeuds représentent non plus des *clusters*, mais bien des documents à classer. Il est l'arbre de longueur minimale qui connecte tous les documents. La longueur est ici la somme du poids des arrêtes qui relient les documents.

L'arbre de la méthode du lien unique peut donc être dérivé de l'arbre de couverture minimale. Il suffit de retirer de l'arbre de couverture minimale les arcs dans l'ordre décroissant de leur longueur. Après chaque retrait, un ensemble de composantes connexes est obtenu. Il s'agit des *clusters* à lien unique.

L'utilisation de cet arbre n'est pas meilleure que l'utilisation de l'arbre de la méthode du lien unique car il n'y a pas de moyen de modifier, sans calcul exhaustif, un arbre de couverture minimale (ajout, suppression de documents).

4.2.6. Quelles méthodes ? Quelles contraintes ?

Les systèmes qui structurent une collection de documents en niveaux de *clusters* sont appropriés. En effet, le niveau d'un *cluster* peut être utilisé de la même manière qu'un numéro de rang ou que la limite d'une fonction de correspondance dans une recherche linéaire. La sélection d'un *cluster* à un bas niveau hiérarchique apportera un niveau élevé de précision et un niveau faible de rappel. Ce qui est équivalent à une limite peu élevée dans une recherche linéaire. De même, si un *cluster* de haut niveau hiérarchique est sélectionné, nous aurons une faible précision mais un taux de rappel élevé.

Les systèmes strictement hiérarchiques sont appropriés en *information retrieval.* Cela permet d'avoir des stratégies de recherche très performantes et un gain de temps non négligeable à la construction de la hiérarchie (en la comparant à une méthode qui produit des recouvrements entre classes). De plus, la capacité de stockage nécessaire pour conserver cette structure est réduite.

Dans un environnement opérationnel, il faut tenir compte de contraintes quant à la taille de la mémoire centrale, de la mémoire secondaire, au temps CPU. Celles-ci restreignent les choix de la méthode de *clustering,* qui doit entre autres, permettre la classification de documents additionnels sans forcer à reclassifier l'ensemble des documents.

Cette dernière contrainte n'est pas habituellement rencontrée de manière efficace car, bien souvent, les méthodes de classification ne donnent pas les mêmes résultats pour une classification additionnelle que pour une classification complète. Pire, il est parfois nécessaire de reclassifier tous les documents de la collection car les classifications additionnelles ont perturbé de manière importante la classification initiale.

Ces derniers commentaires s'appliquent aux méthodes qui ont un temps d'exécution $n \log n$, qui sont en général utilisées dans les systèmes opérationnels. Il faut donc se méfier des conclusions hâtives. Une méthode, qui a un temps

d'exécution de n^2 mais qui ne nécessite pas un recalcul systématique de la classification suite à l'ajout d'un document, peut être préférable à une méthode en $n \log n$ qui nécessite une mise à jour systématique de la classification en cas de modification de la collection de documents.

De toute façon, les méthodes décrites ci-dessus peuvent être modifiées afin de circonscrire le temps d'exécution en n^2. Dans une première phase, une classification de base à partir d'un échantillon peut être construite en utilisant une méthode en n^2. Dans une deuxième phase, on assigne les autres documents à une des classes trouvées lors de la première étape. Une autre manière de procéder est d'établir une classification grossière dans un premier temps; ensuite, il est possible de raffiner la classification dans un *cluster* par une méthode plus consommatrice de ressources.

5. La recherche d'information

Afin de rechercher l'information dans un système d'*information retrieval*, l'utilisateur dispose d'un mécanisme spécifique au système utilisé. Le système utilise soit des requêtes booléennes (éventuellement étendues), soit une fonction de correspondance. La recherche d'information peut se faire soit en parcourant séquentiellement la collection de documents, soit en utilisant la structure de *clusters* mise en place au préalable.

5.1. Les requêtes booléennes

L'utilisateur soumet sa requête en utilisant une expression booléenne composée de termes et d'opérateurs. Par exemple, « *information and retrieval* » permettra de localiser les documents contenant simultanément les termes *information* et *retrieval* à l'exclusion de tout autre document. Ce type de requête utilise les opérateurs classiques de l'algèbre booléen tels le *et*, le *ou* ou encore la *négation*.

Dans les systèmes utilisant une structure de fichier indexé inverse (à un terme de l'index correspond une liste de documents reprenant ce terme), ce style de requête est facilement réalisable. Il s'agit d'effectuer des opérations ensemblistes sur les listes de documents afin d'obtenir le résultat. Ces opérations ensemblistes sont l'union, l'intersection ou la différence d'ensembles.

Même si ce mode de fonctionnement est aisé à implémenter, il est nécessaire de prévoir des règles syntaxiques et sémantiques pour éviter les requêtes ambiguës et permettre ainsi aux utilisateurs de poser des requêtes complexes. Un exemple de requête qui pourrait être ambiguë est « *information and retrieval or hypertext* ». Si le système n'impose pas des règles syntaxiques et sémantiques, le résultat d'une telle requête peut varier suivant l'interprétation donnée à la requête.

La plupart des systèmes évolués introduisent des opérateurs supplémentaires correspondant aux spécificités du système. Dans le système *Inquiry* développé au *Centre for Intelligent Information Retrieval* de l'Université du Massachusets [CIIR, 1994], des opérateurs portant sur la proximité des termes ont été introduits. De même, comme il s'agit d'un système utilisant des pondérations pour les mots clés, celui-ci propose des opérateurs qui spécifient un poids à attribuer à un terme dans une recherche. Cela permet d'introduire un ordre de préférence dans les termes de la requête.

Une requête complexe est « *#sum (reform #2(health care))* » qui exprime une demande portant sur les documents comportant le terme *reform* et les deux termes *health* et *care* séparés au maximum par deux mots.

La requête « *#wsum (3 information 0.5 retrieval 0.5 hypertext 0.7)* » exprime une recherche de documents dont les représentants comprennent les termes de la requête avec une pondération similaire.

5.2. Les fonctions de correspondances

Les systèmes *d'information retrieval* les plus sophistiqués utilisent une fonction de correspondance (*matching*) qui mesure la similarité entre une requête et un document. C'est une mesure analogue à la similarité entre documents mais la fonction de correspondance mesure la similarité entre un document et une requête.

Une simple fonction de correspondance est donnée par

$$M = \frac{2|D \cap Q|}{|D| + |Q|}$$

où D est l'ensemble des mots clés du représentant du document ou du *cluster*, et Q l'ensemble des mots clés de la requête.

La fonction de correspondance utilisée dans le projet SMART est la mesure du cosinus de l'angle formé par les vecteurs de la requête et du document dans un espace à *t* dimensions (où *t* est le nombre de mots clés du document).

L'utilisation d'une fonction de correspondance permet de classer les documents selon leur similarité avec la requête.

5.3. Les méthodes de recherches

Les méthodes de recherches en *information retrieval* sont la recherche séquentielle et la recherche utilisant la structure de *clusters* mise en place au préalable. Nous présenterons ensuite une méthode de recherche interactive qui fera intervenir l'utilisateur.

5.3.1. La recherche séquentielle

Bien qu'il s'agisse d'une méthode de recherche lente, elle est toujours utilisée dans certains systèmes sophistiqués. Elle consiste à balayer la collection de

documents un à un et de comparer le document et la requête. Deux modes de fonctionnement peuvent être envisagés. Le premier consiste à sélectionner les documents pertinents comme ceux dont la mesure de similarité est supérieure à un seuil donné. Le deuxième mode de fonctionnement est de prendre les *r* premiers documents d'un classement des documents selon leur similarité avec la requête.

La difficulté dans ce genre de recherche est la détermination du seuil ou du niveau *r* à partir duquel sont sélectionnés les documents pertinents.

5.3.2. La recherche dans les *clusters*

La recherche dans les *clusters* se base sur l'existence d'un représentant par *cluster*. Elle consiste à comparer les termes de la requête avec les représentants des classes. Nous supposons que si le représentant de *cluster*, comparé à une requête, offre une mesure de correspondance élevée, alors les documents constituants du *cluster* sont pertinents pour la requête. Le problème est donc de choisir une méthode de construction des représentants de classe qui soit la meilleure possible. Mais aucune théorie ne permet de déterminer si une méthode est meilleure qu'une autre.

La recherche se passe de la manière suivante : le système compare la requête aux représentants des *clusters*; si la valeur de la fonction de correspondance est élevée, alors l'hypothèse veut que les documents du *cluster* soient pertinents par rapport à la requête.

Si la classification a abouti à la création d'une hiérarchie, il est possible d'exploiter la hiérarchisation pour accélérer la recherche. Si la classification a généré une hiérarchie sous forme d'arbre, la requête est comparée récursivement avec les branches de l'arbre qui exposent une similarité élevée. Il s'agit d'une approche *top-down*.

Une stratégie de recherche *bottom-up* consiste à localiser un *cluster* qui contient des documents similaires à un document en notre possession. Il est donc

nécessaire de disposer d'un document initial. Mais il ne s'agit pas d'une exigence démesurée dans la mesure où, bien souvent, les utilisateurs désirent localiser des documents similaires à un document déjà en leur possession.

5.3.3. La recherche interactive et le *relevance feed-back*

Un utilisateur d'un système d'*information retrieval* sera dans la plupart des cas incapable d'énoncer en une fois toutes les informations nécessaires à une requête optimale. Afin d'améliorer l'efficacité des systèmes, il a été proposé de faire participer l'utilisateur à la reformulation de la requête [SALTON, 1968B].

La recherche s'effectue de la manière suivante. Une première recherche est effectuée sur base de la requête de l'utilisateur. Les résultats de la recherche lui sont présentés sous une forme résumée (une liste des titres des documents par exemple). Il doit alors indiquer au système quels documents ils considèrent pertinents. Sur base de ces indications, le système modifie la requête initiale et effectue une nouvelle recherche. De proche en proche, l'utilisateur affine l'expression de la requête en indiquant les documents pertinents.

La méthode pour altérer la requête initiale dépend du système utilisé. Par exemple, dans le système *News Retrieval Tool* [SANDERS., 1994], les pondérations w_i des termes de la requête modifiée sont données par la fonction

$$w_i = \log \frac{r_i(N - n_i - R + r_i)}{(R - r_i)(n_i - r_i)}$$

où N est le nombre de documents dans la collection, n_i est le nombre de documents avec une occurrence du terme *i*, R le nombre de documents pertinents dans la collection et r_i est le nombre de documents pertinents avec une occurrence du terme *i*. Cette fonction compare la fréquence d'occurrence d'un terme dans l'ensemble des documents jugés cohérents avec la fréquence d'occurrence de ce terme dans l'ensemble des documents. Donc si un terme apparaît plus fréquemment dans les documents pertinents, un poids important lui sera assigné.

D'autres méthodes d'altération des termes de la requête existent. Elles tendent toutes à accentuer la différence entre les documents jugés pertinents et ceux qui ne le sont pas.

Il a été montré expérimentalement que ce processus de *feed-back* permet d'améliorer sensiblement les performances des systèmes, mais que cette amélioration est difficilement quantifiable et surtout peu généralisable.

Chapitre 3 :
La création des systèmes hypertextes

1. Introduction

La création des systèmes hypertextes constitue une problématique importante qui provient de la nature même des systèmes hypertextes, c'est-à-dire de leur capacité à structurer et à présenter un grand nombre d'informations. Un autre problème est que la personne qui construit un système hypertexte ne connaît pas nécessairement le domaine concerné, puisqu'il n'est pas obligatoirement l'auteur des documents destinés à la base d'informations hypertextes.

Dans ce chapitre, nous présenterons un certain nombre de recommandations, pour les auteurs de systèmes hypertextes, concernant le contenu de la base d'informations. Nous évoquerons l'importance des outils auteurs. Enfin, nous continuerons par une présentation des méthodes et des outils qui permettent de transformer automatiquement ou semi-automatiquement des textes existants en systèmes hypertextes.

Une dernière remarque : nous avons considéré, dans ce mémoire, la création de systèmes hypertextes à partir de documents composés uniquement de texte. Pourquoi se limiter à des documents aussi « primitifs » alors que les systèmes informatiques actuels manipulent facilement les images, le son ou encore la vidéo ? Parce que nous ne disposons pas de mécanismes pour comparer les sons, les vidéo ou les images. La seule technique pour indexer ces différents média est l'attribution d'une description du média sur base de mots clés. Le lecteur intéressé trouvera dans [McCALLUM, 1989] des informations complémentaires sur l'indexation d'informations multimédia.

2. Quelques recommandations

Nielsen [NIELSEN, 1990, p. 163] fait remarquer que les auteurs ont, en général, quelques idées sur la manière dont ces systèmes fonctionnent mais, que ces idées sont souvent superficielles. Le plus regrettable, c'est que les auteurs n'ont pas appris à structurer correctement l'information dans un réseau hypertexte. L'auteur doit passer d'une organisation séquentielle (texte traditionnel) à une organisation hiérarchisée complexe.

Toujours selon Nielsen, quelques idées simples doivent présider à la création d'un système hypertexte :

- les noeuds doivent être centrés sur un sujet bien précis. Le sujet et l'intitulé du noeud seront alors parfaitement identifiés dans la structure hypertexte.
- la lecture sur un écran d'ordinateur étant plus lente que la lecture sur un support conventionnel, les noeuds doivent être plus concis qu'un texte traditionnel. Les systèmes hypertextes facilitent d'ailleurs cette façon de procéder, car ils permettent de découper un thème en sujets subordonnés.
- la structure de l'application hypertexte doit être aussi claire que possible. Une structure claire réduira la sensation de désorientation et facilitera la

navigation. Même si l'auteur ne contrôle pas la lecture de l'utilisateur, il peut indiquer un certain nombre de priorités dans l'ordre de lecture.

- finalement - et peut-être le plus important - c'est l'expérience qui va nous montrer ce qui marche et ce qui ne marche pas dans les systèmes hypertextes.

3. Les outils auteurs

Les auteurs souhaitent, à l'image des utilisateurs, disposer d'un système convivial pour la création d'une application hypertexte.

Les systèmes auteurs hypertextes devraient comprendre des outils comme une vue par diagramme du réseau hypertexte, des outils d'aide à l'écriture (dictionnaire, grammaire), des outils de gestion des liens (en particulier pour éviter les liens pendants). De plus, il ne faut pas oublier que les systèmes hypertextes sont des cibles privilégiées pour la collaboration de plusieurs personnes. Il faut donc des outils permettant de gérer ces aspects de collaboration. Les systèmes hypertextes se prêtent bien à l'écriture en groupe, car par nature, ils sont découpés en noeuds.

Il est tentant de faire le parallèle entre systèmes auteurs hypertextes et les logiciels auteurs multimédia. Goffinet [GOFFINET, 1995, pp. 5-9] établit une classification des logiciels auteurs multimédia suivant leur puissance et la métaphore utilisée. Il est à remarquer que les systèmes auteurs multimédia sont plus riches que les systèmes hypertextes. Ils intègrent les fonctionnalités des systèmes hypertextes, mais proposent des facilités pour la manipulation des images et du son.

A l'instar d'une table de montage, le système auteur hypertexte devrait permettre à l'auteur de rassembler les différents documents que son système va comprendre. Il définirait alors interactivement les liaisons entre les documents ainsi que les commandes additionnelles de navigation, au moyen de boutons par exemple.

La société Eastgate commercialise le logiciel *Storyspace* qui répond à ces critères [EASTGATE, 1995A]. Selon le fascicule publicitaire que nous avons trouvé sur Internet, il s'agit d'un logiciel orienté vers l'écriture de documents complexes. Celui-ci permet de rassembler toutes les informations du système hypertexte et de définir interactivement les liens entre noeuds. Il permet également de présenter un plan graphique du système hypertexte (Figure 1) .

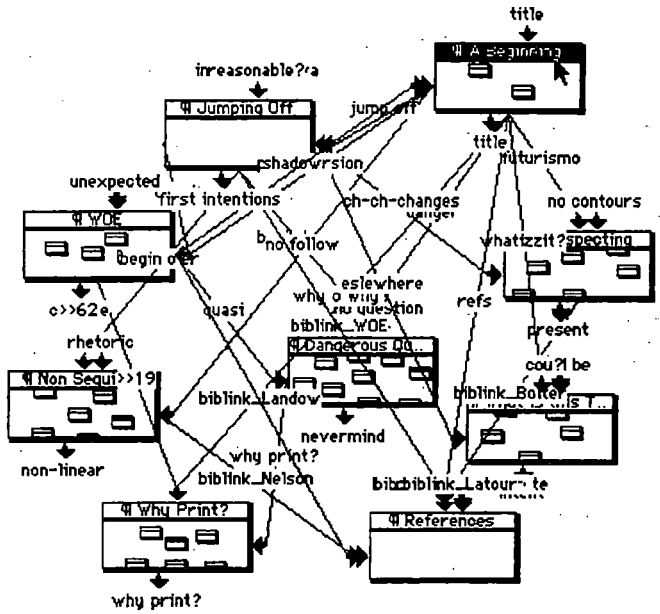


Figure 1 : plan de l'environnement immédiat dans Storyspace[EASTGATE, 1995B]

Ce plan reprend tous les noeuds et liens du document. Une grande partie des liens sont cachés car il s'agit d'une vue de haut niveau de la base d'informations hypertextes. Le logiciel autorise une manipulation directe des noeuds et des liens. Ian Feldman [FELDMAN, 1995] critique cette représentation en affirmant qu'elle est touffue, qu'elle ne correspond pas à des règles ergonomiques et qu'il y a peu de chance qu'elle soit utilisable par des personnes inexpérimentées dans les systèmes hypertextes.

Ce mode de travail intégré procède d'une démarche *bottom-up* (un assemblage de pièces existantes), alors que, selon Nielsen [NIELSEN, 1990, p. 168], une approche *top-down* est la manière recommandée pour construire une application

hypertexte. Les auteurs devraient partir d'un synopsis du système hypertexte et le raffiner au fur et à mesure du développement du système.

Tout comme il existe des systèmes auteurs basés sur des langages de *scripting*, des systèmes hypertextes basés sur l'utilisation d'un langage de haut niveau existent. Par exemple, *HyperCard* d'Apple. Si le langage de *scripting* est insuffisant, il existe des logiciels auteurs qui font appel à des langages de bas niveau pour la réalisation de l'application. Nous faisons ici référence à *Gina* (*Generic Interactive Navigator*), développé au Centre de Recherche Public Henri-Tudor. Ou encore *ScriptX* de Kaleida Labs qui offre un langage orienté objet de bas niveau.

4. La conversion de documents existants

Les systèmes hypertextes sont des outils très intéressants quand il s'agit d'organiser et de présenter de l'information. Cependant, il ne faut pas oublier que la grande majorité des documents sont des documents papiers ou sous une forme non-informatisée (sur microfilms par exemple). Dans le meilleur des cas, nous disposons des textes dans un format électronique. La problématique que nous allons examiner est celle de la conversion des documents textuels en un système hypertexte.

La conversion d'un texte en un système hypertexte consiste à découper le texte existant en une série de noeuds et de créer les liens entre ces noeuds. Il peut s'agir de liens structurels (suivant, précédant, parent) ou de liens sémantiques.

Nous devons distinguer deux classes de conversion : la conversion de documents structurés et la conversion de documents non structurés. Les documents structurés sont des textes qui sont, soit découpés en sections bien définies (par exemple : chapitre, section, sous-section, sous-sous-section, ...), soit encodés de manière appropriée pour mettre en évidence une structure.

Nous pouvons également distinguer les systèmes hypertextes statiques des systèmes hypertextes dynamiques. Une base d'informations hypertextes statique comprend tous les noeuds et liens, ceux-ci sont fixés une fois pour toute. Un système hypertexte est dynamique si tout ou une partie des liens sont déterminés à l'exécution du système. La détermination des liens peut être, soit le résultat de l'analyse d'une structure ou d'un encodage particulier, soit le résultat d'une analyse statistique ou sémantique.

4.1. Les documents structurés

Si le texte est déjà dans un format adéquat ou si la structure du texte est assez régulière, la conversion peut se faire automatiquement avec un programme qui détermine la constitution du texte sur base de la structure (titre, numérotation des titres, découpe en paragraphes, ...). Cette conversion automatique est souvent parachevée par un travail manuel pour établir des liens complémentaires de navigation (index, table des matières, ...). Nous présenterons trois exemples de conversion de documents structurés issus de [NIELSEN, 1990, pp. 173-179].

4.1.1. *IDEX*

La société Office Workstation Limited qui a développé et commercialisé le système hypertexte *Guide,* a développé un système appelé *IDEX*. *IDEX* travaille avec des documents encodés dans le format SGML (*Standard Generalized Markup Language*). Ce standard utilise un marquage du texte pour spécifier la signification d'une partie du texte (par opposition à son apparence). Ces marquages portent sur la structure du document (paragraphe, section, ...)

IDEX utilise cette information pour « déplier » en quelque sorte ce texte dans un format hypertexte et reproduit, dans le système hypertexte, une structure hiérarchique équivalente à celle du texte.

Dans la mesure où cette conversion n'est pas entièrement automatisée, l'auteur peut introduire lui-même des liens dans l'application hypertexte.

Certains auteurs font la différence entre des liens « objectifs », c'est à dire qui découlent directement du texte original, et les liens « subjectifs » qui sont ajoutés parce que l'auteur estime qu'ils sont pertinents.

4.1.2. *The Manual of Medical Therapeutics*

Ce livre de 500 pages, provenant de l'Université de Washington, contient des recommandations pour le diagnostic médical et les thérapies. Il est constitué sur base d'une très forte hiérarchie.

Exemple :

Chapter 6 : Heart Failure
V. Digitalis
D. Digitalis Toxicity
3. Treatment
c. Ventricular Arrhythmias

Grâce à cette structure très régulière, la construction d'un système hypertexte statique correspondant est aisée. Chaque sous-section est projetée dans un noeud portant pour identifiant le numéro de section (i.e. 6.V.D.3.c). Un titre est créé artificiellement en utilisant les 6 premiers mots de la section. Ce n'est pas nécessairement optimal comme convention, mais les termes utilisés sont suffisamment expressifs pour un praticien de manière qu'il puisse comprendre de quoi il s'agit. Pour une section, le système sait recréer les liens vers les têtes de sections et les sous-sections, et vers les noeuds parents de cette section. De plus, le système permet d'effectuer des recherches sur des mots clés.

Afin de satisfaire les besoins des utilisateurs, ceux-ci peuvent effectuer des requêtes. Le moteur hypertexte est muni d'extensions qui utilisent des techniques d'*information retrieval*. A la suite d'une requête, une liste de sections est présentée à l'utilisateur. Ces sections sont classées suivant une pondération, laquelle est le résultat d'un calcul qui associe le poids intrinsèque du noeud auquel il est fait référence et le poids des noeuds auxquels il est connecté.

4.1.3. *The Oxford English Dictionary*

Cette bible de la langue anglaise qui comporte 12 volumes (publiés de 1884 à 1928) et un supplément de 4 volumes (publié de 1972 à 1976), comprend 252.259 entrées pour les 12 volumes principaux et 69.372 entrées pour le supplément. Le texte de l'*Oxford English Dictionary* est encodé dans le format SGML afin de séparer les mots de leur définition, de l'étymologie, des citations, ... Au lieu de recourir à un procédé de lecture optique, toutes les entrées du dictionnaire ont été encodées manuellement afin d'inclure ces marquages.

Le système hypertexte développé utilise l'ensemble du texte (570 Mega de texte continu) et des index extérieurs afin de présenter les informations au lecteur. Il y a une conversion implicite du format texte au format hypertexte à chaque utilisation du logiciel. Il s'agit donc *a priori* d'un système dynamique, mais le support CD-ROM ne permet pas de modifier le contenu du dictionnaire.

Pourquoi l'hypertexte est-il avantageux pour une utilisation dans un dictionnaire tel que l'*Oxford English Dictionary* ? Cela tient à la nature même des dictionnaires d'offrir des références vers d'autres mots, pour les synonymes, les antonymes, les citations des auteurs, ... Ce dictionnaire comprend 569.000 références croisées. De même, qu'il contient 2.400.000 citations illustrant les différentes façons dont les auteurs ont utilisé les mots à travers l'histoire.

Suite à la forte variabilité de la taille des entrées, il n'a pas été possible de développer une seule interface simple pour toutes les entrées du dictionnaire. La taille varie entre des définitions de 50 caractères et des entrées de l'ordre de 500 Ko (le verbe *to set*). 5% des mots ont une entrée supérieure à 4.000 caractères, mais cela représente 48% de la taille du dictionnaire.

L'interface pour des définitions complexes est structurée suivant les différents sens donnés au mot (grâce à la présence d'un marquage approprié). Ce mode de structuration est adéquat car l'utilisateur a alors la possibilité de consulter les différents sens qu'il désire. De même, l'interface peut facilement s'adapter à la catégorie d'utilisateur. Un linguiste sera intéressé par l'évolution des termes et

son étymologie tandis qu'un utilisateur « normal » ne s'intéresse pas à ces détails.

En conclusion, cette conversion de l'*Oxford English Dictionary* est une réussite, car la lecture de celui-ci est plus aisée que la lecture du même dictionnaire dans un format traditionnel (papier ou microfiche). De plus, des mécanismes perfectionnés ont pu être introduits afin de satisfaire les besoins des utilisateurs.

4.2. Les documents non structurés

Si un document n'est pas structuré ou encodé de façon adéquate, il est plus difficile de le transformer en un système hypertexte. C'est également valable pour l'organisation d'une collection de documents.

Il existe quatre approches. La première se base sur la reconnaissance de structures ou de mots propres à un domaine (légal, affaire, ...). La deuxième utilise des outils de traitement du langage naturel. La troisième repose sur l'utilisation de réseaux de neurones. La quatrième exploite les techniques d'*information retrieval* afin de découvrir des liens entre documents.

Cette variété d'approches souligne le fait qu'il s'agit d'un domaine en pleine évolution et que la recherche y est active.

4.2.1. Approche lexicale

L'approche lexicale consiste à parcourir des documents et à repérer des structures sur leur seule base lexicale. Nous allons présenter les travaux de Jeffrey Lorenzen [LORENZ., 1994A et B] et de Jack Conrad et Mary Hunter Utt [CONRAD, 1994].

Jeffrey Lorenzen travaille sur la création automatique de liens hypertextes pour des documents légaux. Ces liens sont établis entre les citations d'autres jugements et celui qui est en train d'être examiné. Un premier programme

parcourt les jugements et repère les trois formes que peuvent prendre les citations de jugement. Par exemple, la citation est de la forme :

In State v. Johnson, 635 P.2d 36 (Utah 1981) the court, by way of dicta, suggests...

Cette citation comprend plusieurs éléments dont le nom des parties et une identification du rapporteur. Le programme transforme cette citation en lien hypertexte dans un format similaire au langage HTML utilisé dans le *World-Wide-Web* sur Internet. Cette citation devient :

In State v. Johnson, 635, P?2d 36 (Utah 1981) the court, by way of dicta, suggests...

Une fois les citations transformées de cette manière, l'utilisateur peut utiliser un *browser* pour parcourir les différents jugements et accéder aux cas cités. L'accès à un autre jugement s'effectue sous la surveillance d'un autre programme qui met en correspondance la citation et un document de la base de données si celui-ci est présent. En effet, le système a pu localiser une citation et la convertir dans le format susmentionné, mais il n'est pas sûr que ce jugement soit présent dans la collection de jugements ou dans les volumes des rapporteurs judiciaires.

Donc, le système est éventuellement incomplet. Il se peut très bien que l'activation d'un lien donne lieu à une erreur causée par l'absence du jugement cité.

Cette forme d'analyse est possible dans le cas où les jugements sont fortement structurés et qu'il existe un certain nombre de formes que peuvent prendre les citations d'autres jugements. Mais, même dans ce cas, il existe des variations et des exceptions qui rendent parfois cette reconnaissance problématique.

Les travaux de [CONRAD, 1994] sont d'un autre ordre. Les auteurs ont travaillé sur le repérage d'éléments (*features*) dans une base de données textuelles. Un élément est un mot ou groupe de mots d'une certaine catégorie (nom propre, nom de ville, nom de société, ...) L'implémentation qu'ils ont

développée a pour but la reconnaissance de termes spécifiques à un domaine et des relations identifiées par ces éléments.

Ils décomposent les traitements en trois phases : le repérage des termes spécifiques, la génération des liens directs entre termes et la génération des liens indirects. Les auteurs ont utilisé des articles du *Wall Street Journal* comme base d'information.

En général, l'extraction d'éléments ou de faits a fait l'objet d'études utilisant des traitements du langage naturel et des statistiques. L'extraction de certaines sortes d'éléments demande des techniques sophistiquées et/ou un grand nombre de données servant à l'apprentissage. Il y a cependant des catégories d'éléments qui demandent une analyse plus réduite; par exemple les noms de sociétés, les noms de personnes, les lieux, ...

Le domaine d'expérimentation est constitué d'articles du domaine financier. Les auteurs ont décidé de rechercher les éléments *nom de société* et *nom de personne*. Ils se basent, pour cela, sur la forme que prennent les noms.

Dans un article, la première fois qu'un nom de société est cité, il apparaît sous sa forme complète et possède donc des mots comme *Corportation, Inc., Pty. Ltd.*. Dans la suite de l'article, le nom apparaît peut-être sous une forme abrégée. Il est donc assez aisé de reconnaître les noms de sociétés. Tous les problèmes ne sont pas résolus pour autant car les auteurs rapportent un taux de précision et de rappel respectivement de 89 % et 79 %.

De la même manière, les noms de personnes commencent par des mots comme *Ms., President, Chairman*. Pour les noms de personnes qui ne comportent pas de telles indications, le système fait usage de listes de noms et prénoms pour isoler les noms de personnes dans le texte des articles. Ici aussi le système est imparfait, mais meilleur que le module de reconnaissance des noms de sociétés. La précision est de 92 % et le rappel de 93 %.

Après cette phase de repérage de termes spécifiques, les auteurs décrivent une technique qui permet de générer des liens directs entre éléments en identifiant

des associations. Une association existe entre deux termes s'ils sont suffisamment proches l'un de l'autre. La mesure de distance est effectuée en calculant le nombre de mots entre les termes. La force de l'association repose sur le nombre d'occurrences des termes dans une fenêtre donnée de mots. Une fenêtre de x mots est composée des $x/2$ mots avant l'élément en question et des $x/2$ mots après celui-ci. Les différentes associations sont ordonnées suivant une mesure statistique afin de sélectionner les associations les plus représentatives. Les auteurs ont utilisé ce principe d'association pour identifier les relations entre un nom de personne et un nom de société.

La dernière phase du processus consiste à générer des liens indirects entre les termes. Ceux-ci sont déterminés en utilisant les techniques traditionnelles de l'*information retrieval* comme la fréquence d'occurrences d'un mot. Pour chaque élément repéré dans la première phase, un pseudo-document est créé. Il est constitué des paragraphes dans lesquels apparaissent le terme en question. Finalement, en comparant les différents pseudo-documents entre eux, il est possible de déterminer les termes qui ont une similarité suffisante et qui sont donc reliés par un lien indirect.

Bien que les auteurs n'en parlent pas explicitement dans leurs articles, les résultats de ces travaux peuvent servir à la création automatique ou semi-automatique de systèmes hypertextes.

4.2.2. L'approche du langage naturel

Certains chercheurs tentent d'utiliser des méthodes d'analyse du langage naturel pour détecter des structures susceptibles de devenir des ancres hypertextes. Nous présenterons les travaux de [LEHNERT, 1992] et de [NORDH., 1994].

Dans ses travaux, Lehnert [LEHNERT, 1992] propose d'identifier les citations d'articles afin de faciliter la tâche des utilisateurs qui consulteraient une base de données d'articles scientifiques. Les citations ainsi repérées permettraient à

l'utilisateur de sélectionner l'une d'entre elles et de consulter le document référencé.

Lehnert propose d'utiliser des techniques existantes d'analyse du langage naturel afin de repérer ces citations, mais il ne s'agit pas d'une analyse approfondie du texte. L'analyse superficielle des phrases incluant des citations devrait être suffisante car ces phrases peuvent être comprises à un niveau vague mais suffisant pour l'identification de la citation. L'analyse des phrases est effectuée par l'analyseur de phrases CIRCUS, développé précédemment par l'auteur.

Les citations sont tout d'abord reconnues et analysées de façon syntaxique, puis elles font l'objet d'une analyse sémantique. Le but est d'identifier la nature du lien entre la phrase qui contient la citation et le texte référencé. L'auteur distingue les citations sur le fond (*background citations*), les citations de support (*supportative citations*), les citations contrastantes (*contrasting citations*) et les citations techniques (*technically motivated citations*).

Le repérage des citations se base sur la présence de certains termes ou expressions ou sur la forme de la phrase, laquelle est dépendante du type de citation. Par exemple, l'expression « *experimental evidence* » peut indiquer la présence d'une citation de fond. Autre exemple, la phrase « *They also provide a computational theory, similar in spirit to the frame theory of representation in artificial intelligence (Minsky, 1981)* » comporte une citation de support à l'auteur. La précision de l'extraction des citations dépend de l'apprentissage et des connaissances données au programme. Plus les expressions ou termes seront nombreux, plus les formes de phrases seront précises, plus le programme sera précis.

Lehnert souhaiterait que l'extraction des citations soit indépendante du domaine mais, toujours selon l'auteur, si nous désirons que cette extraction soit optimale, il est vraisemblable que des connaissances spécifiques au domaine soient nécessaires.

Norhausen *et al.* [NORDH., 1994] décrivent une méthode pour automatiser la création des liens hypertextes en déterminant des liens implicites dans les textes. Quand, dans un texte, le lecteur rencontre les mots « *hypertext link* », il désire pouvoir consulter une explication détaillée du sujet. Malheureusement, deux problèmes se posent : quels *noun phrases* (groupes de mots) seront les noeuds de départ et quels seront les noeuds de destination ?

Les auteurs constatent que les titres des livres, des articles, des chapitres ou sections sont souvent sous la forme de *noun phrases*. Leur proposition est de créer des liens entre les *noun phrases* du corps du texte et les titres des noeuds. Dans ce raisonnement, l'hypothèse est que le titre d'un noeud représente bien le contenu du noeud. La méthode pour créer des liens repose sur la correspondance entre *noun phrases*. Cette méthode travaille en deux phases. La première est d'identifier des noeuds de départ significatifs, chaque *noun phrase* du texte étant considérée comme un noeud de départ potentiel. La deuxième phase consiste à trouver les noeuds de destination adéquats pour les *noun phrases* identifiés. Un filtre se charge d'éliminer les noeuds de départ redondants ou similaires.

Le programme utilise un générateur de *noun phrases*. Il s'agit d'un module qui reçoit en entrée une phrase et fournit comme résultat une liste des *noun phrases* présents dans cette phrase. Il est fait usage d'une grammaire, d'un dictionnaire et d'un algorithme d'élimination de suffixes afin de repérer les *noun phrases*. Un exemple, la phrase « *Selling points for small-scale hypertext applications* » comporte trois *noun phrases* qui sont « *sell point* », « *small scale hypertext apply* » et « *sell point for small scale hypertext apply* ». La dernière reprend d'ailleurs les deux premières.

Un module du programme compare les *noun phrases* générées avec les titres des noeuds. Chaque *noun phrase* est représentée par un vecteur binaire qui indique si un terme est présent ou non dans la *noun phrase*. Un vecteur binaire est choisi car il s'agit, en général, de phrases courtes sans terme répété pour lesquelles une pondération plus sophistiquée est inutile. Le produit des vecteurs nous donne une mesure de la similarité des *noun phrases* concernées. Si cette

similarité est suffisamment élevée (au-dessus d'une limite à déterminer), un lien hypertexte est créé entre la *noun phrase* et le noeud examiné. Si il y a plusieurs noeuds de destination possibles pour une même *noun phrase*, celui pour lequel la mesure de similarité est la plus élevée est choisi comme noeud de destination.

Les auteurs rapportent dans leur article que cette méthode de génération de liens est assez performante mais qu'elle doit encore être améliorée en prenant, par exemple, en considération le contexte des *noun phrases* et des titres. Ils affirment que la construction automatique des liens hypertextes est la clé de voûte des applications hypertextes de grande envergure.

4.2.3. L'approche réseau de neurones

Par soucis de complétude, nous présentons brièvement les travaux reprenant une construction des liens par des réseaux neuronaux.

Dans les travaux de recherche de Niels K. Bauer [BAUER, 1995], celui-ci envisage une approche par les réseaux neuronaux. Les résultats préliminaires de cette recherche sont proposés dans l'article cité, mais celui-ci est très bref et manque de détails sur les aspects techniques. L'auteur propose un système découpé en trois composants qui sont un module d'indexation automatique, un module de *clustering* contextuel et un module de conversion hypertexte.

C'est dans le module de classification contextuelle que se trouvent les réseaux de neurones. En utilisant un algorithme d'apprentissage non-supervisé, un réseau neuronal apprend à identifier des classes sémantiques, c'est-à-dire des groupes de phrases qui partagent un nombre significatif de termes, et pour lesquels des liens sont identifiés.

Dans le système, chaque phrase est représentée par un vecteur de termes pondérés. L'apprentissage du réseau est effectué sur base des vecteurs des phrases. Pour l'apprentissage d'une phrase, les poids des noeuds auxquels est connectée la phrase sont modifiés dans la « direction » de la phrase, de même pour les noeuds connectés aux phrases précédant et suivant la phrase en cours

d'évaluation. Grâce à cela, les noeuds associés aux phrases contextuelles (suivante/précédante) sont « déplacés » dans la direction de la phrase en cours d'apprentissage. Il y a donc une modification contextuelle des noeuds.

Après cet apprentissage, chaque phrase est examinée pour identifier l'impact sur le réseau de neurones. La phrase est reliée provisoirement au noeud de poids le plus élevé. La valeur de sortie du noeud est utilisée pour mesurer la signification d'une phrase. Les phrases ayant une signification élevée servent comme ancres hypertextes. En général, les phrases ayant des vecteurs de termes similaires sont mises en correspondance avec le même noeud.

Résumons l'utilisation du réseau de neurones. En entrée du réseau, nous avons les vecteurs de termes des phrases du document. Une phrases est associée à des noeuds. Les poids de ces associations sont modifiés sur base du contexte des phrases liées. En sortie du réseau, les différents noeuds ont une valeur associée par phrase. Le noeud, qui a la plus grande valeur de sortie pour une phrase et pour autant que cette valeur soit supérieure à une limite à fixer, est lié à la phrase en question. L'algorithme utilisé est un algorithme à apprentissage non supervisé.

Le but ultime des travaux de Bauer est l'étude du *clustering* contextuel. Il suppose qu'en attachant un contexte aux phrases d'un document, celles-ci vont se regrouper dans un *cluster* si elles présentent des termes communs et si elles sont sémantiquement similaires. Les phrases ayant des termes en commun, mais non sémantiquement liées, ne seront pas regroupées.

L'auteur a développé un prototype pour l'indexation et la classification contextuelle. Il a également effectué des tests sur de petits jeux de données. Les phrases couvrant des sujets similaires ont bien été regroupées mais des études comparatives avec d'autres méthodes n'ont pas encore été entreprises.

Les travaux de Georgel *et al.* [GEORGEL, 1993] cherchent à explorer la profondeur d'un espace documentaire en utilisant l'analyse en composantes locales angulaires (ACLA). « *L'ACLA est une méthode statistique dérivée des*

techniques classiques d'analyse des données. [...] L'ensemble des individus est considéré comme un nuage de points plongé dans un espace géométrique où chaque dimension correspond à un caractère. [...] L'ACLA recherche des axes qui sont définis dans cet espace géométrique, et qui synthétisent les données. [...] Il est donc possible d'associer à chaque axe de l'ACLA un sous-ensemble de données. Un axe d'ACLA peut être identifié à une classe résultat d'une classification.» [GEORGEL, 1993, pp. 33-34]. Dans le cadre des systèmes hypertextes, les individus sont les documents et les caractères sont les termes d'indexation.

Un réseau de neurones formels est utilisé pour calculer les axes de l'ACLA lesquels identifient des sous-ensembles de documents à l'intérieur de cet espace.

Dans leurs conclusions, les auteurs affirment que « *les axes de l'ACLA sont une façon d'étudier la profondeur d'un espace documentaire. Ils identifient d'abord, des sous-ensembles documentaires cohérents à l'intérieur de cet espace. Ensuite, ils nous renseignent sur le contenu de chaque sous-ensemble par une double représentation : par les mots d'indexation les plus caractéristiques d'une part; par les documents les plus significatifs d'autre part. Enfin, [...] ils facilitent l'accès aux contenus par un classement qui tient compte du poids décroissant des mots et des documents pour caractériser chaque sous-ensemble.* » [GEORGEL, 1993, P. 41]

L'utilité de cette modélisation est de permettre la description des relations du type « générique/spécifique » dans des thésauri. Les auteurs soulignent également que les axes ainsi définis ne sont pas très utiles pour la navigation hypertextuelle par les liens du type *voir aussi* (lien sémantique entre documents).

4.2.4 L'approche de l'*information retrieval*

Nous avons présenté dans un précédent chapitre les concepts fondamentaux de l'*information retrieval*. Ces concepts ont été utilisés par certains auteurs pour construire automatiquement des liens hypertextes. Nous pouvons mentionner les travaux de [SALTON, 1994A], [SALTON, 1994B], [NORDH., 1994] et de [SANDERS., 1994] que nous présenterons dans cet ordre.

Chez les différents auteurs, la première étape est toujours l'analyse des documents. Elle a pour but de déterminer un représentant de document pour chacun des documents à organiser. Le représentant est un vecteur de mots clés pondérés. Les vecteurs comportent *t* lignes où *t* est le nombre de mots distincts dans la collection de documents. Par convention, dans le cas où un terme est absent d'un document, la pondération de ce terme dans le vecteur vaut 0. La deuxième étape consiste à mesurer la similarité entre documents.

Les formules de pondération des termes et de mesure de similarité entre documents varient d'un auteur à l'autre.

Salton *et al.* utilisent la mesure *tf * idf*, c'est-à-dire *term frequency * inverse document frequency*. Elle donne un poids élevé aux termes qui apparaissent fréquemment dans certains documents, mais rarement dans l'ensemble des documents. Ce sont ces termes qui permettent de distinguer le document parmi les autres documents de la collection. Cette mesure donne une valeur entre 0 et 1 normalisée, c'est-à-dire que cette mesure n'est pas sensible à la taille des documents (le dénominateur assure la normalisation).

La fonction est définie de la manière suivante :

$$w_{ik} = \frac{tf_{ik} \cdot \log\left(\frac{N}{n_k}\right)}{\sqrt{\sum_{j=1}^{t} \left(tf_{ij}\right)^2 \cdot \left(\log\left(\frac{N}{n_j}\right)\right)^2}}$$

où w_{ik} *est le poids du terme* T_k *dans le document* D_i

tf_{ik} *est la fréquence d'occurrences du terme* T_k *dans le document* D_i

N *est le nombre de documents dans la collection*

n_k *est le nombre de documents comprenant le terme* T_k

t *est le nombre de termes distincts dans la collection de documents*

Remarquons que $\sum_{k=1}^{t} w_{ik}^2 = 1$ pour un document *i* quelconque.

Pour mesurer la similarité entre deux documents, les auteurs proposent la mesure suivante. Elle fournit un résultat entre 0 et 1 qui est fonction du poids des termes communs aux vecteurs de mots-clés des représentants de document.

$$sim(d_1, d_2) = \sum_{k=1}^{t} w_{1k} w_{2k}$$

où w_{ik} *est le poids du terme* T_k *dans le document* D_i

Remarquons que la similarité entre un document et lui-même est égale à un ($sim(d1,d1) = \sum_{k=1}^{t} w_{ik}^2 = 1$). Ce qui garantit que $sim(d1,d2) < sim(d1,d1) = 1$ car on suppose qu'au moins un terme est différent entre les documents d1 et d2.

Quant à Nordhausen *et al.*, ils utilisent une forme non normalisée de la formule proposée par Salton *et al.* pour évaluer le poids d'un terme dans un représentant de document.

$$w_{ik} = tf_{ik} . \log\left(\frac{N}{n_k}\right)$$

où w_{ik} *est le poids du terme* T_k *dans le document* D_i

tf_{ik} *est la fréquence d'occurrences du terme* T_k *dans le document* D_i

N *est le nombre de documents dans la collection*

n_k *est le nombre de documents comprenant le terme* T_k

Ils utilisent la mesure du cosinus de l'angle formé par les deux représentants de document dans un espace à t dimensions (les t dimensions correspondent aux t mots distincts dans la collection de documents). Cette mesure fournit également une valeur entre 0 et 1 selon la similarité des documents.

$$sim(A,B) = \frac{\sum_{k=1}^{t} w_{ak} w_{bk}}{\sqrt{\sum_{k=1}^{t} (w_{ak})^2 \sum_{i=1}^{t} (w_{bk})^2}}$$

où w_{ik} *est le poids du terme* T_k *dans le document* D_i

t *est le nombre de mots distincts dans la collection de documents*

Dans le système *News Retrieval Tool,* Sanderson *et al.* font plutôt appel à une mesure simple pour évaluer le poids d'un terme dans la collection de documents.

$$w_k = \log \frac{N}{n_k}$$

où w_k *est le poids du terme* T_k *dans la collection de documents*

N *est le nombre de documents dans la collection*

n_k *est nombre de documents comprenant le terme* T_k

Ils proposent la mesure suivante pour mesurer la similarité entre une requête et un document.

$$score = \sum_{i=1}^{t} d_i . w_i$$

où t *est le nombre de mots dans la requête*

w_i *est le poids du terme i dans le document examiné*

d_i *indique la présence (1) ou l'absence (0) du terme i dans un document*

En conclusion, nous pouvons dire que les mesures utilisées par Salton *et al.* et Nordhausen *et al.* sont normalisées entre 0 et 1. Elles sont utilisées à bon escient car le but avoué des auteurs est de réaliser un système hypertexte. Pour

Sanderson *et al.*, la mesure, dans le contexte de l'outil *News Retrieval Tool*, est adéquate car il s'agit de produire un classement, pas un système hypertexte.

L'important est donc de choisir une mesure qui soit respectueuse des « lois » de l'*information retrieval* et cohérente (normalisée) au sein de collections de documents de taille variable.

Dans [SALTON, 1994A], Salton *et al.* décrivent des méthodes pour construire des systèmes hypertextes à partir des résultats traditionnels d'*information retrieval*. Selon les auteurs, un réseau hypertexte est proposé pour la représentation de structures de textes liés. Les noeuds sont les textes ou morceaux de textes et les liens entre noeuds identifient des relations entre les textes. Les relations hiérarchiques, comme la structure hiérarchique d'un document (chapitres, sections, sous-sections), sont facilement et automatiquement détectées. Mise à part cette construction automatique sur une base lexicale, Salton *et al.* proposent d'analyser les textes et de suggérer des liens sémantiques pour la construction d'un réseau hypertexte.

Les résultats de cette construction automatique peuvent prendre des formes variées. Les auteurs identifient trois formes : la toile d'araignée (Figure 2), le plan d'articles liés (Figure 3) et la classification (Figure 4) [SALTON, 1994A, PP. 103-104].

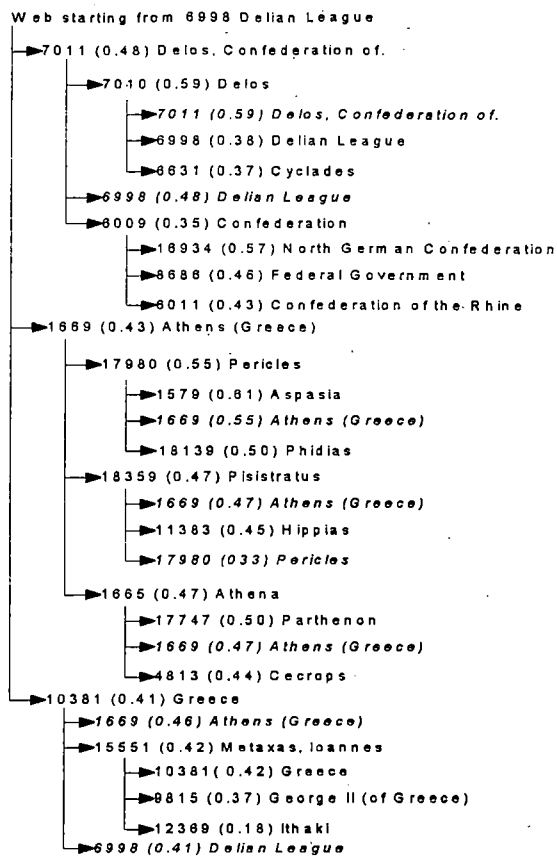


Figure 2 : Toile d'araignée (profondeur 3, largeur 3) depuis l'article "Delian League"

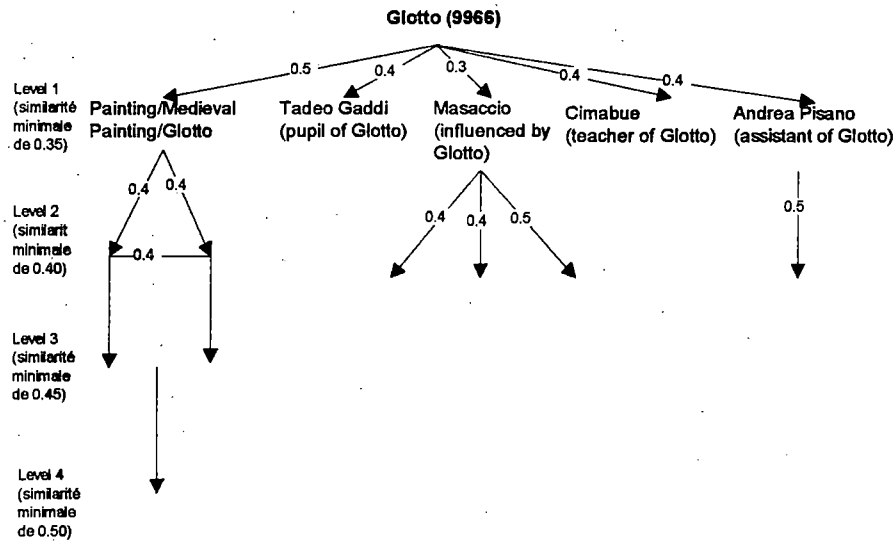


Figure 3 : plan des articles liés

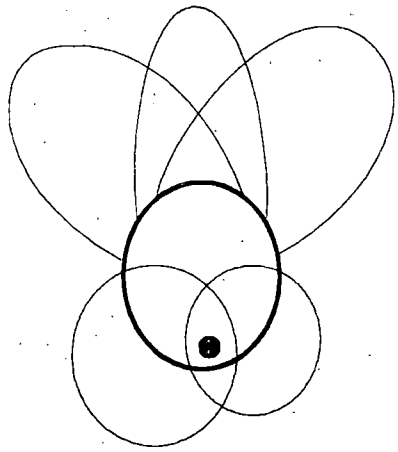


Figure 4 : *clusters* autour d'un article (chaque ensemble reprend des éléments de même similarité avec l'article identifié)

Dans le cas de la toile d'araignée (Figure 2), un document sert de référence. Une recherche est réalisée pour identifier les documents présentant une similarité suffisamment élevée et les *n* premiers sont repris. Ensuite, pour chacun de ces documents, ceux présentant une similarité suffisante sont retenus et les *n* premiers sont sélectionnés. Ce processus se répète sur *m* niveaux identifiés. Il s'agit d'une recherche en profondeur et en largeur. Ce procédé permet de construire une hiérarchie de documents, lesquels deviennent de plus en plus éloignés du document initiateur.

Dans le plan d'articles liés (Figure 3), un document de référence est déterminé. Des similarités minimales sont assignées aux différents niveaux. Au niveau un se trouvent les articles liés au document initiateur et présentant une similarité minimale de 0,30. Au niveau deux, les articles similaires aux articles du niveau 1 et qui présentent une similarité minimale de 0,35 sont repris. Et ainsi de suite pour les autres niveaux en augmentant la similarité minimale requise. Finalement, après un certain niveau, le processus va s'arrêter de lui-même car plus aucun article ne présente de similarité suffisamment élevée pour être retenu.

Dans le processus de *clustering* (Figure 4), les documents qui présentent une similarité entre eux sont regroupés. Un document de départ est sélectionné et les documents présentant une similarité équivalente sont regroupés dans les ensembles. Le cercle en gras dans la Figure 4 reprend des documents de similarité 0,39 par rapport à l'article initiateur. Ces agrégats ont été introduits dans les systèmes hypertextes afin de réduire la complexité du système hypertexte. Les auteurs font remarquer que le processus de *clustering* est assez lourd et inapplicable pour de grandes collections de documents. Dans leur implémentation, ils proposent d'utiliser la classification à un niveau local, c'est-à-dire en classifiant les articles résultant d'une requête.

Selon notre point de vue, une méthode de *clustering* globale est intéressante. Elle permet d'organiser une collection de documents suivant des thèmes particuliers. Mais plusieurs obstacles peuvent s'opposer à cette méthode : notamment un temps de calcul non négligeable, une stabilité de classification sujette à caution en cas de modification du contenu du système et finalement, une difficulté de présentation à l'utilisateur des agrégats ainsi déterminés.

Nordhausen *et al.* utilisent des méthodes moins sophistiquées pour générer des liens entre documents. Leur méthode consiste à calculer la similarité des documents pris deux à deux. Cette méthode est, rappelons-le, fort coûteuse en temps de calcul. L'algorithme a un temps d'exécution de l'ordre de n^2 où n est le nombre de documents. Une limite est définie pour la similarité et toutes paires de documents présentant une similarité supérieure à cette limite fait l'objet d'un lien hypertexte.

Sanderson *et al.* ont développé un système qui permet de trouver des articles de presse similaires à une requête ou à un ensemble d'articles. Les auteurs ne parlent pas explicitement de systèmes hypertextes. Ce système fonctionne de la manière suivante : un utilisateur introduit une requête et consulte une liste de documents jugés pertinents par le logiciel. Ensuite, par un processus de *relevance feedback*, une nouvelle requête est créée et analysée par le système qui retourne une nouvelle liste de documents pertinents.

Quel est l'intérêt de cette méthode pour la création de systèmes hypertextes ? Il ne s'agit pas d'un processus automatique de construction de réseaux hypertextes, mais d'un processus interactif qui détermine une liste de documents similaires à la requête ou à d'autres documents. C'est là que nous voyons l'intérêt de cette méthode. Il s'agit d'un outil d'aide à la conception de réseaux hypertextes pour un auteur de systèmes hypertextes. Au lieu de se reposer uniquement sur sa connaissance du domaine (qui est vraisemblablement limitée), l'auteur peut utiliser un système interactif qui lui suggère une série de liens vers des documents pertinents.

Chapitre 4 :
L'application des techniques d'*information retrieval* à la création d'un système hypertexte

1. Introduction

Dans ce chapitre, nous allons proposer d'utiliser des techniques d'*information retrieval* pour la construction automatique d'un système hypertexte. Le but ultime d'un programme de ce type est la génération d'un système hypertexte pour un ensemble de documents quelconques. Nous prendrons comme exemple les messages échangés sur les *newsgroups* d'Internet comme documents pour les raisons suivantes :

- les messages des *newsgroups* sont des documents textuels (à part les groupes spécialisés *alt.binaries...*);

- la toute grande majorité des messages sont écrits en anglais. Cela s'avère déterminant car nous disposons d'un algorithme d'élimination des suffixes pour la langue anglaise uniquement;

- les messages de *newsgroups* possèdent aussi une série d'informations qui permettent d'organiser la collection de documents selon l'auteur, le sujet ou encore la date.

Pour ces différentes raisons, notre programme traitera les messages des *newsgroups* et générera une série de pages *World Wide Web* (WWW) au format *HyperText Mark-Up Langage* (HTML). Elles pourront être mises à la disposition des utilisateurs sur un serveur WWW. Nous avons utilisé ce format pour une raison pragmatique qui est la simplicité de la génération de pages WWW. Le format HTML est un format ASCII avec des marquages pour les ancres hypertextes, pour le style des caractères, ... L'utilisation d'un autre format pour la génération des pages du système hypertexte (par exemple l'aide de *Windows*) se révèle beaucoup trop compliquée pour les besoins expérimentaux.

Nous allons donc, dans un premier temps, présenter les données et les traitements que nous allons appliquer. Nous évoquerons ensuite l'implémentation.

2. Les données

Les données sont les messages de *newsgroups*. Nous avons choisi d'organiser les messages des groupes de discussion suivants : *alt.hypertext, comp.os.linux.announce, comp.security.unix, comp.security.misc* et *comp.text*. N'importe quel groupe de discussion peut être utilisé.

Le format des messages Usenet se trouve dans la RFC 850 [RFC850]. Un message Usenet se présente de la manière suivante (Figure 1)

```
From: Lonestar <i219@stio1.fh-wuerzburg.de>
Newsgroups: alt.hypertext
Subject: Re: Custom Backgrounds and Tiling in, How?
Date: 3 Jul 1995 14:17:12 GMT
Organization: Dep. of Computer Science at University Wuerzburg, Germany
Lines: 21
Message-ID: <3t8u58$srr@winx03.informatik.uni-wuerzburg.de>
References: <3t4p8p$i2j@news.pacificrim.net>
NNTP-Posting-Host: rzpc14.sari.fh-wuerzburg.de
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
X-Mailer: Mozilla 1.2b1 (Windows; I; 16bit)
To: jsedg@pacificrim.net

John Sedgwick <jsedg@pacificrim.net> wrote:
>How does one take a custom background graphic created in a
>paint or illustration program and set it up so it tiles in as
>an HTML background on a Web document.

Nothing easier than that !
Define your WHOLE page as BODY (apart from the title)
as:

<body background=(your image)>

Thats it !


Lonestar http://stio1.sari.fh-wuerzburg.de/student/i219
                                                    //
  Amiga1230CDT * 3DO * MegaCD * SuperFamicom      \\//
                                                   X/
-- Department of Computer Science, FH Wuerzburg --
```

Figure 1 : exemple d'un message Usenet

Un message est composé d'un groupe de lignes appelé l'en-tête. L'en-tête est séparé du corps du message par une ligne blanche et reprend des informations utiles à l'échange des messages entre les systèmes informatiques et des informations sur le message lui-même.

Dans notre programme, nous ne ferons usage que des lignes d'en-tête suivantes :

- **From:** adresse e-mail de l'auteur du message;
- **Subject:** sujet du message, attribué par l'auteur;
- **Date:** date de création du message, attribuée par le système;
- **Message-ID:** identifiant du message dans le système Usenet;
- **References:** identifiant du ou des articles pour lesquels ce message constitue une réponse.

Les autres lignes d'en-tête ne nous servirons pas dans le cadre de ce programme.

3. Les traitements

Les traitements appliqués aux messages des groupes de discussion sont les suivants. Les traitements sont présentés dans l'ordre conceptuel et seront détaillés par après.

- lecture du fichier constitué des messages d'un *newsgroup* et séparation des différents messages;
- constitution d'une base de données des messages;
- calcul du poids des mots dans les messages;
- classification des messages;
- génération des pages WWW et des index.

3.1. Séparation des messages

Les messages d'un *newsgroup* sont regroupés dans un seul fichier. Ils sont séparés par une ligne de 8 signes « = ». La séparation consiste à créer pour chaque message un fichier qui reprend le sujet et le corps du message. Ce fichier servira au calcul du poids des mots dans les messages. Ainsi les informations superflues dans l'en-tête ne seront pas prises en compte lors du calcul des pondérations.

3.2. Constitution d'une base de données des messages

Pour les traitements ultérieurs, une base de données est créée. Son modèle de donnée est le suivant (Figure 2).

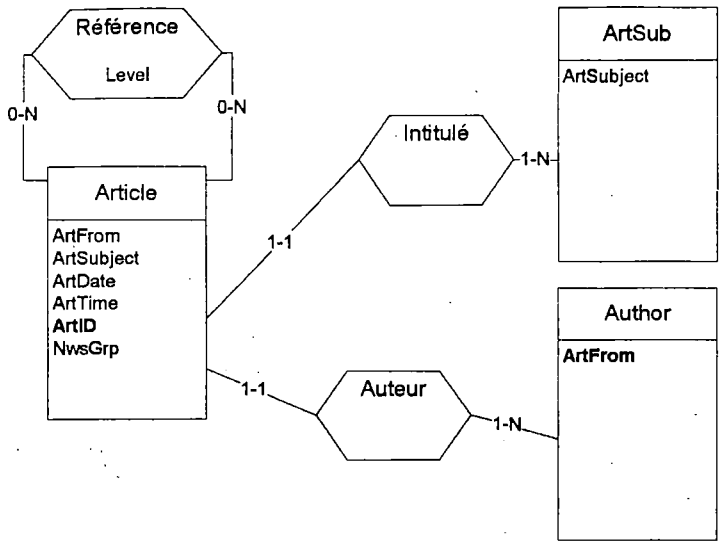


Figure 2 : modèle de données pour les messages Usenet

Un article (entité Article) reprend les informations pour un message Usenet. Un message Usenet, identifié par **ArtID**, référence zéro, un ou plusieurs autres articles et un message peut être référencé par zéro, un ou plusieurs autres messages (association Référence). Les articles référencés sont ordonnés par valeur de Level.

Un article possède un sujet (entité ArtSub). Plusieurs articles peuvent concerner un même sujet.

Un article est écrit par un auteur (entité Author). Un auteur a écrit un ou plusieurs articles.

Les tables relationnelles correspondantes sont reprises dans la Figure 3.

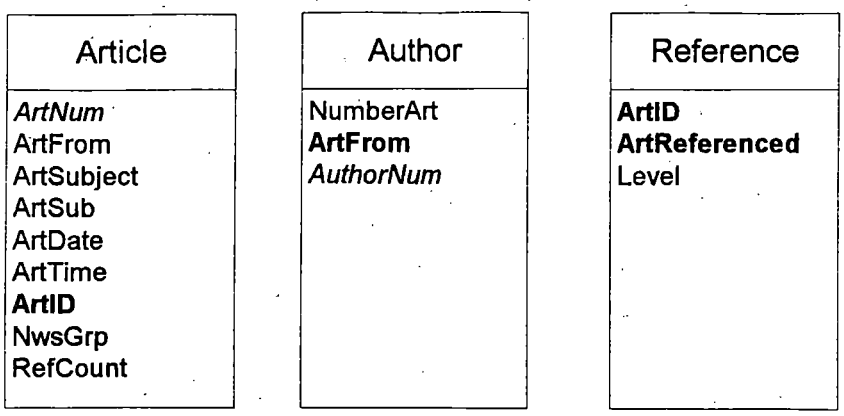


Figure 3 : tables de la base de données pour les messages Usenet

Un certain nombre de redondances apparaissent dans les tables. Bien qu'un article soit identifié par l'attribut **ArtID**, nous avons ajouté un identifiant *ArtNum* (utilisé pour générer un nom de fichier unique par message). De même, la table Author possède un identifiant naturel **ArtFrom**. Mais, pour pouvoir générer un nom de fichier unique, nous avons ajouté un identifiant *AuthorNum*.

L'entité ArtSub est fusionnée avec l'entité Article. A propos du champ ArtSub, le sujet d'un article dans ArtSub est peut-être différent de celui repris dans ArtSubject. Dans le cas où un message constitue une réponse à un autre message, la convention est d'ajouter « Re: » au début du sujet du message réponse. Ce champs reprend le sujet des messages sans ce préfixe afin de pouvoir regrouper plus facilement les articles et leurs réponses.

Des index sont établis afin d'accélérer les recherches. Pour des raisons de performance, les index sont créés après l'introduction des données dans les tables de telle sorte que le système de gestion de base de données ne doive pas mettre à jour continuellement les index.

Toujours pour des raisons de performance, les tables ne sont pas munies de contraintes structurelles (*not null, primary key,* ...) qui ralentissent considérablement les ajouts dans la base de données.

3.3. Calcul des poids des mots dans les messages

Afin d'établir une classification des messages, il est nécessaire de procéder à une évaluation du poids des mots dans les messages. La théorie a été expliquée dans le chapitre 2. Le déroulement opérationnel se fait de la manière suivante.

Une première passe à travers la collection des messages permet de repérer tous les mots, d'éliminer ceux qui apparaissent dans une liste de mots fréquents (*stopwords*). Après avoir parcouru l'ensemble de la collection de documents une première fois, nous avons décidé d'éliminer les termes qui apparaissent une seule fois dans la collection de documents ou ceux qui apparaissent dans un seul document. Ces termes ne nous permettent pas de comparer des documents et sont donc inutiles.

Une deuxième passe est effectuée dans la collection de messages. Les mots intéressants sont conservés dans les vecteurs de mots clés des messages. Ensuite, un calcul est effectué pour déterminer le poids des mots en fonction du nombre d'occurrences dans le document et dans la collection de documents. La pondération utilisée est

$$w_{ik} = \frac{tf_{ik} \cdot \log\left(\frac{N}{n_k}\right)}{\sqrt{\sum_{j=1}^{t} \left(tf_{ij}\right)^2 \cdot \left(\log\left(\frac{N}{n_j}\right)\right)^2}}$$

où w_{ik} *est le poids du terme* T_k *dans le document* D_i

tf_{ik} *est la fréquence d'occurrences du terme* T_k *dans le document* D_i

N *est le nombre de documents dans la collection*

n_k *est le nombre de documents comprenant le terme* T_k

t *est le nombre de termes distincts dans la collection de documents*

Pour rappel, cette formule fournit un poids normalisé entre 0 et 1; il est indépendant de la taille du document. Le résultat de ce traitement est conservé dans une base de données dont voici le schéma conceptuel (Figure 4).

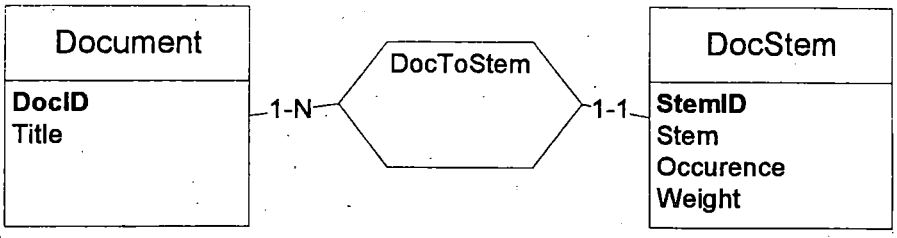


Figure 4 : modèle de données pour le poids des mots dans les documents

Un document (entité Document) comporte un titre et est identifié par **DocID**. Il est composé de 1 ou plusieurs *stems* (mots réduits à une racine). Chaque racine (entité DocStem) est identifiée par **StemID et DocID**. Une racine est caractérisée par son nombre d'occurrences dans le document et son poids.

Les tables relationnelles correspondantes sont reprises ci-dessous (Figure 5).

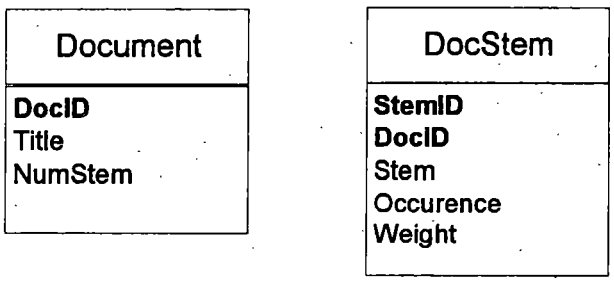


Figure 5 : tables de la base de données pour le poid des mots dans les documents

3.4. La classification des messages

Les messages vont faire l'objet d'une procédure de *clustering* afin de regrouper les documents similaires. La classification se base sur une mesure de similarité entre représentants de document. La fonction utilisée est :

$$sim(d_1, d_2) = \sum_{k=1}^{t} w_{1k} w_{2k}$$

Cette fonction retourne une valeur entre 0 et 1 selon le degré de similarité. Plus la similarité d'une paire de documents se rapproche de 1, plus les documents sont similaires.

La classification qui sera établie n'est pas idéale puisque l'algorithme choisi ne garantit pas un *clustering* stable s'il est exécuté une nouvelle fois sur le même jeu de données. L'algorithme de classification est le suivant. Il travaille en une passe après une phase d'initialisation.

Celle-ci consiste à déterminer une série de documents qui serviront de *clusters* initiaux. Le nombre de clusters initiaux est déterminé en fonction du nombre de documents à classer. Par expérimentation, nous avons déterminé que prendre 3% des documents comme *clusters* initiaux est approprié. Ces documents seront choisis au hasard dans la collection de documents mais aussi dissemblables que possible les uns des autres. Cette condition sera mesuré par une similarité inférieure à un seuil à fixer. Dans notre prototype, nous avons choisi de prendre une similarité inférieure à 0,05.

Les autres documents sont pris un à un et comparés avec l'ensemble des classes. Si le document en question présente une similarité supérieure à un seuil donné avec une ou plusieurs classes, le document est ajouté à la classe avec laquelle il présente la similarité maximale. Si aucune classe ne répond au critère de similarité, le document est utilisé comme représentant d'une nouvelle classe. Nous avons déterminé expérimentalement qu'un seuil de similarité acceptable était de 0,175.

Lors de l'ajout du document à la classe déterminée, il est procédé à la fusion du document avec la classe, c'est-à-dire qu'un nouveau représentant de classe est calculé comme étant la moyenne arithmétique des poids des pondérations de mots clés des documents déjà présents dans la classe et du représentant du nouveau document. Cela est fait uniquement pour les termes communs au représentant de la classe et au représentant du document.

Précédemment, nous avions envisagé de fusionner totalement les représentants de document, c'est-à-dire de prendre tous les termes de tous les documents comme représentants du *cluster*. Cette solution s'est avérée impraticable pour la raison suivante. La taille des représentants de *clusters* croît fortement puis se stabilise. Le nombre élevé de termes fait que la similarité avec de nouveaux documents est de plus en plus élevée et donc que les documents deviennent de plus en plus similaires aux différentes classes. Il devient alors difficile de justifier l'attribution d'un document à une classe si la différence de similarité est faible par rapport à l'affectation du document à une autre classe.

Tout comme le représentant de la classe, l'inertie de celle-ci est recalculée. L'inertie de la classe se définit normalement comme la somme des similarités des documents de la classe pris deux à deux. C'est un indicateur de la « cohérence » des documents de la classe. Dans notre implémentation, à nouveau pour des raisons de performance, il est impossible d'utiliser cette mesure telle quelle. Nous avons donc décidé d'approcher cette mesure par la similarité des articles avec le représentant de classe. Par convention, une classe d'un élément a une inertie de 0,4. Ce seuil a été déterminé expérimentalement afin de garantir l'homogénéité des classes obtenues. L'ajout d'un document à la classe modifie l'inertie et augmente celle-ci de la similarité du document avec le représentant de document recalculé.

Si, après la fusion du document et de la classe, la classe comporte plus de 5 documents et que son inertie par document (*inertie de la classe/nombre de documents*) est inférieure à un seuil donné, celle-ci est divisée en deux. Le seuil que nous avons déterminé expérimentalement est 0,4. Dans une première version du

prototype, nous utilisions seulement un nombre maximum de documents. Cela a pour désavantage que certains groupements naturels de documents soient arbitrairement coupés alors que les documents sont fortement cohérents. La solution actuelle permet des classes de taille supérieure au seuil de 5 documents pour autant que la cohérence des documents dans la classe soit assurée.

La séparation de la classe s'effectue de la manière suivante. Les documents de la classe sont comparés deux à deux. Les deux documents pour lesquels la similarité est la plus faible (donc la dissimilarité la plus forte) servent de représentants initiaux de deux nouvelles classes. Ensuite, les autres documents de l'ancienne classe sont assignés à une des nouvelles classes pour laquelle leur similarité est la plus forte.

Le résultat de cet algorithme est un ensemble de classes comprenant un certain nombre de messages. Ce nombre n'est pas borné. Ce résultat est conservé dans la base de données dans la table Clusters (Figure 6).

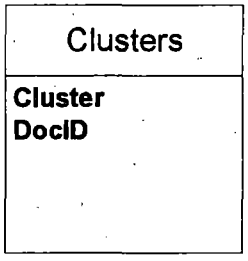


Figure 6 : table qui associe un document à un *cluster*

L'identifiant de cette table est **Cluster et DocID**. Elle répertorie l'appartenance d'un document à un *cluster*.

3.5. La génération des pages WWW et des index

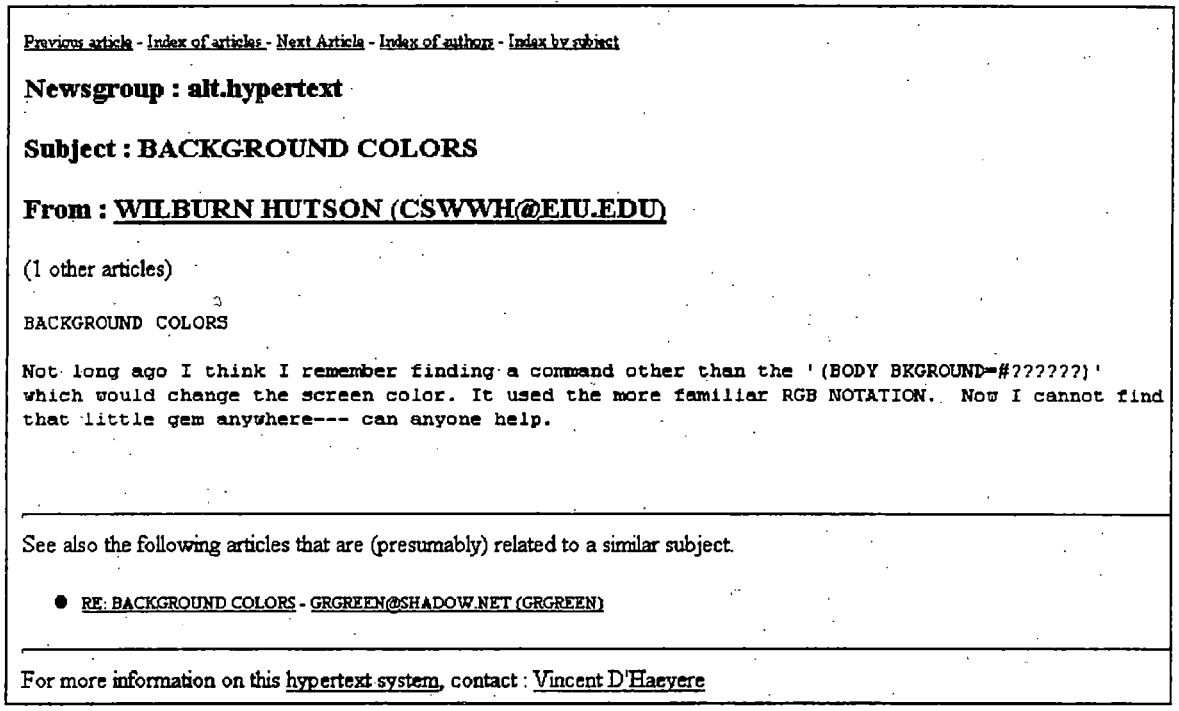
Pour chaque message, une page WWW est créée. Cette page reprend les éléments suivants : le nom du *newsgroup* dont le message est issu, le sujet du

message, le nom de l'auteur du message, une liste de messages dont il constitue une suite, le corps du message et une liste de messages dans le même cluster.

Des ancres hypertextes sont définies pour accéder à :

- l'article précédant et suivant du *newsgroup* (selon l'ordre chronologique);
- l'index des auteurs qui ont écrit dans le groupe de discussion;
- l'index des messages triés par ordre chronologique;
- l'index des sujets traités dans le groupe de discussion.

Un message se présentera de la manière suivante (Figure 7). N'importe quel outil de navigation WWW permet de visualiser ces pages.


Previous article - Index of articles - Next Article - Index of author - Index by subject

Newsgroup : alt.hypertext

Subject : BACKGROUND COLORS

From : WILBURN HUTSON (CSWWH@EIU.EDU)

(1 other articles)

BACKGROUND COLORS

Not long ago I think I remember finding a command other than the '(BODY BKGROUND=#??????)' which would change the screen color. It used the more familiar RGB NOTATION. Now I cannot find that little gem anywhere--- can anyone help.

See also the following articles that are (presumably) related to a similar subject.

- RE: BACKGROUND COLORS - GRGREEN@SHADOW.NET (GRGREEN)

For more information on this hypertext system, contact : Vincent D'Haeyere

Figure 7 : apparence d'un article

4. L'implémentation

4.1. Les outils de programmation

L'implémentation a été effectuée en *Borland C++ 4.0 sous Windows* de Borland International. Le programme ne dispose pas d'une interface graphique, mais d'un système de menu textuel, le caractère expérimental de ce logiciel n'exigeant pas une interface compliquée, d'autant plus qu'il s'agit d'un système entièrement automatique et qu'à priori, aucune interface n'est nécessaire. Le système est paramétré par un fichier de configuration qui reprend la liste des fichiers de messages à traiter.

La base de données qui a été utilisée est *Ocelot2 - The SQL*, de Ocelot Computer Services Incorporated. Il s'agit d'une base de données relationnelle. Cette base de données est conçue pour être utilisée à partir du langage de programmation (C, C++, Basic, Cobol, Pascal) en incluant dans le code source des commandes SQL (du style *exec sql select * from document;*). Un précompilateur se charge de transformer ces commandes SQL en commandes compilables par le compilateur choisi.

Pourquoi avoir choisi cette approche pour la base de données ? Parce que nous avions à notre disposition la base de données, la documentation et surtout que nous n'avions pas le temps de nous lancer dans la programmation complexe d'un système de fichiers indexés plus adapté à l'utilisation en *information retrieval.*

Ce choix a été celui du bon sens, mais malheureusement, cette base de données a de sérieuses limitations. Pour notre programme, la principale limitation a été l'impossibilité d'utiliser simultanément deux curseurs pour parcourir la base de données. Pour contourner cette difficulté, nous avons dû introduire de fortes redondances dans la base de données. Pour la même raison, lors de la génération des pages WWW, le programme s'exécute en deux passes. La première passe

construit les pages WWW pour les messages et les index. La seconde passe exploite les résultats de l'algorithme de *clustering* et intègre ceux-ci dans les pages des messages.

La conception du programme est suffisamment modulaire pour permettre de changer de base de données si cela s'avère nécessaire.

4.2. L'architecture du programme

L'architecture du programme est basée partiellement sur une philosophie orientée objet et sur une programmation procédurale plus traditionnelle. Les aspects orientés objets se retrouvent dans la gestion des documents et du calcul des pondérations pour les mots. L'approche procédurale a été utilisée dans la séparation initiale des messages, l'élimination des suffixes, la classification des messages, la génération des pages WWW, la manipulation de la base de données et une série de fonctions utilitaires.

Nous proposerons également une découpe en niveau présentation - traitement - base de données pour les différentes fonctions.

Le code du programme se trouve en Annexe A.

4.2.1. L'approche orientée objet

Dans la gestion des documents et le calcul du poids des mots, nous avons favorisé une architecture orientée objet. Nous avons défini les classes suivantes. Les noms des fichiers du programme apparaissent en italique.

- les chaînes de caractères (*mystring.h* et *mystring.cpp*);
- les racines (*stemc.h* et *stemc.cpp*);
- la liste chaînée (*dlink.h*);
- l'arbre binaire trié (*tree.h*);
- les documents (*document.h* et *document.cpp*);

- le gestionnaire de documents (*document.h* et *document.cpp*).

Voici les classes telles que définies (Figure 8).

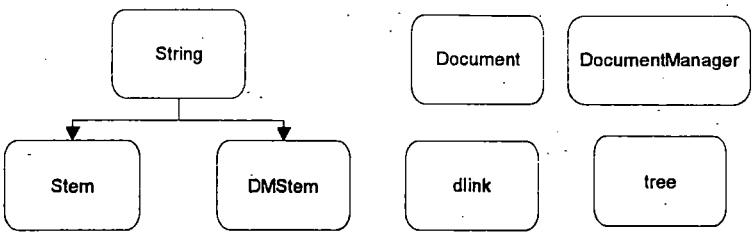


Figure 8 : classes pour la gestion des documents et le calcul du poids des mots

Détaillons ces classes.

a) La classes String

Cette classe est chargée de gérer une chaîne de caractères. Cette définition de classe est nécessaire car la gestion de listes doublement chaînées et d'arbres binaires triés manipulent des objets (voir infra.).

Les méthodes définies sont les suivantes :

Mystring() : constructeur d'une chaîne de caractères vide;

Mystring(char *) : constructeur d'une chaîne de caractères copiant la chaîne de caractères passée en paramètre;

~Mystring() : destructeur de la chaîne de caractère si elle n'est pas vide;

Les opérateurs suivants sont redéfinis pour la classe Mystring : inférieur (<), supérieur (>) et égalité (==).

L'opérateur << est défini pour l'affichage d'une instance de MyString sur un *stream* C++.

b) La classe Stem

La classe Stem dérive directement de la classe MyString. Elle est chargée de gérer les informations pour une racine, telles que le nombre d'occurrences ou la pondération de la racine.

Les méthodes suivantes sont définies :

Stem() : constructeur pour une racine vide. Le nombre d'occurrences est de 1 par convention;

Stem(char *) : constructeur d'une racine copiant la chaîne de caractères donnée en paramètre. Le nombre d'occurrences est mis à 1 et le poids à 0.

Le destructeur n'a aucune fonctionnalité supplémentaire par rapport au destructeur de la classe MyString.

L'opérateur << est défini pour l'affichage d'une instance de Stem sur un *stream* C++.

c) La classe DMStem

La classe DMStem gère également des racines, mais pour le gestionnaire de documents. Elle dérive directement de la classe MyString. Elle conserve le nombre de documents dans la collection possédant ce terme ainsi que le nombre d'occurrences de ce terme dans la collection. Elle conserve également le numéro du dernier document dans lequel elle a été localisée.

Les méthodes suivantes sont définies :

Dmstem() : ce constructeur crée un chaîne de caractères vide, initialise, par convention, le nombre d'occurrences à 1, met à -1 le numéro du dernier document dans lequel elle a été localisée et met à 0 le nombre de documents dans lesquels cette racine est présente;

Dmstem(char *) : réalise la même chose que le constructeur ci-dessus mais copie la chaîne de caractères donnée en paramètre.

Le destructeur n'a pas de fonctionnalité supplémentaire par rapport à celui de la classe String.

L'opérateur << est défini pour l'affichage d'une instance DMStem sur un *stream* C++.

d) La classe Dlink

La classe Dlink gère une liste doublement chaînée d'objets quelconques. Elle est définie sous la forme d'un *template* C++, ce qui garantit sa généricité. Sa structure interne peut être représentée comme ceci (Figure 9). Elle possède de plus la notion d'objet courant. L'objet courant peut être consulté, modifié ou effacé. La liste peut être parcourue dans les deux sens.

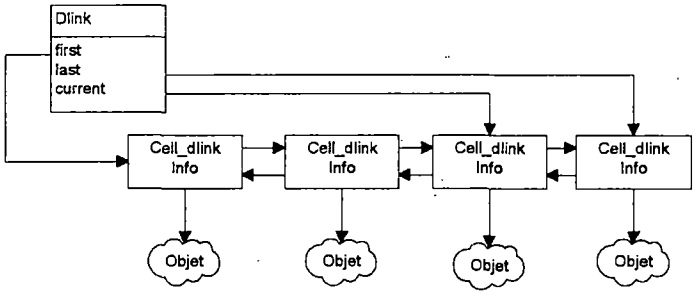


Figure 9 : représentation d'une liste doublement chaînée

Les fonctions suivantes sont définies :

Dlink() : construit une liste doublement chaînée vide;

~Dlink() : détruit la liste des objets mais pas les objets eux-mêmes.

AddAtBegin() : ajout d'un objet en tête de liste. L'objet courant n'est pas modifié;

AddAtEnd() : ajout d'un objet en queue de liste. L'objet courant n'est pas modifié;

InsertBeforeCurrent() : insère un objet avant l'objet courant. Celui-ci n'est pas modifié;

InsertAfterCurrent() : insère un objet après l'objet courant. Celui-ci n'est pas modifié;

InsertAtPos() : insère un objet à une place bien précise dans la liste d'objets. L'objet courant n'est pas modifié;

MoveToNext() : l'objet courant devient l'objet suivant l'ancien objet courant;

MoveToPrevious() : l'objet courant devient l'objet précédant l'ancien objet courant;

MoveTo() : l'objet à la position spécifiée devient l'objet courant;

MoveAtBegin() : l'objet courant est le premier de la liste;

MoveAtEnd() : l'objet courant devient le dernier de la liste;

GetCurrentInfo() : un pointeur vers l'objet courant est retourné comme résultat;

SetCurrentInfo() : l'objet courant est remplacé par l'objet donné en paramètre;

DeleteCurrentInfo() : l'objet courant est détruit;

DeleteAllInfo() : tous les objets de la liste sont effacés;

DeleteAtBegin() : supprime le premier objet de la liste;

DeleteAtEnd() : supprime le dernier objet de la liste;

DeleteCurrent() : supprime l'objet courant de la liste;

e) La classe Tree

La classe Tree est chargée de gérer des arbres binaires triés d'objets. Elle est définie sous la forme d'un *template* C++, d'où sa généricité. Un arbre binaire est trié si tous les objets du sous-arbre gauche sont inférieurs à l'objet de la racine et si tous les objets du sous-arbre droit sont supérieurs à l'objet de la racine. Les objets qui sont insérés dans un tel arbre doivent définir ou hériter des opérateurs inférieur(<), supérieur(>) et égalité (= =). Cet arbre n'admet pas les doublons.

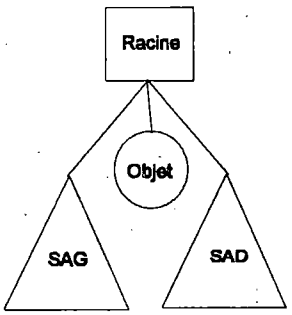


Figure 10 : un arbre binaire

Les méthodes suivantes sont définies :

Tree() : initialise un arbre vide;

~Tree() : détruit un arbre mais pas les objets;

Add() : ajout dans l'arbre de l'élément passé en paramètre;

Find() : recherche dans l'arbre l'élément passé en paramètre;

DeleteAllInfo() : détruit tous les objets de l'arbre;

e) La classe Document

Les documents sont représentés par des instances de la classe Document. Les attributs d'un document sont son identifiant docID, un fichier auquel il est associé, un titre, une liste doublement chaînée de racines ainsi qu'un arbre sur ces mêmes racines (pour des raisons d'efficacité). Les représentants des classes sont aussi des documents. Un document est éventuellement géré par un DocumentManager.

Les méthodes suivantes sont définies :

Document() : création d'un document vide, sans fichier associé, sans titre, avec une liste et un arbre vide de racines;

Document(char *, documentManager *, char *) : construction d'un document associé à un fichier donné, lié à un DocumentManager et portant le titre passé en paramètre. Une liste et un arbre vides de racines sont créés.

Store() : sauvegarde dans la base de données des informations concernant un document, y compris les racines;

Update() : mise à jour dans la base de données des informations concernant les racines d'un document;

Retrieve() : récupération des racines depuis la base de données;

DeleteLists() : détruit la liste et l'arbre de racines;

FirstPass() : parcourt le fichier associé mot par mot. Si le mot est présent dans la liste des mots fréquents du documentManager, alors il est ignoré. Sinon, le mot est réduit à sa racine. Si la racine répond à des impératifs de longueur minimale et maximale, elle est conservée dans la liste et l'arbre des racines du document. Les informations sur les racines dans le documentManager sont modifiées de manière appropriée;

SecondPass() : parcourt le fichier associé mot par mot. Si le mot est présent dans la liste des mots fréquents du documentManager, alors il est ignoré. Sinon, le mot est réduit à sa racine. Si la racine est présente dans la liste des racines du documentManager, elle est conservée dans la liste et l'arbre des racines du document;

CalculateWeight() : calcule le poids de toutes les racines du document en fonction des informations sur les racines dans le document et dans le documentManager;

d) La classe documentManager

Une instance de la classe documentManager est créée afin de faciliter le traitement de tous les documents. Cet objet conserve une liste des *stopwords*, une liste (et un arbre) des racines présentes dans la collection de documents et une liste des documents.

Les méthodes définies sont :

DocumentManager() : construit une liste vide de documents, une liste et un arbre vide de racines;

~DocumentManager() : détruit la liste des documents, la liste et l'arbre des racines ainsi que la liste des *stopwords*;

Add() : ajoute le document passé en paramètre aux documents que le documentManager prend en gestion.

DeleteDocuments() : détruit les documents dont il a la charge;

FirstPass() : active la méthode FirstPass sur tous les documents qu'il gère;

SecondPass() : active la méthode SecondPass sur tous les documents qu'il gère;

EliminateStems() : élimine les racines qui ne sont pas pertinentes dans une collection de documents (celles qui apparaissent dans un seul document, ou une seule fois dans la collection);

CalculateWeight() : parcourt la liste des documents et calcule le poids de toutes les racines de ces documents;

ReadStopWords() : construit l'arbre des mots fréquents à partir du fichier passé en paramètre;

4.2.2. L'approche procédurale

L'approche procédurale a été utilisée dans la séparation initiale des messages (*splitnws.h* et *splinws.cpp*), l'élimination des suffixes (*stem.h* et *stem.c*), la classification des messages (*clusters.h* et *clusters.cpp*), la génération des pages WWW (*splitnws.h* et *splitnws.cpp*), la manipulation de la base de données (*dbirnws.h, dbirnws.cpp* et *dbirnwsc.cpp*), une série de fonctions utilitaires et les fonctions de menu.

Explicitons ces traitements.

a) La séparation initiale des messages

But : Les messages constituant le fichier de messages sont mis dans un fichier propre. Les informations sur ces messages (date, auteur, sujet, ...) sont introduites dans la base de données.

Traitement : parcourt le fichier d'un newsgroup ligne par ligne et identifie les messages. Il conserve les informations sur l'auteur, la date, le sujet, l'identifiant du message, les articles référencés afin de les conserver dans une base de données. Les messages apparaissant en double dans le fichier sont ignorés. Pour chaque message identifié, il génère un fichier contenant le sujet et le corps du message pour permettre l'évaluation des pondérations pour les racines;

b) L'élimination des suffixes

But : l'élimination des suffixes afin d'obtenir des racines;

Traitement : application de règles de substitutions qui déterminent les suffixes à remplacer, les conditions dans lesquelles cela peut être effectué et la chaîne de caractères pour le remplacement. Nous avons utilisé dans notre implémentation l'algorithme de Porter tel qu'implanté par Frakes [FRAKES, 1992, pp. 131-160]. Aucune modification n'a été apportée à l'algorithme.

c) La classification des messages

But : établir une classification des messages sur une base sémantique. La méthode du *clustering* est utilisée.

Traitement : l'algorithme procède de la façon suivante. Un certain nombre de documents vont servir de représentants des *clusters* initiaux. Ils sont choisis pour être assez dissemblables les uns des autres.

Ensuite, les documents sont comparés avec les classes existantes. Si le document est similaire avec une ou plusieurs classes, il est attribué à la classe pour laquelle sa similarité est la plus élevée. Sinon, une nouvelle classe est créée pour ce document.

Le représentant de la classe est recalculé. Si la classe devient trop importante et qu'elle devient incohérente, celle-ci est séparée en deux classes homogènes.

Finalement, le résultat du *clustering* est conservé dans la base de données.

d) La génération des pages WWW

But : générer les pages WWW pour les articles, les index par sujet pour les *newsgroups*, les index par auteur, les index chronologiques et l'index général des *newsgroups*.

Traitement : la génération des pages WWW est assurée par les fonctions *OrganizeNewsgroup, OrganizeNewsgroupBySubjet, OrganizeNewsgroupUsingClusters, OrganizeAuthors* et *OrganizeAuthorsNewsgroup.*

Le traitement consiste à parcourir l'ensemble des articles et à générer une page WWW pour chaque message avec les ancres hypertextes adéquates.

Les index sont générés en parcourant des tables de la base de données et en générant un index par *newsgroup.*

La dernière phase du traitement *OrganizeNewsgroupUsingClusters* est particulière car elle relit les pages WWW des articles générés précédemment et y ajoute de l'information si nécessaire (si l'article n'est pas seul dans le *cluster*). Cette manière de procéder est rendue nécessaire par les limitations de la base de données employée.

e) La manipulation de la base de données

But : ces fonctions ont pour but de créer la base de données, de modifier ou de consulter des données. Toutes les fonctions de manipulation de la base de données sont regroupées dans un seul module afin de faciliter le portage éventuel vers une autre base de données et aussi pour faciliter la compilation (un seul module à précompiler). Ces fonctions sont utilisées par les différents modules et objets de l'application.

Traitement : exécution des instructions SQL nécessaires à la création de la base de données, à la modification, la consultation, l'effacement de données. Certains groupes de fonctions (*Open..., Close..., Fetch...*) sont utilisés pour parcourir des tables à l'aide d'un curseur.

f) Fonctions utilitaires

Un nombre assez important de fonctions utilitaires sont définies dans les différents modules et dans les fichiers *stdfunc.h* et *stdfunc.cpp.* Des définitions communes de type et énumération apparaissent aussi dans *stddefs.h.* Il s'agit

principalement de fonctions pour le traitement des chaînes de caractères. Elles ne seront pas décrites ici.

g) Les fonctions de menu

Les fonctions de menu sont chargées de présenter à l'utilisateur la phase de traitement qu'il désire déclencher. Une autre fonction s'occupe d'écraser ou non la base de données existante.

4.2.3 Découpe en niveaux

Les fonctionnalités d'un logiciel peuvent être classées selon le niveau auquel elles agissent. Il peut s'agir de fonctions de présentation, c'est-à-dire qu'elles s'occupent d'affichage à l'écran ou de demande à l'écran d'informations. Soit il s'agit de fonctions de traitement, ce sont alors les fonctionnalités au coeur du programme. Ou encore ce sont des fonctions gérant les données persistantes, en d'autre terme, les fonctions de la base de données.

Nous proposons ici une représentation graphique de cette découpe en niveaux (Figure 11).

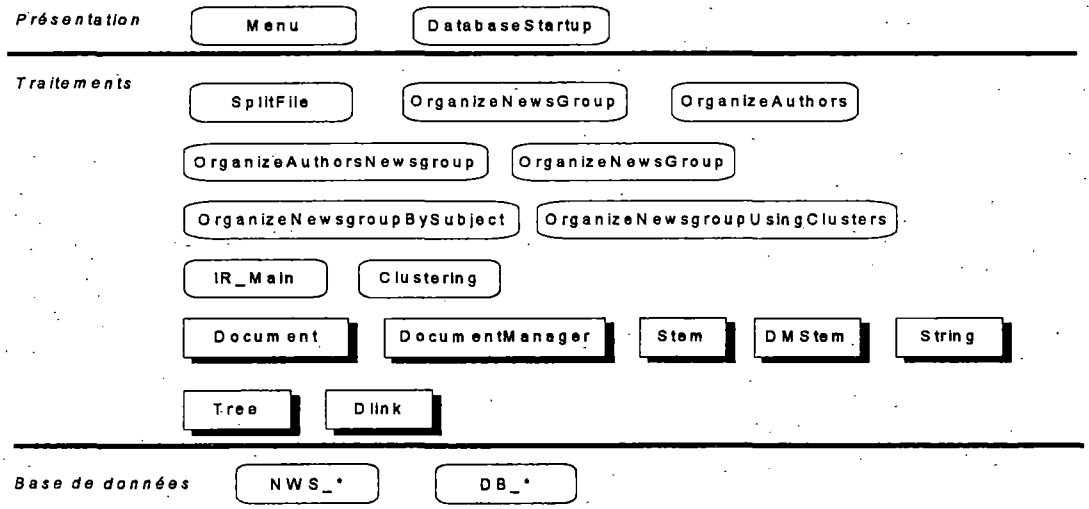


Figure 11 : découpe en niveau des fonctions

5. Les résultats

Le résultat de cette application est donc un système hypertexte reprenant les articles des *newsgroups*. Ces articles sont organisés de différentes manières :

- par des indications chronologiques : précédant/suivant, index des articles pour un *newsgroup;*
- par des indications lexicales : l'auteur, le sujet, les index d'auteurs et de sujets;
- par des liens sémantiques obtenus par le *clustering*.

Nous proposons au lecteur de consulter l'Annexe B pour un exemple d'utilisation du système hypertexte[1].

Dans l'Annexe C, le lecteur trouvera une série de documents ainsi que les liens mis en évidence par le programme pour cette collection de documents.

En ce qui concerne le temps d'exécution, il est fortement dépendant de la taille de la collection de messages. Pour illustration, un système hypertexte constitué de 931 messages provenant de 5 *newsgroups* a demandé un temps d'exécution de plus ou moins 6 heures sur un Pentium 90 Mhz, 16 Mb RAM. Nous pensons que ce sont les nombreuses entrées/sorties qui sont à l'origine de ce temps de traitement fort important.

6. L'évaluation des résultats

Comment évaluer la pertinence des liens mis à jour par notre programme ?

---

[1] Jusqu'à la défense de ce mémoire, un exemple de système hypertexte est disponible à l'URL suivante : http://www.info.fundp.ac.be/~vdh/news/index.html. La disponibilité de ce système n'est pas garantie par après.

La pertinence des liens fait partie du domaine de l'*information retrieval.* L'évaluation des résultats de ce système est malaisée pour les raisons suivantes :

- bien qu'il existe des collections typiques de documents pour les systèmes traditionnels d'*information retrieval,* il n'existe pas de collection type de documents servant à la création de systèmes hypertextes. Les différents auteurs ont tous travaillé avec des documents différents. Il n'est donc pas question de pouvoir comparer les liens entre des systèmes différents sur des collections de documents différents;

- les liens découverts par ce programme ne sont pas nécessairement les meilleurs liens possibles. Cela découle de l'algorithme de *clustering* que nous avons choisi. Celui-ci a de bonnes performances mais ne garantit pas la stabilité de la classification si, par exemple, les documents sont organisés dans un ordre différent ou si la phase d'initialisation utilise d'autres *clusters* initiaux;

- alors qu'en *information retrieval* traditionnel les mesures de la précision et de rappel servent à mesurer l'efficacité du système en réponse à une requête, il apparaît difficile de faire de même ici.

Une autre forme possible pour l'évaluation des résultats est l'enquête auprès d'utilisateurs potentiels d'un système hypertexte. En choisissant un panel assez large de personnes et une méthodologie appropriée, il serait possible d'estimer la pertinence des liens. Malheureusement, nous n'avons pas eu le temps de faire une telle enquête.

Intuitivement - et tout à fait subjectivement - voici quelques remarques auxquelles nous sommes arrivés :

- les liens découverts par ce programme sont dans la majorité des cas assez pertinents. Pour affirmer cela, nous nous basons sur le fait que les articles qui font partie d'un groupe d'articles liés (articles référencés ou articles/réponses à ces articles) font souvent partie des mêmes classes;

- il arrive qu'une classe nous semble tout à fait inappropriée. Cela peut arriver lorsque les articles contiennent des mots trop spécifiques (comme une adresse e-mail ou une « signature » d'un message) ou lorsque la longueur des messages est réduite et ne permet pas une évaluation correcte de la pondération des termes;

- la paramétrisation du logiciel (le seuil de similarité minimale, le seuil d'inertie, le nombre d'articles admissible dans les classes, l'inertie initiale d'un *cluster*, ...) est problématique car pour être raisonnablement certain des choix posés, il faudrait réaliser de multiples essais et enquêtes auprès d'utilisateurs potentiels. C'est évidemment difficile à réaliser. Nous avons déterminé ces seuils nous-mêmes sur des collections de taille réduite. Il est quasi certain qu'une paramétrisation, même légèrement différente, aboutisse à une classification totalement différente. C'est surtout vrai pour le seuil de similarité minimale qui a un rôle prépondérant dans la classification. De plus, ces paramètres dépendent de la collection de documents;

- la navigation dans le système hypertexte tel qu'existant est assez difficile. En effet, pour chaque article, nous présentons une liste d'articles appartenant au même *cluster*. Il est donc impossible de sortir du *cluster* d'articles sans passer par une autre sorte de liens. C'est une limitation sévère qu'il faudrait lever, mais que nous n'avons pu développer pour trois raisons : le manque de temps, la limitation des possibilités de la base de données mais surtout et avant tout, la question était de savoir comment présenter à l'utilisateur les classes obtenues par le *clustering* ? Nous pensons que le manque de représentation des *clusters* constitue un obstacle majeur pour une navigation aisée. Une hiérarchie des classes ou une toile d'araignée des classes serait peut-être adéquate;

- nous avons choisi d'attribuer les documents à une seule classe, sur base de la similarité maximale. Cette solution ne rencontre pas la richesse sémantique potentielle des différentes classes. Il se peut que des documents connexes soient attribués à des classes distinctes pour une différence de similarité minime. Pour résoudre ce problème, une solution serait de fusionner les classes qui sont fort semblables.

Au vu de ces résultats, nous estimons que ce prototype, bien qu'imparfait - permet la création de liens hypertextes. Ces liens sont plus ou moins pertinents suivant les collections de documents, les messages et la paramétrisation du logiciel. Nous estimons cependant que la réalisation d'un tel système nous a permis de mettre en exergue les faiblesses d'un système entièrement automatique. Les problèmes cités plus haut sont autant de freins à l'utilisation d'un tel système à des fins non expérimentales.

Conclusions et perspectives

Dans ce mémoire, nous avons présenté en toute généralité les applications hypertextes. Nous avons soulevé le problème essentiel de la création de ces systèmes. Jusqu'à présent, la norme est de créer ceux-ci manuellement.

Des recherches récentes tendent à déterminer la faisabilité de la création automatique des liens hypertextes. Les approches prises par les chercheurs sont de plusieurs types : les approches lexicales qui se basent sur la reconnaissance de structures particulières dans le texte, les approches du langage naturel qui tendent à utiliser une « compréhension » du langage pour détecter les liens entre les textes, les approches basées sur les réseaux de neurones qui visent à classifier les documents et finalement, l'approche de l'*information retrieval* qui cherche à appliquer les différentes techniques développées dans la gestion documentaire.

Dans ce mémoire, nous avons décidé de nous attarder sur l'application des techniques d'*information retrieval*. Nous en avons présenté les concepts et mécanismes et nous avons décrit quelques réalisations de systèmes pour la création de systèmes hypertextes ou pour des domaines connexes.

Afin de compléter cette étude théorique, un programme prototype a été réalisé pour la création automatique de systèmes hypertextes. L'exemple suivi a été la

classification des messages échangés sur les groupes de discussion d'Internet afin de déduire des liens entre ces messages.

Enfin, nous avons présenté les résultats de ce programme de manière tout à fait subjective. Nous avons décelé de bonnes potentialités pour ce genre de système mais notre implémentation est assez limitée. Nous avons déterminé un certain nombre de problèmes généraux.

Tout d'abord, le processus de classification n'est pas parfait. Certes la classification aboutit à des résultats honorables, mais certaines zones d'ombre existent avec, par exemple, des liens qui semblent incongrus à première vue. De même, nous nous sommes interrogés sur l'exploitation de cette classification et sur sa représentation pour l'utilisateur. Ensuite, l'important problème de la paramétrisation du logiciel est apparu. Une paramétrisation optimale de la classification est essentielle pour obtenir des résultats acceptables. Finalement, le problème de l'évaluation de la pertinence des liens doit être approfondi.

Nous proposons donc que des investigations complémentaires soient menées dans les domaines suivants : l'application de mécanismes d'*information retrieval* plus sophistiqués, la représentation des classes obtenues par le processus de *clustering*, la paramétrisation optimale de la classification et la pertinence des liens.

Même si, à première vue, ces résultats semblent mitigés, nous pensons que l'application de techniques plus sophistiquées d'*information retrieval* pour la création des systèmes hypertextes est appropriée. Nous émettons cependant des réserves sur l'opportunité d'une création strictement automatique. Les techniques de création automatique ont chacune leurs défauts et faiblesses. Nous suggérons d'effectuer des recherches complémentaires sur l'intégration de ces mécanismes à des outils auteurs afin qu'une personne puisse juger de la pertinence des liens établis par un processus automatique.

Une autre de voie de recherche que nous suggérons est l'étude de faisabilité d'un système automatique de classification faisant appel à des techniques d'intelligence artificielle. Le système expert utiliserait une base de connaissances introduite par un

expert. Imaginons que cet expert introduit dans cette base d'informations des catégories de mots clés et de concepts et qu'il identifie chacune de ces catégories à un domaine. Il serait alors possible d'attribuer un document à une catégorie en fonction des mots clés en présence.

Cette optique rejoint en partie le processus d'indexation manuel car il exige l'intervention humaine pour définir cette base de connaissances mais elle nous semble plus intéressante parce que, à coté de cette base de connaissances, le système pourrait comprendre des règles pour établir une classification optimale selon une logique propre. Un désavantage de cette technique est qu'elle demande l'intervention d'un expert et que les connaissances introduites sont spécifiques à un domaine. Dans le cas d'un système expert, l'intervention humaine se fait au préalable à l'utilisation du programme. Dans le cadre d'une construction semi-automatique par des procédés de *clustering*, l'intervention humaine se fait après exécution du programme. Il reste donc à déterminer à quel moment l'intervention d'une personne est la plus appropriée.

Bibliographie

[BALASU., 1994] V. BALASUBRAMANIAN, *State of the art Review on Hypermedia Issues and Application*, Graduate School of Management, Rutgers University, New-Jersey, 1994

[BAUER, 1995] Niels K. Bauer, *AutoLink : An Automated Link Generator for Building Hypertext*, Department of Computer Science, Texas A&M University, USA, 1995

[BLUSTEIN, 1993] William James BLUSTEIN, *An Evaluation of Tools for Converting Text To Hypertext*, Departement of Computer Science, University of Western Ontario, Canada, December 1993

[BUSH, 1945] Vannevar BUSH, *As We May Think*, The Atlandic Monthly, July 1945

[CAMPBELL, 1988] B. CAMPBELL, J.M. GOODMAN, *HAM : A general purpose hypertext abstract machine*, in Communications of the ACM, vol. 31, n° 7, July 1988, pp. 856-861

[CIIR, 1994] *Inquiry - Query formulation*, Information Retrieval Laboratory, Department of Computer Science, University of Massachusetts, Amherts, USA, URL : `http://ciir.cs.umass.edu/`, 1994

[CONKLIN, 1987] Jeff CONKLIN, *Hypertext : An Introduction and Survey*, IEEE Computer, September 1987

[CONRAD, 1994] Jack G. CONRAD, Mary Hunter UTT, *A System for Discovering RelationShips by Feature Extraction from Text Databases*, Center for Intelligent Information Retrieval, University of Massachusetts, Amherts, U.S.A., 1994

[COOK, 1988] P. COOK, *Multimédia technology : An encyclopédia publisher's perspective*, in Ambron S. and Hooker K., *Interactive Multimédia : Visions of Multimedia for Developers, Educators & Information Providers*, Microsoft Press, 1988, pp. 217-240

[CROFT, 1988] W. Bruce CROFT, Pasquale SAVINO, *Implementing Ranking Strategies Using Text Signatures*, ACM Transactions on Office Information Systems, vol. 6, n° 1, pp. 42-62, January 1988

[EASTGATE, 1995A] EASTGATE Inc., *Storyspace - A Hypertext Tool for Writerss and Readers*, URL : `http://www.eastgate.com/Storyspace.html`

[EASTGATE, 1995B] EASTGATE Inc, *Map of the Immediate Neighberhood*, URL : `http://raven.ubalt.edu/Moulthrop/hypertexts/hoptext/Map88158.html` et *About this Map*, URL : `http://raven.ubalt.edu/Moulthrop/hypertexts/hoptext/About_This_Map0863.html`

[FELDMAN, 1995] Ian FELDMAN, *Re: About this map*, URL : `http://raven.ubalt.edu/Moulthrop/hypertexts/hoptext/map_critique.html`

[FID/IFIP67, 1968] Proceedings of the F.I.D./I.F.I.P. Joint Conference - Rome, June 14-17 1967, *Mechanized Information Storage, Retrieval and Dissemination*, North-Holland Publishing Compagny, Amsterdam, Netherland, 1968

[FRAKES, 1992] William B. FRAKES, Ricardo BAESA-YATES, *Information Retrieval - Data structures & algorithms*, Prentice-Hall, 1992

[GEORGEL, 1993] A. GEORGEL, D. BOUAM, W.A. Turner, *L'analyse statistique au service de la navigation hypertextuelle : la construction d'un modèle neuronal d'accès aux informations documentaires*, CERESI/CNSR, Information Science Research Group in Revue française de bibliométrie, n° 12, Octobre 1993

[GOFFINET, 1995] Luc GOFFINET, *Le Multimédia et ses Systèmes Auteurs*, Institut d'Informatique, Facultés Universitaires Notre-Dame de la Paix, 1995

[KIMBRELL, 1988] Roy E. KIMBRELL, *Searching for Text ? Send an N-Gram* in BYTE, May 1988, pp. 297-312

[LEHNERT, 1992] Wendy G. LEHNERT, *Automating the Construction of a Hypertext System for Scientific Literature,* Proceeding of the AAAI Workshop on Communicating Scientific and Technical Knowledge, 1992

[LORENZ., 1994A] Jeffrey A. LORENZEN, *Automatic Creation of Hypertext Links for Legal Cictations (public version),* Department of Computer Science, University of Utah, U.S.A., October 16, 1994

[LORENZ., 1994B] Jeffrey A. LORENZEN, *Processing the Selection of a Hypertext Link : A technical Description of the LINK Module and LINDEX,* Department of Computer Science, University of Utah, U.S.A., October 16, 1994

[LUHN, 1958] H. P. LUHN, *The automatic creation of literature abstracts,* IBM Journal of Research and Development, pp. 159-165, 1958

[McCALLUM, 1989] John McCALLUM, *Indexing of Multimedia Information,* Technical Report, Departement of Information Systems and Computer Science, National University of Singapore, Singapore, November 1989

[MINKER, 1972] J. MINKER, G.A WILSON, B.H. ZIMMERMAN, *An evaluation of query expandsion by the addition of clustered terms for a document retrieval system,* Information Storage and Retrieval, vol. 8, pp. 329-348

[NIELSEN, 1990] Jakob NIELSEN, *Hypertext and Hypermedia,* Academic Press INC., 1990

[NIELSEN, 1990B] Jakob NIELSEN, *The Art of Navigating through Hypertext,* Communications of the ACM, March 1990

[NIELSEN, 1991] Jakob NIELSEN, *Panel Discussion - The Nielsen Ratings : Hypertext Reviews,* Proceedings of Hypertext '91, 1991

[NORDH., 1994] Cheng KOK KOH, Bernd NORDHAUSEN, Tat SENG CHUA, *Automatic Hypertext Link Creation,* Institute of Systems Science, National University of Singapore, November 22, 1994

[PORTER, 1980] M. F. PORTER, *An algorithm for suffix stripping,* Program 14 (vol 3), pp. 130-137, July 1980

[RFC850] Mark R. HORTON, *Standard for Interchange of USENET messages,* RFC 850, June 1983

[SALTON, 1968A] Gerard SALTON, *Automatic Information Organization and Retrieval,* McGraw-Hill, New-York, 1968

[SALTON, 1968B] Gerard SALTON, *Search Strategy and the Optimization of Retrieval Effectiveness*, in [FID/IFIP67, 1968], pp. 73-107

[SALTON, 1970] Gerard SALTON, *Automatic text analysis*, Science, n° 168, pp. 335-343, 1970

[SALTON, 1983] Gerard SALTON, Michael J. McGILL, *Introduction to Modern Information Retrieval*, McGraw-Hill International, 1983

[SALTON, 1994A] Gerard SALTON, James ALLAN, Chris BUCKLEY, *Automatic Structuring of Large Text Files* in Communications of the ACM, vol. 37, n° 2, February 1994, pp. 97-108

[SALTON, 1994B] Gerard SALTON, James ALLAN, *Automatic Text Decomposition and Structuring*, Department of Computer Science, Cornell University, USA, 1994

[SANDERS., 1994] Mark SANDERSON, Keith van RIJSBERGEN, *NRT (News Retrieval Tool)*, 11 June 1994, Department of Computing Science, University of Glasgow

[SHNEID., 1989] B. SHNEIDERMAN, *Reflections on authoring, editing and managing hypertext*, in E. Barrett, *The Society of Text*, MIT Press, Cambridge, pp. 115-131

[SPARCK, 1971] Karen SPARK JONES, *Automatic Keyword Ckassification for Information Retrieval*, Butterworths, London, England, 1971

[VANRIJS, 1979] C. J. van RIJSBERGEN, *Information Retrieval - 2nd edition*, Butterworths, London, England, 1979

[VICKERY, 1970] B. C. VICKERY, *Techniques of Information Retrieval*, Butterworths, London, England, 1970

[WRIGHT, 1991] Patricia WRIGHT, *Cognitive Overheads and Prosthesis : Some issue in Evaluating Hypertexts*, Proceedings of Hypertext '91, 1991

[YU, 1977] C. T. YU, Gerard Salton, *Effective information retrieval using term accuracy*, Communications of the ACM, vol. 20, pp. 135-142, 1977

Annexe A
Le code du programme

Pour des raisons techniques, les pages suivantes ne sont pas numérotées. Le nom du fichier est inclu dans l'en-tête de chaque page.

Les fichiers sont dans l'ordre suivant :

- fichiers d'en-tête *(*.h)*
- fichiers source C++ *(*.cpp)*
- fichier de configuration *(irnews.ini)*
- fichier de directives de compilation *(irnews.mak)*

Le fichier *dbirnws.cpp* est présenté dans sa forme non-précompilée, c'est-à-dire que les instructions SQL appaissent en toutes lettres.

```
#ifndef __CLUSTERS__
#define __CLUSTERS__

void SimilarityTwoByTwo();
void Clustering();

#endif
```

```
#ifndef __DBIRNWS__
#define __DBIRNWS__

void DB_CheckError();
int DB_ConnectTo();

void DB_AddStem(int pdocID, int pstemID, char pstem[21], int poccurence, float
    pweight);
void DB_GetStem(int pdocID, int pstemID, char pstem[21], int& poccurence, float&
    pweight);
void DB_UpdateStem(int pdocID, int pstemID, int poccurence, float pweight);
void DB_AddDocument(int pdocID, int pnumstem, char * title);
void DB_GetDocument(int pdocID, int & pnumstem, char * title);
int  DB_GetNumDocument();
void DB_AddSimilarity(int pdocID1, int pdocID2, float psimilarity);
void DB_GetSimilarity(int pdocID1, int pdocID2, float & psimilarity, int &
    status);
void DB_DeleteClusters();
void DB_AddClusterData(int cluster, int docID);

void DB_Commit();

void NWS_CreateDatabase(int force);
void NWS_CreateIndexes();
void NWS_Commit(void);

int NWS_GetNumArticle(void);
void NWS_AddArticle(int artnum, char * artid, char * artfrom, char * artsubject,
    char * artsub, char * sd, char * st, int nwsgrp, int refcount);
int NWS_CheckArticle(char * artid);
void NWS_AddArticleReference(char * mess_id, char * areference, int reflevel);

void NWS_OpenNewsGroup(int nwsgrp, int & status);
void NWS_ArticleFetch(int & anum, char * artid, char * artfrom, char *
    artsubject, int & refcount, int & status);
void NWS_CloseNewsGroup(void);

int NWS_CheckArticleFromAuthor(char * artfrom);

void NWS_OpenArticleFrom(char * artfrom, int & status);
void NWS_ArticleFromFetch(int & anum, char * artsubject, char * artdate, char *
    arttime, int & refcount, int & status);
void NWS_CloseArticleFrom(void);

void NWS_OpenAuthors(int & status);
void NWS_AuthorFetch(char * artfrom, int & numberart, int & status);
void NWS_CloseAuthors(void);

void NWS_OpenAuthorsNewsgroup(int nwsgrp, int & status);
void NWS_AuthorNewsgroupFetch(char * artfrom, int & numberart, int & status);
void NWS_CloseAuthorsNewsgroup(void);

void NWS_InsertInAuthorTable(void);
void NWS_AuthorInFile(int authornum, char * artfrom);
void NWS_GetAuthorInfo(char * artfrom, int & numberart, int & authornum);
void NWS_GetAuthorOnlyArticle(char * artfrom, int & anum );

void NWS_GetArticleByID(char * artid, char * artsubject, char * artfrom, char *
    artdate, char * arttime, int & refcount, int & anum, int & present);
void NWS_GetArticleByNum(int artnum, char * artsubject, char * artfrom, int &
    present);
void NWS_GetReferencedArticle(char * artid, int level, char * referenced);

int NWS_GetClusterCountButItself(int cluster, int docid);
void NWS_OpenNewsByCluster(int cluster, int docid, int & status);
void NWS_FetchArticleByCluster(int & artnum, int & status);
void NWS_CloseNewsByCluster(void);
int NWS_GetClusterForArticle(int docid);
void NWS_GetDocIDForArticle(int i, int & artdocid);
void NWS_GetArticleByDocID(int i, int & artnum);
void NWS_AddSubject(char * artsubject, char * artid, char * artdate, char *
    arttime, int nwsgrp, int artnum);

void NWS_OpenBySubject(int nwsgrp, int & status);
void NWS_FetchBySubject(char * artsubject, char * artid, int & artnum, int &
    status);
void NWS_CloseBySubject();

#endif
```

```
#ifndef __DLINK__
#define __DLINK__

// The cell_dlink class defines a cell in the double linked list.
// The cell is composed of two pointers : one for the previous cell in list and
    another
// for its successor. The cell contains information in another pointer.
// When created, the pointers are all set to NULL.

template<class TT>
class cell_dlink
{
   public :
   cell_dlink* prev;
   cell_dlink* succ;
   TT* info;

   cell_dlink() {prev=succ=NULL;info=NULL;};
   virtual ~cell_dlink() {};
};

// The dlink class defines a double linked list. This class relies on the class
    cell_dlink.
// dlink manages the list, i.e. add a new element to the list, remove one
    element, delete all
// the list, ...
// For debugging purposes, a display method exists.

template<class T>
class dlink
{
   public :
   cell_dlink<T>* first; // pointer to the first cell in the list
   cell_dlink<T>* last; // pointer to the last cell in the list
   cell_dlink<T>* current; // pointer to the current cell in the list. The list
       is empty if
                     // current is NULL.
   int num_elem, current_pos; // the number of elements in the list and the rank
       of the item
                        // pointed by current

   public :

   int AddAtBegin(T* elem);
   int AddAtEnd(T* elem);

   int InsertBeforeCurrent(T* elem);
   int InsertAfterCurrent(T* elem);
   int InsertAtPos(T* elem, int pos);

   int MoveToNext();
   int MoveToPrevious();
   int MoveTo(int pos);

   int MoveAtEnd();
   int MoveAtBegin();

   T* GetCurrentInfo(int& status);
   int SetCurrentInfo(T* elem);

   int DeleteCurrentInfo();
   int DeleteAllInfo();

   int DeleteCurrent();
   int DeleteAtBegin();
   int DeleteAtEnd();
   int DeleteAtPos(int pos);

   void Display();
   void DisplayReverse();

   dlink();
   virtual ~dlink();
};
// The constructor for the dlink class
// All information is initialized to ensure proper functionning

template<class T> dlink<T>::dlink()
{
   first=NULL;
   last=NULL;
   current=NULL;
   num_elem=0;
   current_pos=0;
};

// The destructor for the dlink class
// All the cells making the list are destroyed. BUT the information is NOT
    deleted !!!
// It is the responsability of the programmer to ensure proper deletion by him
    self or
// by calling first DeleteAllInfo.

template<class T> dlink<T>::~dlink()
{
   cell_dlink<T>* curr=first;
   cell_dlink<T>* tmp;

   while (curr)
   {
      tmp=curr->succ;
      delete curr;
      curr=tmp;
   };

   first=last=current=NULL;
   num_elem=current_pos=0;
};

// Add a new cell at the end of the list.
// Proper check is made to ensure proper treatment of (non-)empty list.
// Current is not modified, but last is changed to point to the new cell.
// The number of element in the list is incremented
// Return true if succes, otherwise false (memory low).

template<class T> int dlink<T>::AddAtEnd(T* elem)
{
   cell_dlink<T>* newcell=new cell_dlink<T>; // Creates the new cell and put the
       info

   if (!newcell) return false; // Not enough memory

   newcell->info=elem;
   newcell->succ=NULL; // Inserted as last, then no sucessor.

   num_elem++; // One more element in the list

   if (first)
```

```
    {
       // First is true(<>0), we have a non empty list and last is correctly
           positionned
       last->succ=newcell;
       newcell->prev=last;
       last=newcell;
    }
    else
    {
       //First is false(=0), then this is the first element. we add it to the
           list
       first=last=current=newcell;
       newcell->prev=newcell->succ=NULL;
       current_pos=1;
    };

    return true;
}

// Add a new cell at the beginning of the list.
// Proper checkade to ensure proper treatment of (non-)empty list
// Current is not modified but first is changed to the new cell.
// The number of element in the list is incremented.
// Returns true if successful, false otherwise

template<class T> int dlink<T>::AddAtBegin(T* elem)
{
    cell_dlink<T>* newcell=new cell_dlink<T>; // Make a new cell and put info in
        it.

    if (!newcell) return false; // Not enough memory ?

    newcell->info=elem;
    newcell->prev=NULL; // No previuos cell in list

    num_elem++; // A new element has been added.

    if (first)
    {
       // If first is true(<>0) then the list is not empty.
       first->prev=newcell;
       newcell->succ=first;
       first=newcell;
       current_pos++; // The position of the current element shifts
    }
    else
    {   // The list was empty. It's the firstt cell in it.
       first=last=current=newcell;
       current_pos=1;
       newcell->prev=newcell->succ=NULL;
    };

    return true;
}

// Add a new cell to the list before the current one.
// Returns true if successful, false otherwise

template<class T> int dlink<T>::InsertBeforeCurrent(T* elem)
{
    if (!current) // List empty ?
       return AddAtBegin(elem); // Adding it at the beginning of list

    cell_dlink<T>* newcell=new cell_dlink<T>; // Not empty. Create the newcell
    if (!newcell) return false; // Not enough memory ?

    newcell->info=elem; // Set the info in the cell

    newcell->prev=current->prev; // Set the pointers of the new cell and adjust
        the
                                 // current cell's pointers
    current->prev->succ=newcell;
    newcell->succ=current;
    current->prev=newcell;
    num_elem++; // we have added a new cell
    current_pos++; // the current cell shifts
    return true; // Success !
}

// Add a new cell after the current one.
// Return true if successful, false otherwise.

template<class T> int dlink<T>::InsertAfterCurrent(T* elem)
{
    if (!current) // List empty ?
       return AddAtBegin(elem); // Adding it at the beginning of the list

    cell_dlink<T>* newcell=new cell_dlink<T>; // Create the new cell

    if (!newcell) return false; // Out of memory

    newcell->info=elem; // set the info and adapt pointers
    newcell->prev=current;
    current->succ->prev=newcell;
    newcell->succ=current->succ;
    current->succ=newcell;
    num_elem++; // we have added a new element in the list

    return true; // Success !
}

// Insert a new element in the list, shifting all the following elements.
// Returns true is successful, false otherwise.
// The position specified must be within the present list.

template<class T> int dlink<T>::InsertAtPos(T* elem, int pos)
{
    if ((!current) || (pos<1) || (pos>num_elem)) // List empty or position out of
        range
       return false; // Get out !

    if (pos==1)
        return AddAtBegin(elem);

    cell_dlink<T>* cur=current;
    int cur_pos=current_pos;

    if (cur_pos < pos) // Is the current cell inferior to the one we are looking
        for
       while (cur_pos<pos)
       {
          cur_pos++; // Move now
          cur=cur->succ;
       }
    else while (cur_pos>pos)
       {
          cur_pos--; // move now
          cur=cur->prev;
```

```
};

// Now, cur and cur_pos points to the cell we should shit

cell_dlink<T>* newcell=new cell_dlink<T>; // Create a new cell

if (!newcell) return false; // out of memory ?

newcell->info=elem; // Set the info in the cell

newcell->prev=cur->prev; // Set the pointers of the new cell and adjust the
                         // current cell's pointers
cur->prev->succ=newcell;
newcell->succ=cur;
cur->prev=newcell;
num_elem++; // we have added a new cell
if (pos<=current_pos)
   current_pos++; // if the cell was introduced before the current one, it
       has shifted
return true; // Success !
}

// Makes the next cell in the list current if possible
// Returns true is success, false otherwise

template<class T> int dlink<T>::MoveToNext()
{
   if (current) // List not empty ?
   {
      if (current->succ) // does a succesor exist ?
      {
         current=current->succ; // we have a next element. so get there
         current_pos++;
         return true;
      }
      else return false; // no succesor. we stay where we are...
   }
   else return false; // no element in list
}

// Makes the previous cell in the list current if possible
// If succes, returns true, false otherwise

template<class T> int dlink<T>::MoveToPrevious()
{
   if (current) // List not empty ?
   {
      if (current->prev) // Does a previous element exist ?
      {
         current=current->prev; // move to the previous element.
         current_pos--;
         return true;
      }
      else return false; // no previous element in list
   }
   else return false; // List empty
}

// Move to the cell whose rank is given as argument
// If successful, returns true, false otherwise

template<class T> int dlink<T>::MoveTo(int pos)
{
   if (current && (pos>=1) && (pos<=num_elem)) // List not empty and cell in
       range of list ?
   {
      if (current_pos < pos) // Is the current cell inferior to the one we are
          looking for
         while (current_pos<pos)
         {
            current_pos++; // Move now
            current=current->succ;
         }
      else while (current_pos>pos)
         {
            current_pos--; // move now
            current=current->prev;
         }
      return true;
   }
   else return false; // List empty or out of range.
}

// Move to the beginning of the list.
// Returns true if successful, false otherwise.

template<class T> int dlink<T>::MoveAtBegin()
{
   if (current) // List empty ?
   {
      current=first; // Go to the first element !
      current_pos=1;
      return true;
   }
   else return false; // List empty !
};

// Move to the end of the list
// Returns true is successful, false otherwise.

template<class T> int dlink<T>::MoveAtEnd()
{
   if (current) // List empty ?
   {
      current=last; // No, so get there !
      current_pos=num_elem;
      return true
      ;
   }
   else return false; // List empty !
}

// Returns the current info if possible, NULL otherwise.
// Status is true is successful, false otherwise
// It is possible to have status true and a NULL return value !

template<class T> T* dlink<T>::GetCurrentInfo(int& status)
{
   if (current) // List empty ?
   {
      status=true; // No, return the info
      return current->info;
   }
   status=false; // List empty, so return
   return NULL;
}

// Set the info of the current cell. No deletion of the info is made.
```

```
// Returns true is successfull, false otherwise

template<class T> int dlink<T>::SetCurrentInfo(T* elem)
{
   if (current) // List empty ?
   {
      current->info=elem; // No, set the info
      return true;
   }
   return false; // List empty !
}

// Delete the info of the current cell in list.
// Set the info to NULL.
// Returns true if successful, false otherwise !

template<class T> int dlink<T>::DeleteCurrentInfo()
{
   if (current) // List empty ?
   {
      delete current->info; // Delete it !
      current->info=NULL;
      return true;
   }
   else return false; // List empty !
}

// Delete the info of all the cells in the list.
// Do not remove the cell themselves !!!
// Returns true if successful, false otherwise

template<class T> int dlink<T>::DeleteAllInfo()
{
   if (current) // List empty ?
   {
      cell_dlink<T>* tmp=first; // Iterate through the list and delete the info.
      while (tmp)
      {
         delete tmp->info;
         tmp->info=NULL;
         tmp=tmp->succ;
      }
      return true; // successful !
   }
   else return false; // List empty !
}

// Delete the first cell of the list. The info is NOT deleted (see
    DeleteCurrentInfo)
// Returns true if successful, false otherwise.

template<class T> int dlink<T>::DeleteAtBegin()
{
   if (!current) // List empty ?
      return false; // get out now !

   cell_dlink<T>* tmp=first;

   if (first->succ) // More than one cell ?
      first->succ->prev=NULL; // Make the second cell's previous pointer point
          to NULL

   if (first==current) // Is the first cell the current cell ?
      { // Yes, so
         if (first->succ) // More than one cell in the list ?
            current=current->succ; // Make the second one the current.
         else {
            first=last=current=NULL; // Only one cell in list... Delete it
            num_elem=current_pos=0;  // Set value for an empty list
            delete tmp;
            return true;
            };
      };

   first=first->succ; // The first cell will become the next one.

   delete tmp; // Delete the cell
   num_elem--; // Decrease the number of element in the list
   return true;

}

// Delete the last item of the list. The info part is NOT deleted (see
    DeleteCurrentInfo())
// Returns true if successful, false otherwise

template<class T> int dlink<T>::DeleteAtEnd()
{
   if (!current)  // List empty ?
      return false; // Get out now

   cell_dlink<T>* tmp=last;

   if (last->prev) // A previous cell exists ?
      last->prev->succ=NULL; // Then tell it that there is no following cell.

   if (last==current) // Is the last cell the current one ?
      { // Yes, so
         if (last->prev) // does a previous cell exist ?
            {
               current=current->prev; // Make it the current one !
               current_pos--; // Decrease the rank of the current cell
            }
         else
         {
            first=last=current=NULL; // Only cell in list. Delete it
            num_elem=current_pos=0;  // Set value for an empty list.
            return true;
         }
      };

   last=last->prev; // The last will be the lastest's cell previous.

   delete tmp; // Delete the cell
   num_elem--; // We have deleted an element
   return true; // It's over...
}

// Delete the current cell. The info is NOT deleted by this function (see
    DeleteCurrentInfo)
// Returns true is successful, false otherwise.

template<class T> int dlink<T>::DeleteCurrent()
{
   if (!current)  // List empty ?
      return false; // Get out

   if (current==first)  // First item ?
```

```
      return DeleteAtBegin();

   if (current==last) // Last item in list ?
      return DeleteAtEnd();

   current->prev->succ=current->succ; // Adjust pointers of adjacent cells
   current->succ->prev=current->prev;

   cell_dlink<T>* tmp=current;
   tmp=current->succ; // Let current point to the following cell.

   delete current; // Delete the cell

   current=tmp;
   num_elem--; // We have deleted a cell
   return true;
}

// Display the information contained in the list for debugging

template<class T> void dlink<T>::Display()
{
   if (current) // List empty ?
   {
      cell_dlink<T>* tmp=first; // No, print the info
      while (tmp)
      {
         tmp=tmp->succ;
      }
   }
}

template<class T> void dlink<T>::DisplayReverse()
{
   if (current)
   {
      cell_dlink<T>* tmp=last;
      while (tmp)
      {
         tmp=tmp->prev;
      }
   }
}

#endif
```

```
#ifndef __DOCUMENT__
#define __DOCUMENT__

#include "dlink.h"
#include "tree.h"
#include "stemc.h"

#define FNLENGTH 170
#define LINELENGTH 1024
#define MOTLENGTH 80

#define MINSTEMLENGTH 2
#define MAXSTEMLENGTH 20

//
// Class document
//

class documentManager;

class document
{
   public :
   int docID;
   char * fichier;
   char titre[21];
   FILE * fileh;
   documentManager * managedBy;
   tree<stem> * stems;
   dlink<stem> * dlstems;

   void Display();

   virtual char * FirstPass();
   virtual char * SecondPass();
   void CalculateWeight();

   void Store();
   void Update();
   void DeleteLists();
   void Retrieve();

   void Copy(document * & doccopied);

   document();
   document(char * fich, documentManager * manager, char * ptitre);
   virtual ~document();
};

//
// Class documentManager
//

class documentManager
{
   public :
   int lastID;
   dlink<document> * manageddocs;
   tree<mystring> * stopwords;
   tree<dmstem> * stems;
   dlink<dmstem> * dlstems;

   void Add(document* doc);
   void DeleteDocuments();
   void EleminateStems();

   void FirstPass();
   void SecondPass();

   void CalculateWeight();

   void Display();
   void ReadStopWord(char * fich);

   documentManager();
   ~documentManager();
};

#endif
```

```
#ifndef __IRNEWS__

#define __IRNEWS__

#include <stdlib.h>

#include <stdio.h>

#include <iostream.h>

#include <string.h>

#include <dir.h>

#include "stddefs.h"

#include "ocelot.h"

#include "dbirnws.h"

#include "dlink.h"

#include "tree.h"
#include "document.h"

#endif
```

```
#ifndef __MYSTRING__

#define __MYSTRING__

#include <iostream.h>

//
// Class mystring

//

class mystring

{

   public :

   char * string;

   mystring();

   mystring(char * is);

   virtual ~mystring();

   int operator<(mystring&);

   int operator>(mystring&);
   int operator==(mystring&);
};

//
// Function prototype

//

ostream& operator<<(ostream& s, mystring& n);

ostream& operator<<(ostream& s, mystring * n);

#endif
```

```
#ifndef __SPLITNWS__
#define __SPLITNWS__

void split_main();
void SplitFile(char * filename, int nwsgrp);
void OrganizeNewsGroup(int nwsgrp, char * nwsgrpdesc);
void OrganizeAuthors();
void OrganizeAuthorsNewsgroup(int nwsgrp, char * nwsgrpdesc);
void OrganizeNewsgroupUsingClusters();
void OrganizeNewsgroupBySubject(int nwsgrp, char * nwsgrpdesc);

#endif
```

```
#ifndef __STDDEFS__
#define __STDDEFS__
#include <ctype.h>
#include <fstream.h>

#if !defined(BOOL_DEFINED) && !(defined _bool_h)
#define BOOL_DEFINED
enum bool {false=0,true=1};
#endif

//#define true 1
//#define false 0

#define MAXLONGFILENAME 63;
#define MAXLONGOFTITLE 160;
#define LONFOFLINE 80;

// Global variable

#endif
```

```
#ifndef __STDFUNC__
#define __STDFUNC__

int ReadLine(FILE * fileh, char * line, int & length);
inline int IsDelimiter(char ch);
inline int IsAlphaNum(char ch);
inline int IsAlpha(char ch);
int AllNum(char * s);
int GetNextWord(char * mot, char * & lastpos);

#endif
```

```
/*********************************   stem.h   ***********************************

   Purpose: Header file for an implementation of the Porter stemming
            algorithm.

   Notes:   This module implemnts a fast stemming function whose results
            are about as good as any other.
**/

#ifndef __STEM__
#define __STEM__

/*****************************************************************************/
/*****************************  Public Routines  *****************************/

int Stem( char *word );      /* returns 1 on success, 0 otherwise */

/*********************************   stem.c   ***********************************

   Purpose:    Implementation of the Porter stemming algorithm documented
               in: Porter, M.F., "An Algorithm For Suffix Stripping,"
               Program 14 (3), July 1980, pp. 130-137.

   Provenance: Written by B. Frakes and C. Cox, 1986.
               Changed by C. Fox, 1990.
                  - made measure function a DFA
                  - restructured structs
                  - renamed functions and variables
                  - restricted function and variable scopes
               Changed by C. Fox, July, 1991.
                  - added ANSI C declarations
                  - branch tested to 90% coverage

   Notes:      This code will make little sense without the the Porter
               article.  The stemming function converts its input to
               lower case.
**/

/************************   Standard Include Files   *************************/

#include <stdio.h>
#include <string.h>
#include <ctype.h>

/*****************************************************************************/
/******************   Private Defines and Data Structures   ******************/

#define FALSE                          0
#define TRUE                           1
#define EOS                           '\0'

#define IsVowel(c)        ('a'==(c)||'e'==(c)||'i'==(c)||'o'==(c)||'u'==(c))

typedef struct {
          int id;                 /* returned if rule fired */
          char *old_end;          /* suffix replaced */
          char *new_end;          /* suffix replacement */
          int old_offset;         /* from end of word to start of suffix */
          int new_offset;         /* from beginning to end of new suffix */
          int min_root_size;      /* min root word size for replacement */
          int (*condition)(char *);     /* the replacement test function */
          } RuleList;

static char LAMBDA[1] = "";       /* the constant empty string */
static char *end;                 /* pointer to the end of the word */

/*****************************************************************************/
/********************   Private Function Declarations   **********************/

static int WordSize( char * word );
static int ContainsVowel( char * word );
static int EndsWithCVC( char * word );
static int AddAnE( char * word );
static int RemoveAnE( char * word );
static int ReplaceEnd( char * word, RuleList * rule );

/*****************************************************************************/
/******************   Initialized Private Data Structures   ******************/

static RuleList step1a_rules[] =
          {
            {101,  "sses",      "ss",     3,  1, -1,  NULL},
            {102,  "ies",       "i",      2,  0, -1,  NULL},
            {103,  "ss",        "ss",     1,  1, -1,  NULL},
            {104,  "s",         LAMBDA,   0, -1, -1,  NULL},
            {000,  NULL,        NULL,     0,  0,  0,  NULL},
          };

static RuleList step1b_rules[] =
          {
            {105,  "eed",       "ee",     2,  1,  0,  NULL},
            {106,  "ed",        LAMBDA,   1, -1, -1,  ContainsVowel},
            {107,  "ing",       LAMBDA,   2, -1, -1,  ContainsVowel},
            {000,  NULL,        NULL,     0,  0,  0,  NULL},
          };

static RuleList step1b1_rules[] =
          {
            {108,  "at",        "ate",    1,  2, -1,  NULL},
            {109,  "bl",        "ble",    1,  2, -1,  NULL},
            {110,  "iz",        "ize",    1,  2, -1,  NULL},
            {111,  "bb",        "b",      1,  0, -1,  NULL},
            {112,  "dd",        "d",      1,  0, -1,  NULL},
            {113,  "ff",        "f",      1,  0, -1,  NULL},
            {114,  "gg",        "g",      1,  0, -1,  NULL},
            {115,  "mm",        "m",      1,  0, -1,  NULL},
            {116,  "nn",        "n",      1,  0, -1,  NULL},
            {117,  "pp",        "p",      1,  0, -1,  NULL},
            {118,  "rr",        "r",      1,  0, -1,  NULL},
            {119,  "tt",        "t",      1,  0, -1,  NULL},
            {120,  "ww",        "w",      1,  0, -1,  NULL},
            {121,  "xx",        "x",      1,  0, -1,  NULL},
            {122,  LAMBDA,      "e",     -1,  0, -1,  AddAnE},
            {000,  NULL,        NULL,     0,  0,  0,  NULL},
            };

static RuleList step1c_rules[] =
          {
            {123,  "y",         "i",       0,  0, -1,  ContainsVowel},
            {000,  NULL,        NULL,     0,  0,  0,  NULL},
          };

static RuleList step2_rules[] =
          {
            {203,  "ational",   "ate",    6,  2,  0,  NULL},
            {204,  "tional",    "tion",   5,  3,  0,  NULL},
            {205,  "enci",      "ence",   3,  3,  0,  NULL},
            {206,  "anci",      "ance",   3,  3,  0,  NULL},
```

```
            {207,  "izer",      "ize",    3,  2,  0,  NULL},
            {208,  "abli",      "able",   3,  3,  0,  NULL},
            {209,  "alli",      "al",     3,  1,  0,  NULL},
            {210,  "entli",     "ent",    4,  2,  0,  NULL},
            {211,  "eli",       "e",      2,  0,  0,  NULL},
            {213,  "ousli",     "ous",    4,  2,  0,  NULL},
            {214,  "ization",   "ize",    6,  2,  0,  NULL},
            {215,  "ation",     "ate",    4,  2,  0,  NULL},
            {216,  "ator",      "ate",    3,  2,  0,  NULL},
            {217,  "alism",     "al",     4,  1,  0,  NULL},
            {218,  "iveness",   "ive",    6,  2,  0,  NULL},
            {219,  "fulnes",    "ful",    5,  2,  0,  NULL},
            {220,  "ousness",   "ous",    6,  2,  0,  NULL},
            {221,  "aliti",     "al",     4,  1,  0,  NULL},
            {222,  "iviti",     "ive",    4,  2,  0,  NULL},
            {223,  "biliti",    "ble",    5,  2,  0,  NULL},
            {000,  NULL,        NULL,     0,  0,  0,  NULL},
        };

static RuleList step3_rules[] =
        {
            {301,  "icate",     "ic",     4,  1,  0,  NULL},
            {302,  "ative",     LAMBDA,   4, -1,  0,  NULL},
            {303,  "alize",     "al",     4,  1,  0,  NULL},
            {304,  "iciti",     "ic",     4,  1,  0,  NULL},
            {305,  "ical",      "ic",     3,  1,  0,  NULL},
            {308,  "ful",       LAMBDA,   2, -1,  0,  NULL},
            {309,  "ness",      LAMBDA,   3, -1,  0,  NULL},
            {000,  NULL,        NULL,     0,  0,  0,  NULL},
        };

static RuleList step4_rules[] =
        {
            {401,  "al",        LAMBDA,   1, -1,  1,  NULL},
            {402,  "ance",      LAMBDA,   3, -1,  1,  NULL},
            {403,  "ence",      LAMBDA,   3, -1,  1,  NULL},
            {405,  "er",        LAMBDA,   1, -1,  1,  NULL},
            {406,  "ic",        LAMBDA,   1, -1,  1,  NULL},
            {407,  "able",      LAMBDA,   3, -1,  1,  NULL},
            {408,  "ible",      LAMBDA,   3, -1,  1,  NULL},
            {409,  "ant",       LAMBDA,   2, -1,  1,  NULL},
            {410,  "ement",     LAMBDA,   4, -1,  1,  NULL},
            {411,  "ment",      LAMBDA,   3, -1,  1,  NULL},
            {412,  "ent",       LAMBDA,   2, -1,  1,  NULL},
            {423,  "sion",      "s",      3,  0,  1,  NULL},
            {424,  "tion",      "t",      3,  0,  1,  NULL},
            {415,  "ou",        LAMBDA,   1, -1,  1,  NULL},
            {416,  "ism",       LAMBDA,   2, -1,  1,  NULL},
            {417,  "ate",       LAMBDA,   2, -1,  1,  NULL},
            {418,  "iti",       LAMBDA,   2, -1,  1,  NULL},
            {419,  "ous",       LAMBDA,   2, -1,  1,  NULL},
            {420,  "ive",       LAMBDA,   2, -1,  1,  NULL},
            {421,  "ize",       LAMBDA,   2, -1,  1,  NULL},
            {000,  NULL,        NULL,     0,  0,  0,  NULL},
        };

static RuleList step5a_rules[] =
        {
            {501,  "e",         LAMBDA,   0, -1,  1,  NULL},
            {502,  "e",         LAMBDA,   0, -1, -1,  RemoveAnE},
            {000,  NULL,        NULL,     0,  0,  0,  NULL},
        };

static RuleList step5b_rules[] =
```



```
        {
            {503,  "ll",        "l",      1,  0,  1,  NULL},
            {000,  NULL,        NULL,     0,  0,  0,  NULL},
        };

/*****************************************************************************
******************   Private Function Declarations   *********************

FN*************************************************************************

       WordSize( word )

   Returns: int -- a weird count of word size in adjusted syllables

   Purpose: Count syllables in a special way:  count the number
            vowel-consonant pairs in a word, disregarding initial
            consonants and final vowels.  The letter "y" counts as a
            consonant at the beginning of a word and when it has a vowel
            in front of it; otherwise (when it follows a consonant) it
            is treated as a vowel.  For example, the WordSize of "cat"
            is 1, of "any" is 1, of "amount" is 2, of "anything" is 3.

   Plan:    Run a DFA to compute the word size

   Notes:   The easiest and fastest way to compute this funny measure is
            with a finite state machine.  The initial state 0 checks
            the first letter.  If it is a vowel, then the machine changes
            to state 1, which is the "last letter was a vowel" state.
            If the first letter is a consonant or y, then it changes
            to state 2, the "last letter was a consonant state".  In
            state 1, a y is treated as a consonant (since it follows
            a vowel), but in state 2, y is treated as a vowel (since
            it follows a consonant.  The result counter is incremented
            on the transition from state 1 to state 2, since this
            transition only occurs after a vowel-consonant pair, which
            is what we are counting.
*/

static int
WordSize( char * word )
   {
   register int result;   /* WordSize of the word */
   register int state;    /* current state in machine */

   result = 0;
   state = 0;

                /* Run a DFA to compute the word size */
   while ( EOS != *word )
      {
      switch ( state )
         {
         case 0: state = (IsVowel(*word)) ? 1 : 2;
                 break;
         case 1: state = (IsVowel(*word)) ? 1 : 2;
                 if ( 2 == state ) result++;
                 break;
         case 2: state = (IsVowel(*word) || ('y' == *word)) ? 1 : 2;
                 break;
         }
      word++;
      }

   return( result );
```

```
   } /* WordSize */

/*****************************************************************************

       ContainsVowel( word )

   Returns: int -- TRUE (1) if the word parameter contains a vowel,
            FALSE (0) otherwise.

   Purpose: Some of the rewrite rules apply only to a root containing
            a vowel, where a vowel is one of "aeiou" or y with a
            consonant in front of it.

   Plan:    Obviously, under the definition of a vowel, a word contains
            a vowel iff either its first letter is one of "aeiou", or
            any of its other letters are "aeiouy".  The plan is to
            test this condition.

   Notes:   None
*/

static int
ContainsVowel( char * word )
   {

   if ( EOS == *word )
      return( FALSE );
   else
      return( IsVowel(*word) || (NULL != strpbrk(word+1,"aeiouy")) );

   } /* ContainsVowel */

/*FN*************************************************************************

       EndsWithCVC( word )

   Returns: int -- TRUE (1) if the current word ends with a
            consonant-vowel-consonant combination, and the second
            consonant is not w, x, or y, FALSE (0) otherwise.

   Purpose: Some of the rewrite rules apply only to a root with
            this characteristic.

   Plan:    Look at the last three characters.

   Notes:   None
*/

static int
EndsWithCVC( char * word )
   {
   int length;          /* for finding the last three characters */

   if ( (length = strlen(word)) < 2 )
      return( FALSE );
   else
      {
      end = word + length - 1;
      return(     (NULL == strchr("aeiouwxy",*end--))      /* consonant */
              && (NULL != strchr("aeiouy",  *end--))       /* vowel */
              && (NULL == strchr("aeiou",   *end  )) );   /* consonant */
      }

   } /* EndsWithCVC */
```

```
/*FN*************************************************************************

       AddAnE( word )

   Returns: int -- TRUE (1) if the current word meets special conditions
            for adding an e.

   Purpose: Rule 122 applies only to a root with this characteristic.

   Plan:    Check for size of 1 and a consonant-vowel-consonant ending.

   Notes:   None
*/

static int
AddAnE( char * word )
   {

   return( (1 == WordSize(word)) && EndsWithCVC(word) );

   } /* AddAnE */

/*****************************************************************************

       RemoveAnE( word )

   Returns: int -- TRUE (1) if the current word meets special conditions
            for removing an e.

   Purpose: Rule 502 applies only to a root with this characteristic.

   Plan:    Check for size of 1 and no consonant-vowel-consonant ending.

   Notes:   None
*/

static int
RemoveAnE( char * word )
   {

   return( (1 == WordSize(word)) && !EndsWithCVC(word) );

   } /* RemoveAnE */

/*****************************************************************************

       ReplaceEnd( word, rule )

   Returns: int -- the id for the rule fired, 0 is none is fired

   Purpose: Apply a set of rules to replace the suffix of a word

   Plan:    Loop through the rule set until a match meeting all conditions
            is found.  If a rule fires, return its id, otherwise return 0.
            Connditions on the length of the root are checked as part of this
            function's processing because this check is so often made.

   Notes:   This is the main routine driving the stemmer.  It goes through
            a set of suffix replacement rules looking for a match on the
            current suffix.  When it finds one, if the root of the word
            is long enough, and it meets whatever other conditions are
            required, then the suffix is replaced, and the function returns.
*/
```

```
static int
ReplaceEnd( char * word, RuleList * rule )
          /* in/out: buffer with the stemmed word */
          /* in: data structure with replacement rules */
   {
   register char *ending;   /* set to start of possible stemmed suffix */
   char tmp_ch;             /* save replaced character when testing */

   while ( 0 != rule->id )
      {
      ending = end - rule->old_offset;
      if ( word <= ending )
         if ( 0 == strcmp(ending,rule->old_end) )
            {
            tmp_ch = *ending;
            *ending = EOS;
            if ( rule->min_root_size < WordSize(word) )
               if ( !rule->condition || (*rule->condition)(word) )
                  {
                  (void)strcat( word, rule->new_end );
                  end = ending + rule->new_offset;
                  break;
                  }
            *ending = tmp_ch;
            }
      rule++;
      }

   return( rule->id );

   } /* ReplaceEnd */

/*****************************************************************************
*******************   Public Function Declarations   **********************

FN*****************************************************************************

       Stem( word )

   Returns: int -- FALSE (0) if the word contains non-alphabetic characters
            and hence is not stemmed, TRUE (1) otherwise

   Purpose: Stem a word

   Plan:    Part 1: Check to ensure the word is all alphabetic
            Part 2: Run through the Porter algorithm
            Part 3: Return an indication of successful stemming

   Notes:   This function implements the Porter stemming algorithm, with
            a few additions here and there.  See:

               Porter, M.F., "An Algorithm For Suffix Stripping,"
               Program 14 (3), July 1980, pp. 130-137.

            Porter's algorithm is an ad hoc set of rewrite rules with
            various conditions on rule firing.  The terminology of
            "step 1a" and so on, is taken directly from Porter's
            article, which unfortunately gives almost no justification
            for the various steps.  Thus this function more or less
            faithfully refects the opaque presentation in the article.
            Changes from the article amount to a few additions to the
            rewrite rules;  these are marked in the RuleList data
            structures with comments.
**/

int Stem( char * word )
   /* in/out: the word stemmed */
   {
   int rule;    /* which rule is fired in replacing an end */

            /* Part 1: Check to ensure the word is all alphabetic */
   for ( end = word; *end != EOS; end++ )
      if ( !isalpha(*end) ) return( FALSE );
      else *end = tolower( *end );
   end--;

               /*  Part 2: Run through the Porter algorithm */
   (void)ReplaceEnd( word, step1a_rules );
   rule = ReplaceEnd( word, step1b_rules );
   if ( (106 == rule) || (107 == rule) )
      (void)ReplaceEnd( word, step1b1_rules );
   (void)ReplaceEnd( word, step1c_rules );

   (void)ReplaceEnd( word, step2_rules );

   (void)ReplaceEnd( word, step3_rules );

   (void)ReplaceEnd( word, step4_rules );

   (void)ReplaceEnd( word, step5a_rules );
   (void)ReplaceEnd( word, step5b_rules );

           /* Part 3: Return an indication of successful stemming */
   return( TRUE );

   } /* Stem */

#endif
```

```
#ifndef __STEMC__
#define __STEMC__
#include "mystring.h"
//
// Class Stem for a document
//
class stem:public mystring
{
   public :
   int number;
   float weight;
   stem();
   stem(char * mot);
   virtual ~stem();
};
//
// Class Stem for a documentManager
//
class dmstem:public mystring
{
   public :
   int number;
   int lastdoc;
   int numdoc;
   dmstem();
   dmstem(char * mot);
   virtual ~dmstem();
};
//
// Function prototype
//
ostream& operator<<(ostream& s, stem& n);
ostream& operator<<(ostream& s, dmstem& n);
ostream& operator<<(ostream& s, dmstem * n);
#endif
```

```
#ifndef __STEMC__
#define __STEMC__

#include "mystring.h"

//
// Class Stem for a document
//

class stem:public mystring
{
   public :
   int number;
   float weight;

   stem();
   stem(char * mot);
   virtual ~stem();
};

//
// Class Stem for a documentManager
//

class dmstem:public mystring
{
   public :
   int number;
   int lastdoc;
   int numdoc;

   dmstem();
   dmstem(char * mot);
   virtual ~dmstem();
};

//
// Function prototype
//

ostream& operator<<(ostream& s, stem& n);
ostream& operator<<(ostream& s, dmstem& n);
ostream& operator<<(ostream& s, dmstem * n);

#endif
```

```
#ifndef __TREE__

#define __TREE__

#include "stddefs.h"

template<class TT>
class treenode
{
public :
   treenode<TT>* left;
   treenode<TT>* right;

   TT* info;

   int Add(TT* elem);
   int DeleteAllInfo();

   TT* Find(TT* elem);

   void Display();

   treenode(TT* elem);
   virtual ~treenode();
}

template<class TT> int treenode<TT>::Add(TT * elem)
{
   if (*info>*elem)
      if (!left)
      {
         left=new treenode<TT>(elem);
         return true;
      }
      else return left->Add(elem);

   if (*info<*elem)
      if (!right)
      {
         right=new treenode<TT>(elem);
         return true;
      }
      else return right->Add(elem);

   return false;
}

template<class TT> int treenode<TT>::DeleteAllInfo()
{
   int l=true;
   int r=true;

   delete info;
   info=NULL;

   if (left)
      l=left->DeleteAllInfo();
   if (right)
      r=right->DeleteAllInfo();

   return (l && r);
}

template<class TT> TT * treenode<TT>::Find(TT * elem)
{
   if (*info==*elem)
      return info;
   if (*info>*elem)
      if (left) return left->Find(elem);
   if (*info<*elem)
      if (right) return right->Find(elem);
   return NULL;
}

template<class TT> treenode<TT>::treenode(TT* elem)
{
   left=right=NULL;
   info=elem;
}

template<class TT> treenode<TT>::~treenode()
{
   delete left;
   delete right;
   left=right=NULL;
}

template<class TT> void treenode<TT>::Display()
{
   if (left)
      left->Display();
   if (right)
      right->Display();
   return;
}

template<class T>
class tree
{
 public :
   treenode<T>* root;

   int Add(T* elem);
   T* Find(T* elem);

   long num_elem;

   int DeleteAllInfo();

   void Display();

   tree(T* elem);
   tree();
   virtual ~tree();
}

template<class T> tree<T>::tree()
{
   root=NULL;
   num_elem=0;
}

template<class T> tree<T>::tree(T* elem)
{
   root=new treenode<T>(elem);
}

template<class T> tree<T>::~tree()
```

```
{
   if (root) delete root;
}

template<class T> T* tree<T>::Find(T* elem)
{
   if (root) return root->Find(elem);
   return NULL;
}

template<class T> int tree<T>::Add(T * elem)
{
   if (root) return root->Add(elem);
   root=new treenode<T>(elem);
   num_elem++;
   return (root) ? true : false;
}

template<class T> void tree<T>::Display()
{
   if (root) root->Display();
   return;
}

template<class T> int tree<T>::DeleteAllInfo()
{
   if (root) return root->DeleteAllInfo();
   return false;
}

#endif
```

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <math.h>
#include "string.h"
#include "dbirnws.h"
#include "dlink.h"
#include "tree.h"
#include "document.h"

#define INCLUSTERSIM 0.175
#define EXCLUSIONSIM 0.05
#define MAXELEMCLUSTER 5
#define INITINERTIE 0.4
#define MININERTIE 0.4
#define SPLITMAXCLUSTER 5 // MAXELEMCLUSTER/2

struct clusterelem {
   document * clusterrep;
   float inertie;
   int docincluster;
};

struct st
{
   float simil;
   int docnum;
};

float CalculateSimilarity(document * doc1, document * doc2)
{
   int i;
   int status;

   bool change;

   document * doctmp;

   stem * thestem;
   stem * astem;

   float s;

   // Look in DB for faster access...

   DB_GetSimilarity(doc1->docID, doc2->docID, s, status);

   if (status)
      return s;

   // Not in DB, calculate...

   s=0.0;
   change=false;

   if (doc1->dlstems->num_elem>doc2->dlstems->num_elem)
   {
      doctmp=doc1;
      doc1=doc2;
      doc2=doctmp;
      change=true;
   }

   doc1->dlstems->MoveAtBegin();
   for(i=1; i<=doc1->dlstems->num_elem; i++)
   {
      thestem=doc1->dlstems->GetCurrentInfo(status);

      astem=doc2->stems->Find(thestem);

      if (astem)
         s+=(thestem->weight*astem->weight);

      doc1->dlstems->MoveToNext();
   }

   if (change)
   {
      doctmp=doc1;
      doc1=doc2;
      doc2=doctmp;
   }

   DB_AddSimilarity(doc1->docID, doc2->docID, s);

   return s;
}

float EvaluateSimilarity(document * doc1, document * doc2)
{
   int i;
   int status;

   bool change;

   document * doctmp;

   stem * thestem;
   stem * astem;

   float s;

   s=0.0;
   change=false;

   if (doc1->dlstems->num_elem>doc2->dlstems->num_elem)
   {
      doctmp=doc1;
      doc1=doc2;
      doc2=doctmp;
      change=true;
   }

   doc1->dlstems->MoveAtBegin();
   for(i=1; i<=doc1->dlstems->num_elem; i++)
   {
      thestem=doc1->dlstems->GetCurrentInfo(status);

      astem=doc2->stems->Find(thestem);

      if (astem)
         s+=(thestem->weight*astem->weight);

      doc1->dlstems->MoveToNext();
   }

   if (change)
   {
```

```
      doctmp=doc1;
      doc1=doc2;
      doc2=doctmp;
   }

   return s;
}

void SimilarityTwoByTwo()
{
   int numdoc;
   int i,j;
   document * doc1;
   document * doc2;
   float sim;

   if (!DB_ConnectTo())
   {
      cout << "Database could not be found !" << endl;
      exit(0);
   }
   else cout << "Database opened." << endl;

   numdoc=DB_GetNumDocument();

   printf("%i documents.\n", numdoc);
   printf("%li documents' similarities to calculate.\n",
               (long)((((long)numdoc*(long)numdoc)/2)-numdoc));

   doc1=new document;
   doc2=new document;

   for(i=0; i<numdoc; i++)
   {
      doc1->docID=i;
      doc1->Retrieve();

      for(j=(i+1); j<numdoc; j++)
      {
         doc2->docID=j;
         if (i==j) goto skip;

         doc2->Retrieve();

         sim=CalculateSimilarity(doc1, doc2);
         printf("Similarity (%i, %i) : %f\n", i, j, sim);

         doc2->DeleteLists();

         skip : ;
      }

      DB_Commit();
      doc1->DeleteLists();
   }

   delete doc1;
   delete doc2;
}

void FusionCluster(clusterelem * cluster, document * doc)
// We have found a cluster sufficiently similar to our document
// Modify the cluster representative accordingly
{
   stem * docstem;
   stem * clusterstem;
   int status;
   int i;

   cluster->clusterrep->dlstems->MoveAtBegin();
   for(i=1; i<=cluster->clusterrep->dlstems->num_elem; i++)
   {
      clusterstem=cluster->clusterrep->dlstems->GetCurrentInfo(status);

      docstem=doc->stems->Find(clusterstem);

      if (docstem)
         // Adjust the weight of the cluster representative with term weight
             from document
         clusterstem->weight=(((cluster->docincluster*clusterstem->weight)+docst
             em->weight)/(float)(cluster->docincluster+1));
      else
      {
         // Adjust the weight of the cluster representative
         clusterstem->weight=(((cluster->docincluster*clusterstem->weight))/(flo
            at)(cluster->docincluster+1));
      }

      cluster->clusterrep->dlstems->MoveToNext();
   }

   cluster->docincluster++;

   cluster->inertie+=EvaluateSimilarity(cluster->clusterrep, doc);
}

float GetSim(float * matrix, int row, int col, int dim)
{
   return matrix[(row-1)*dim+(col-1)];
}

void SetSim(float * matrix, int row, int col, int dim, float value)
{
   matrix[(row-1)*dim+(col-1)]=value;
}

void SplitCluster(clusterelem * cluster, int clusternum, int * docincluster, int
    numdoc, int & currentcluster, dlink<clusterelem> * clusterlist)
{
   float * simi;
   int * thedocs;
   int i;
   int j;
   float thesim;
   float minsim;
   int mindoc1;
   int mindoc2;
   int docinclust;

   document * doc1;
   document * doc2;

   clusterelem * newcluster;

   docinclust=cluster->docincluster;

   // Create the similarity matrix
```

```
simi=(float *)malloc(docinclust*docinclust*sizeof(float));

// Create the document array for documents in cluster

thedocs=(int *)malloc(docinclust*sizeof(int));

// Init similarity matrix

for(i=1;i<=docinclust;i++)
  for(j=1;j<=docinclust;j++)
      SetSim(simi, i, j, docinclust, (i==j)? 1.0 : 0.0);

// Init document array for cluster

i=0;
for(j=0;j<numdoc; j++)
   if (abs(docincluster[j])==clusternum)
      thedocs[i++]=j;

// Calculate similarity matrix

doc1=new document;
doc2=new document;

for(i=1;i<=docinclust;i++)
{
   doc1->docID=thedocs[i-1];
   doc1->Retrieve();

   for(j=i+1;j<=docinclust;j++)
   {
      doc2->docID=thedocs[j-1];
      doc2->Retrieve();

      thesim=CalculateSimilarity(doc1, doc2);

      SetSim(simi, i, j, docinclust, thesim);
      SetSim(simi, j, i, docinclust, thesim);

      doc2->DeleteLists();
   }

   doc1->DeleteLists();

   DB_Commit();

}

delete doc1;
delete doc2;

// Find the lowest similarity measure for docs pair

minsim=1.0;
for(i=1; i<=docinclust; i++)
   for(j=(i+1); j<=docinclust; j++)
   {
      if (GetSim(simi, i, j, docinclust)<minsim)
      {
         mindoc1=thedocs[i-1];
         mindoc2=thedocs[j-1];
         minsim=GetSim(simi, i, j, docinclust);
      }
   }

// Organize cluster along two centroids

newcluster=new clusterelem;
newcluster->clusterrep=new document();
newcluster->clusterrep->docID=mindoc1;
newcluster->clusterrep->Retrieve();
newcluster->docincluster=1;
newcluster->inertie=INITINERTIE;

docincluster[mindoc1]=++currentcluster;

cluster->clusterrep->DeleteLists();
delete cluster->clusterrep;

cluster->clusterrep=new document();
cluster->clusterrep->docID=mindoc2;
cluster->clusterrep->Retrieve();
cluster->docincluster=1;
cluster->inertie=INITINERTIE;

for(i=0; i<docinclust; i++)
{
   if ((thedocs[i]!=mindoc1)&&(thedocs[i]!=mindoc2))
   {
      // Get the doc...

      doc1=new document();
      doc1->docID=thedocs[i];
      doc1->Retrieve();

      // In which cluster goes this doc ?

      if (EvaluateSimilarity(doc1, newcluster->clusterrep)>EvaluateSimilarity
          (doc1, cluster->clusterrep))
      {
         FusionCluster(newcluster, doc1);

         docincluster[thedocs[i]]=currentcluster;
      }
      else
      {
         FusionCluster(cluster, doc1);
      }

      doc1->DeleteLists();
      delete doc1;
   }
}

// All is done...

free(simi);
free(thedocs);

clusterlist->AddAtEnd(newcluster);

DB_Commit();
}

void Clustering()
{
   dlink<clusterelem> * clusterlist;
   int clusternumber;
   document * doc1;
```

```
document * doc2;
document * doctmp;
clusterelem * cluster;
document * cluster_rep;
int id1;
int status;
int id2;
int numdoc;
int * docincluster;
int i;
int j;
int ok;
int currentcluster;
//int first;
float maxsimilarity;
float similarity;
int maxcluster;
int initialcluster;

float s;
FILE * outfile;

if (!DB_ConnectTo())
{
   cout << "Database could not be found !" << endl;
   exit(0);
}
else cout << "Database opened." << endl;

numdoc=DB_GetNumDocument();
initialcluster=(int)ceil((double)numdoc*0.03)+1;

clusterlist=new dlink<clusterelem>;
clusternumber=0;
currentcluster=0;

docincluster=(int *)malloc((numdoc+1)*sizeof(int));

for(i=0; i<numdoc; i++)
   docincluster[i]=0;

doc1=new document;
doc2=new document;

randomize();
for(i=1; i<=initialcluster; i++)
{
   debut :
   ok=false;
   while (!ok)
   {
      id1=random(numdoc);
      (docincluster[id1]==0) ? (ok=true) : (ok=false);
   }

   doc1->docID=id1;

   doc1->Retrieve();

   maxsimilarity=0.0;
   maxcluster=0;

   clusterlist->MoveAtBegin();
   for(j=1;j<=clusterlist->num_elem;j++)
   {
      cluster=clusterlist->GetCurrentInfo(status);

      similarity=EvaluateSimilarity(cluster->clusterrep, doc1);

      if (similarity>EXCLUSIONSIM)
         maxcluster=j;

      clusterlist->MoveToNext();
   }

   if (maxcluster)
   {
      doc1->DeleteLists();
      goto debut;
   }

   cout << "Found cluster representative N: " << i << endl;

   currentcluster++;

   docincluster[id1]=-currentcluster;
   doc1->Copy(doctmp);
   doc1->DeleteLists();

   cluster=new clusterelem;
   cluster->clusterrep=doctmp;
   cluster->docincluster=1;
   cluster->inertie=INITINERTIE;

   clusterlist->AddAtEnd(cluster);
}

cout << "Initial cluster representatives found." << endl;

DB_Commit();

for(id2=0;id2<numdoc;id2++)
{
   if (docincluster[id2]==0)
   {
      doc2->docID=id2;

      doc2->Retrieve();

      maxcluster=0;
      maxsimilarity=0;

      clusterlist->MoveAtBegin();
      for(i=1; i<=clusterlist->num_elem; i++)
      {
         cluster=clusterlist->GetCurrentInfo(status);

         s=EvaluateSimilarity(cluster->clusterrep, doc2);

         if ((s>INCLUSTERSIM) && (s>maxsimilarity))
         {
            maxsimilarity=s;
            maxcluster=i;
         }

         clusterlist->MoveToNext();
      }
```

```
      if (maxcluster)
      {
         // Document goes in 'maxcluster' cluster

         docincluster[id2]=maxcluster;
         clusterlist->MoveTo(maxcluster);
         cluster=clusterlist->GetCurrentInfo(status);

         FusionCluster(cluster, doc2);

         if ((cluster->docincluster>MAXELEMCLUSTER)&&((cluster->inertie/(floa
             t)cluster->docincluster)<MININERTIE))
         {
            cout << "Splitting cluster " << maxcluster << endl;
            SplitCluster(cluster, maxcluster, docincluster, numdoc,
                currentcluster, clusterlist);
            cout << "Splitting of cluster "<< maxcluster << " ended." <<
                endl;
         }
      }
      else
      {
         // Document should go in another cluster... Create a new one...
         // Make it the cluster representative

         doc2->Copy(doctmp);

         cluster=new clusterelem;
         cluster->clusterrep=doctmp;
         cluster->docincluster=1;
         cluster->inertie=INITINERTIE;

         clusterlist->AddAtEnd(cluster);

         currentcluster++;

         docincluster[id2]=currentcluster;
      }

      doc2->DeleteLists();
   }
}

delete doc1;
delete doc2;

DB_DeleteClusters();

outfile=fopen("c:\\vincent\\dev\\clust.out", "wt+");

fprintf(outfile, "Number of clusters : %i\n", clusterlist->num_elem);

clusterlist->MoveAtBegin();
for(j=1; j<=clusterlist->num_elem;j++)
{
   cluster=clusterlist->GetCurrentInfo(status);

   fprintf(outfile, "Cluster %i : (%i,%f) -> ", j, cluster->docincluster,
       (cluster->inertie/cluster->docincluster));

   for(i=0; i<numdoc; i++)
      if (abs(docincluster[i])==j)
            fprintf(outfile, "%i ", i);
      fprintf(outfile, "\n");

      clusterlist->MoveToNext();
   }

   for(j=0; j<numdoc; j++)
      DB_AddClusterData(abs(docincluster[j]), j);

   DB_Commit();

   fclose(outfile);
}

void cluster_main()
{
   Clustering();
}
```

```
#include <stdio.h>
#include <iostream.h>
#include <string.h>
#include <stdlib.h>
#include "ocelot.h"
#include "dbirnws.h"

EXEC SQL INCLUDE SQLCA;
EXEC SQL INCLUDE SQLDA;

// Variables manipulees par le moteur SQL
EXEC SQL BEGIN DECLARE SECTION;
int nws_nwsgrp;
char nws_artid[101];
char nws_reference[101];
char nws_artfrom[255];
char nws_artsubject[255];
char nws_artsub[255];
char nws_sd[11];
char nws_st[9];
int nws_artnum;
int nws_numarticle;
int nws_authornum;
int nws_numberart;
int nws_reflevel;
int nws_refcount;
int db_cluster;
int db_docID, db_stemID;
int db_docID1, db_docID2;
char db_stem[21];
int db_occurence;
float db_weight;
float db_similarity;
char db_title[21];
int db_numstem;
int db_numdoc;
EXEC SQL END DECLARE SECTION;

void NWS_CreateDatabase(int force)
{
   EXEC SQL CREATE DATABASE IRNEWS;
   if ((sqlcode==-601)&&(!force))
   {
      printf("Database already created ! Just opening it...\n\n");

      EXEC SQL OPEN DATABASE IRNEWS;

      return;
   }

   if ((sqlcode==-601)&&(force))
   {
      printf("Database already created ! Dropping it and recreating it...\n\n");

    EXEC SQL OPEN DATABASE IRNEWS;
    DB_CheckError();

    EXEC SQL DROP DATABASE IRNEWS;
    DB_CheckError();

    EXEC SQL CREATE DATABASE IRNEWS;
    DB_CheckError();
   }

   EXEC SQL CREATE TABLE ARTICLE (ARTNUM SMALLINT, ARTFROM char(255), ARTSUBJECT
       char(255), ARTSUB char(255), ARTDATE DATE, ARTTIME TIME, ARTID char(100),
       NWSGRP SMALLINT, REFCOUNT SMALLINT);

   EXEC SQL CREATE TABLE AUTHOR(NUMBERART SMALLINT, ARTFROM char(255), AUTHORNUM
       SMALLINT);

   EXEC SQL CREATE TABLE REFERENCE(ARTID char(100), ARTREFERENCED char(100),
       LEVEL SMALLINT);

   EXEC SQL CREATE INDEX IND_ATRNUM ON ARTICLE(ARTNUM);
   EXEC SQL CREATE INDEX IND_ARTID ON ARTICLE(ARTID);
   EXEC SQL CREATE INDEX IND_NWSGRP ON ARTICLE(NWSGRP);
   EXEC SQL CREATE INDEX IND_ARTFROM ON ARTICLE(ARTFROM);

   EXEC SQL CREATE INDEX IND_AUTHFROM ON AUTHOR(ARTFROM);

   EXEC SQL CREATE INDEX IND_REFARTID ON REFERENCE(ARTID);

   EXEC SQL DECLARE CURS1 CURSOR FOR SELECT ARTNUM, ARTID, ARTFROM, ARTSUBJECT,
       REFCOUNT FROM ARTICLE WHERE NWSGRP=:nws_nwsgrp ORDER BY ARTDATE, ARTTIME;

   EXEC SQL DECLARE CURS2 CURSOR FOR SELECT ARTNUM, ARTDATE, ARTTIME, ARTSUBJECT
       FROM ARTICLE WHERE ARTFROM=:nws_artfrom ORDER BY ARTDATE DESC, ARTTIME;

   EXEC SQL DECLARE CURS3 CURSOR FOR SELECT ARTFROM, NUMBERART FROM AUTHOR ORDER
       BY ARTFROM;

   EXEC SQL DECLARE CURS4 CURSOR FOR SELECT ARTFROM,COUNT(*) FROM ARTICLE WHERE
       NWSGRP=:nws_nwsgrp GROUP BY ARTFROM ORDER BY ARTFROM;

   EXEC SQL DECLARE CURS5 CURSOR FOR SELECT DOCID FROM CLUSTERS WHERE
       CLUSTER=:db_cluster AND DOCID<>:db_docID;

   EXEC SQL DECLARE CURS6 CURSOR FOR SELECT ARTSUB, ARTID, ARTNUM FROM ARTICLE
       WHERE NWSGRP=:nws_nwsgrp ORDER BY ARTSUB, ARTDATE, ARTTIME;

   EXEC SQL CREATE TABLE DOCSTEM (DOCID SMALLINT, STEMID SMALLINT, STEM
       char(20), OCCURENCE SMALLINT, WEIGHT REAL);

   EXEC SQL CREATE TABLE DOCUMENT (DOCID SMALLINT, NUMSTEM SMALLINT, TITLE
       CHAR(20));

   EXEC SQL CREATE TABLE DOCSIMILARITY (DOCID1 SMALLINT, DOCID2 SMALLINT,
       SIMILARITY FLOAT);

   EXEC SQL CREATE TABLE CLUSTERS(CLUSTER SMALLINT, DOCID SMALLINT);

   NWS_Commit();

   DB_CheckError();
}

void NWS_CreateIndexes()
{
   EXEC SQL CREATE INDEX IND_STEM on DOCSTEM(STEM);

   EXEC SQL CREATE INDEX INDSTEM2 on DOCSTEM(DOCID, STEMID);

   EXEC SQL CREATE INDEX INDSTEM3 on DOCUMENT(DOCID);

   EXEC SQL CREATE INDEX INDSIMI on DOCSIMILARITY(DOCID1, DOCID2);

   NWS_Commit();
```

```
}

void NWS_Commit()
{
   EXEC SQL COMMIT;
   DB_CheckError();
}

int NWS_GetNumArticle()
{
   EXEC SQL SELECT COUNT(*) INTO :nws_numarticle FROM ARTICLE;

   DB_CheckError();
   return nws_numarticle;
}

void NWS_AddArticle(int artnum, char * artid, char * artfrom, char * artsubject,
    char * artsub, char * sd, char * st, int nwsgrp, int refcount)
{
   nws_artnum=artnum;
   strcpy(nws_artid, artid);
   strcpy(nws_artfrom, artfrom);
   strcpy(nws_artsubject, artsubject);
   strcpy(nws_artsub, artsub);
   strcpy(nws_sd, sd);
   strcpy(nws_st, st);
   nws_nwsgrp=nwsgrp;
   nws_refcount=refcount;

   EXEC SQL INSERT INTO ARTICLE VALUES(:nws_artnum, :nws_artfrom,
       :nws_artsubject, :nws_artsub, :nws_sd, :nws_st, :nws_artid, :nws_nwsgrp,
       :nws_refcount);

   DB_CheckError();
}

int NWS_CheckArticle(char * artid)
{
   strcpy(nws_artid, artid);

   EXEC SQL SELECT COUNT(*) INTO :nws_numarticle FROM ARTICLE WHERE
       ARTID=:nws_artid;

   DB_CheckError();

   // Return true if numarticle!=0, i.e. the article exists in DB

   return nws_numarticle;
}

// Open a cursor for all articles of a specified newsgroup

void NWS_OpenNewsGroup(int nwsgrp, int & status)
{
   nws_nwsgrp=nwsgrp;

   EXEC SQL OPEN CURS1;

   status=(sqlcode==0);
}

// Fetch the first article of a newsgroup
// Status is true if operation OK, false otherwise

void NWS_ArticleFetch(int & anum, char * artid, char * artfrom, char *
    artsubject, int & refcount ,int & status)
{
   EXEC SQL FETCH NEXT CURS1 INTO :nws_artnum, :nws_artid, :nws_artfrom,
       :nws_artsubject, :nws_refcount;

   anum=nws_artnum;
   strcpy(artid, nws_artid);
   strcpy(artfrom, nws_artfrom);
   strcpy(artsubject, nws_artsubject);
   refcount=nws_refcount;

   status=(sqlcode==0);
}

// Close the cursor opened for a newsgroup

void NWS_CloseNewsGroup(void)
{
   EXEC SQL CLOSE CURS1;

   DB_CheckError();
}

int NWS_CheckArticleFromAuthor(char * artfrom)
{
   strcpy(nws_artfrom, artfrom);

   EXEC SQL SELECT NUMBERART INTO :nws_numarticle FROM AUTHOR WHERE
       ARTFROM=:nws_artfrom;

   DB_CheckError();

   return nws_numarticle;
}

void NWS_OpenArticleFrom(char * artfrom, int & status)
{
   strcpy(nws_artfrom, artfrom);

   EXEC SQL OPEN CURS2;

   status=(sqlcode==0);
}

void NWS_ArticleFromFetch(int & anum, char * artsubject, char * artdate, char *
    arttime, int & refcount, int & status)
{
   EXEC SQL FETCH NEXT CURS2 INTO :nws_artnum, :nws_sd, :nws_st,
       :nws_artsubject, :nws_refcount;

   anum=nws_artnum;
   strcpy(artsubject, nws_artsubject);
   strcpy(artdate, nws_sd);
   strcpy(arttime, nws_st);
   refcount=nws_refcount;

   status=(sqlcode==0);
}

void NWS_CloseArticleFrom(void)
{
   EXEC SQL CLOSE CURS2;
```

```
   DB_CheckError();
}

void NWS_InsertInAuthorTable(void)
{
   EXEC SQL DELETE FROM AUTHOR;

   NWS_Commit();

   EXEC SQL INSERT INTO AUTHOR SELECT COUNT(*), ARTFROM, 0 FROM ARTICLE GROUP BY
       ARTFROM ORDER BY ARTFROM;

   NWS_Commit();
}

void NWS_AuthorInFile(int authornum, char * artfrom)
{
   nws_authornum=authornum;
   strcpy(nws_artfrom, artfrom);

   EXEC SQL UPDATE AUTHOR SET AUTHORNUM=:nws_authornum WHERE
       ARTFROM=:nws_artfrom;

   DB_CheckError();
}

void NWS_OpenAuthors(int & status)
{
   EXEC SQL OPEN CURS3;

   status=(sqlcode==0);
}

void NWS_AuthorFetch(char * artfrom, int & numberart, int & status)
{
   EXEC SQL FETCH NEXT CURS3 INTO :nws_artfrom, :nws_numberart;

   strcpy(artfrom, nws_artfrom);
   numberart=nws_numberart;

   status=(sqlcode==0);
}

void NWS_CloseAuthors()
{
   EXEC SQL CLOSE CURS3;

   DB_CheckError();
}

int NWS_AuthorInFile(char * artfrom)
{
   strcpy(nws_artfrom, artfrom);

   EXEC SQL SELECT AUTHORNUM INTO :nws_authornum FROM AUTHOR WHERE
       ARTFROM=:nws_artfrom;

   DB_CheckError();

   return nws_authornum;
}

void NWS_GetAuthorInfo(char * artfrom, int & numberart, int & authornum)
{
   strcpy(nws_artfrom, artfrom);

   EXEC SQL SELECT NUMBERART, AUTHORNUM INTO :nws_numberart, :nws_authornum FROM
       AUTHOR WHERE ARTFROM=:nws_artfrom;

   DB_CheckError();

   numberart=nws_numberart;
   authornum=nws_authornum;
}

void NWS_GetAuthorOnlyArticle(char * artfrom, int & anum)
{
   strcpy(nws_artfrom, artfrom);

   EXEC SQL SELECT ARTNUM INTO :nws_artnum FROM ARTICLE WHERE
       ARTFROM=:nws_artfrom;

   DB_CheckError();

   anum=nws_artnum;
}

void NWS_AddArticleReference(char * mess_id, char * areference, int reflevel)
{
   strcpy(nws_artid, mess_id);
   strcpy(nws_reference, areference);
   nws_reflevel=reflevel;

   EXEC SQL INSERT INTO REFERENCE VALUES(:nws_artid, :nws_reference,
       :nws_reflevel);

   DB_CheckError();
}

void NWS_GetArticleByID(char * artid, char * artsubject, char * artfrom, char *
    artdate, char * arttime, int & refcount, int & anum, int & present)
{
   strcpy(nws_artid, artid);

   EXEC SQL SELECT ARTSUBJECT, ARTFROM, ARTDATE, ARTTIME, REFCOUNT, ARTNUM INTO
       :nws_artsubject, :nws_artfrom, :nws_sd, :nws_st, :nws_refcount,
       :nws_artnum FROM ARTICLE WHERE ARTID=:nws_artid;

   strcpy(artsubject, nws_artsubject);
   strcpy(artfrom, nws_artfrom);
   strcpy(artdate, nws_sd);
   strcpy(arttime, nws_st);
   refcount=nws_refcount;
   anum=nws_artnum;

   present=(sqlcode==0);
}

void NWS_GetReferencedArticle(char * artid, int level, char * referenced)
{
   strcpy(nws_artid, artid);
   nws_reflevel=level;

   EXEC SQL SELECT ARTREFERENCED INTO :nws_reference FROM REFERENCE WHERE
       (ARTID=:nws_artid) AND (LEVEL=:nws_reflevel);

   DB_CheckError();
```

```
   strcpy(referenced, nws_reference);
}

void NWS_OpenAuthorsNewsgroup(int nwsgrp, int & status)
{
   nws_nwsgrp=nwsgrp;

   EXEC SQL OPEN CURS4;

   status=(sqlcode==0);
}

void NWS_AuthorNewsgroupFetch(char * artfrom, int & numberart, int & status)
{
   EXEC SQL FETCH CURS4 INTO :nws_artfrom, :nws_numarticle;

   strcpy(artfrom, nws_artfrom);
   numberart=nws_numarticle;

   status=(sqlcode==0);
}

void NWS_CloseAuthorsNewsgroup(void)
{
   EXEC SQL CLOSE CURS4;
}

int DB_ConnectTo()
{
   EXEC SQL OPEN DATABASE IRNEWS;

   if (sqlcode!=0)
      return 0;
   else return 1;
}

void DB_Commit()
{
   EXEC SQL COMMIT;
   DB_CheckError();
}

void DB_CheckError()
{
   int tmp;

   if (sqlcode!=0)
   {
      printf("%i, %s\n", sqlcode, sqlmess_);
      scanf("%i", &tmp);
   }
}

void DB_AddStem(int pdocID, int pstemID, char pstem[21], int poccurence, float
    pweight)
{
   db_docID=pdocID;
   db_stemID=pstemID;
   strcpy(db_stem, pstem);
   db_occurence=poccurence;
   db_weight=pweight;

   EXEC SQL INSERT INTO DOCSTEM VALUES(:db_docID, :db_stemID, :db_stem,
       :db_occurence, :db_weight);
   DB_CheckError();
}

void DB_GetStem(int pdocID, int pstemID, char pstem[21], int& poccurence, float
    & pweight)
{
   db_docID=pdocID;
   db_stemID=pstemID;
   EXEC SQL SELECT STEM, OCCURENCE, WEIGHT INTO :db_stem, :db_occurence,
       :db_weight FROM DOCSTEM WHERE (DOCID=:db_docID) AND (STEMID=:db_stemID);
   DB_CheckError();
   strcpy(pstem, db_stem);
   poccurence=db_occurence;
   pweight=db_weight;
}

void DB_UpdateStem(int pdocID, int pstemID, int poccurence, float pweight)
{
   db_docID=pdocID;
   db_stemID=pstemID;
   db_occurence=poccurence;
   db_weight=pweight;
   EXEC SQL UPDATE DOCSTEM SET OCCURENCE=:db_occurence, WEIGHT=:db_weight WHERE
       (DOCID=:db_docID) AND (STEMID=:db_stemID);
   DB_CheckError();
}

void DB_AddDocument(int pdocID, int pnumstem, char * title)
{
   db_docID=pdocID;
   db_numstem=pnumstem;
   strcpy(db_title, title);
   EXEC SQL INSERT INTO DOCUMENT VALUES(:db_docID, :db_numstem, :db_title);
   DB_CheckError();
}

void DB_GetDocument(int pdocID, int & pnumstem, char * title)
{
   db_docID=pdocID;
   EXEC SQL SELECT NUMSTEM, TITLE INTO :db_numstem, :db_title FROM DOCUMENT
       WHERE DOCID=:db_docID;
   DB_CheckError();
   pnumstem=db_numstem;
   strcpy(title, db_title);
}

int  DB_GetNumDocument()
{
   EXEC SQL SELECT COUNT(*) INTO :db_numdoc FROM DOCUMENT;
   DB_CheckError();
   return db_numdoc;
}

void DB_AddSimilarity(int pdocID1, int pdocID2, float psimilarity)
{
   int tmp;

   if (pdocID1>pdocID2)
   {
      tmp=pdocID2;
      pdocID2=pdocID1;
      pdocID1=tmp;
   }
```

```
    db_docID1=pdocID1;
    db_docID2=pdocID2;
    db_similarity=psimilarity;

    EXEC SQL INSERT INTO DOCSIMILARITY VALUES(:db_docID1, :db_docID2,
        :db_similarity);
    DB_CheckError();
}

void DB_GetSimilarity(int pdocID1, int pdocID2, float & psimilarity, int &
    status)
{
    int tmp;

    if (pdocID1>pdocID2)
    {
        tmp=pdocID2;
        pdocID2=pdocID1;
        pdocID1=tmp;
    }

    db_docID1=pdocID1;
    db_docID2=pdocID2;

    EXEC SQL SELECT DOCSIMILARITY INTO :db_similarity FROM DOCSIMILARITY WHERE
        (DOCID1=:db_docID1) and (DOCID2=:db_docID2);

    psimilarity=db_similarity;

    status=(sqlcode==0);
}

void DB_DeleteClusters()
{
    EXEC SQL DELETE FROM CLUSTERS;

    DB_Commit();
}

void DB_AddClusterData(int cluster, int docID)
{
    db_cluster=cluster;
    db_docID1=docID;

    EXEC SQL INSERT INTO CLUSTERS VALUES(:db_cluster, :db_docID1);

    DB_CheckError();
}

void NWS_GetArticleByNum(int artnum, char * artsubject, char * artfrom, int &
    present)
{
    nws_artnum=artnum;

    EXEC SQL SELECT ARTSUBJECT, ARTFROM INTO :nws_artsubject, :nws_artfrom FROM
        ARTICLE WHERE ARTNUM=:nws_artnum;

    strcpy(artsubject, nws_artsubject);
    strcpy(artfrom, nws_artfrom);

    present=(sqlcode==0);
}

int NWS_GetClusterCountButItself(int cluster, int docid)
{
    db_cluster=cluster;
    db_docID=docid;

    EXEC SQL SELECT COUNT(*) INTO :nws_numarticle FROM CLUSTERS WHERE
        CLUSTER=:db_cluster AND DOCID<>:db_docID;

    return nws_numarticle;
}

void NWS_OpenNewsByCluster(int cluster, int docid, int & status)
{
    db_cluster=cluster;
    db_docID=docid;

    EXEC SQL OPEN CURS5;

    status=(sqlcode==0);
}

void NWS_FetchArticleByCluster(int & artnum, int & status)
{
    EXEC SQL FETCH CURS5 INTO :db_docID1;

    artnum=db_docID1;

    status=(sqlcode==0);
}

void NWS_CloseNewsByCluster(void)
{
    EXEC SQL CLOSE CURS5;

    DB_CheckError();
}

int NWS_GetClusterForArticle(int docid)
{
    db_docID=docid;

    EXEC SQL SELECT CLUSTER INTO :db_cluster FROM CLUSTERS WHERE DOCID=:db_docID;

    DB_CheckError();

    return db_cluster;
}

void NWS_GetDocIDForArticle(int i, int & artdocid)
{
    char name[9];

    sprintf(name, "%08i", i);
    strcpy(db_title, name);

    EXEC SQL SELECT DOCID INTO :db_docID FROM DOCUMENT WHERE TITLE=:db_title;

    DB_CheckError();

    artdocid=db_docID;
}

void NWS_GetArticleByDocID(int i, int & artnum)
{
    db_docID=i;
```

```
    EXEC SQL SELECT TITLE INTO :db_title FROM DOCUMENT WHERE DOCID=:db_docID;

    DB_CheckError();

    artnum=atoi(db_title);
}

void NWS_OpenBySubject(int nwsgrp, int & status)
{
    nws_nwsgrp=nwsgrp;

    EXEC SQL OPEN CURS6;

    status=(sqlcode==0);
}

void NWS_FetchBySubject(char * artsubject, char * artid, int & artnum, int &
     status)
{
    EXEC SQL FETCH CURS6 INTO :nws_artsubject, :nws_artid, :nws_artnum;

    strcpy(artsubject, nws_artsubject);
    strcpy(artid, nws_artid);
    artnum=nws_artnum;

    status=(sqlcode==0);
}

void NWS_CloseBySubject()
{
    EXEC SQL CLOSE CURS6;

    DB_CheckError();
}
```

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "document.h"
#include "stddefs.h"
#include "stdfunc.h"
#include "stemc.h"
#include "stem.h"
#include "dbirnws.h"

//
// Constructor
//

document::document()
{
   managedBy=NULL;
   fichier=(char *)malloc(FNLENGTH);
   stems=new tree<stem>;
   dlstems=new dlink<stem>;
   fileh=NULL;
   strcpy(titre, "");
}

void document::Copy(document * & doccopied)
{
   int numelem;
   int i;
   int status;
   stem * thestem;
   stem * anotherstem;

   doccopied=new document();

   doccopied->docID=docID;
   numelem=dlstems->num_elem;

   dlstems->MoveAtBegin();
   for(i=1;i<=numelem;i++)
   {
      thestem=dlstems->GetCurrentInfo(status);

      anotherstem=new stem(thestem->string);
      anotherstem->weight=thestem->weight;
      anotherstem->number=thestem->number;

      doccopied->dlstems->AddAtEnd(anotherstem);
      doccopied->stems->Add(anotherstem);

      dlstems->MoveToNext();
   }
}

document::document(char * fich, documentManager * manager, char * ptitre)
{
   managedBy=manager;
   docID=manager->lastID++;
   fichier=(char *)malloc(FNLENGTH);
   strcpy(titre,ptitre);
   stems=new tree<stem>;
   dlstems=new dlink<stem>;
   strcpy(fichier, fich);
   fileh=NULL;
}

void document::Store()
{
   int i;
   int status;
   stem * thestem;

   cout << dlstems->num_elem << " stems to store " << endl;

   DB_AddDocument(docID, dlstems->num_elem, titre);

   dlstems->MoveAtBegin();
   for(i=1;i<=dlstems->num_elem;i++)
   {
      thestem=dlstems->GetCurrentInfo(status);

      DB_AddStem(docID, i, thestem->string, thestem->number,
                 thestem->weight);

      dlstems->MoveToNext();
   }

   DB_Commit();

   DeleteLists();

   cout << "Total num of stems " << managedBy->dlstems->num_elem << endl;

}

void document::Update()
{
   stem * thestem;
   int i;
   int status;

   cout << dlstems->num_elem << " stems to update " << endl;

   dlstems->MoveAtBegin();
   for(i=1;i<=dlstems->num_elem;i++)
   {

      thestem=dlstems->GetCurrentInfo(status);

      DB_UpdateStem(docID, i, thestem->number, thestem->weight);

      dlstems->MoveToNext();
   }

   DB_Commit();

   DeleteLists();

   cout << "Document " << docID << " updated " << endl;

}

void document::Retrieve()
{
   stem * thestem;
   int numofstem;
   int i;
```

```
   char tStem[MAXSTEMLENGTH+1];
   int tOccurence;
   float tWeight;

   if (!stems) stems=new tree<stem>;
   if (!dlstems) dlstems=new dlink<stem>;

   DB_GetDocument(docID, numofstem, titre);

   for(i=1; i<=numofstem; i++)
   {
      DB_GetStem(docID, i, tStem, tOccurence, tWeight);

      thestem=new stem(tStem);

      thestem->number=tOccurence;
      thestem->weight=tWeight;

      dlstems->AddAtEnd(thestem);
      stems->Add(thestem);
   }

   cout << "Retrieved document " << docID << ',' << dlstems->num_elem << "
       stems" <<endl;

}

void document::DeleteLists()
{
   stems->DeleteAllInfo();

   delete stems;
   delete dlstems;

   dlstems=NULL;
   stems=NULL;
}

char * document::FirstPass()
{
   fileh=fopen(fichier, "rb"); // Open text file in binary mode for read only
       access

   if (ferror(fileh))  /* test for an error on the stream */
   {
      /* reset the error and EOF indicators */
      clearerr(fileh);
      return NULL;
   }

   char * ligne;
   char * mot;
   int longueur;
   dmstem * racinedm;
   dmstem * intreedm;
   long stemlen;

   ligne=(char *)malloc(LINELENGTH);
   mot=(char *)malloc(MOTLENGTH);

   while (!ReadLine(fileh, ligne, longueur))
   {
       char * lastpos=ligne;
       while (GetNextWord(mot, lastpos))
       {
         if (managedBy->stopwords)
         {
            mystring * lookup=new mystring(mot);
            if (managedBy->stopwords->Find(lookup))
            {
               delete lookup;
               goto skip;
            }
            delete lookup;
         }
         Stem(mot);
         stemlen=strlen(mot);
         if ((stemlen>=MINSTEMLENGTH)&&(stemlen<=MAXSTEMLENGTH)&&(!AllNum(mot)))
         {
            racinedm=new dmstem(mot);

            // Set the number of occurences in all collection
            intreedm=managedBy->stems->Find(racinedm);
            if (!intreedm)
            {
               managedBy->stems->Add(racinedm);
               managedBy->dlstems->AddAtEnd(racinedm);
            }
               else intreedm->number++;

            intreedm=managedBy->stems->Find(racinedm);

            if (intreedm->lastdoc<docID)
            {
               intreedm->lastdoc=docID;
               intreedm->numdoc++;
            }
         }

         skip : ;
       }
   }

   free(ligne);
   free(mot);

   fclose(fileh);

   return NULL;
}

char * document::SecondPass()
{
   fileh=fopen(fichier, "rb"); // Open text file in binary mode for read only
       access

   if (ferror(fileh))  /* test for an error on the stream */
   {
      /* reset the error and EOF indicators */
      clearerr(fileh);
      return NULL;
   }

   char * ligne;
   char * mot;
   int longueur;
   stem * racine;
```

```
    stem * intree;
    long stemlen;

    ligne=(char *)malloc(LINELENGTH);
    mot=(char *)malloc(MOTLENGTH);

    while (!ReadLine(fileh, ligne, longueur))
    {
        char * lastpos=ligne;

        while (GetNextWord(mot, lastpos))
        {
          if (managedBy->stopwords)
          {
             mystring * lookup=new mystring(mot);
             if (managedBy->stopwords->Find(lookup))
             {
                delete lookup;
                goto skip;
             }
             delete lookup;
          }
          Stem(mot);
          if (managedBy->stems)
          {
             dmstem * lookup=new dmstem(mot);
             if (managedBy->stems->Find(lookup))
                delete lookup;
             else
                {
                 delete lookup;
                 goto skip;
                }
          }

          stemlen=strlen(mot);
          if ((stemlen>=MINSTEMLENGTH)&&(stemlen<=MAXSTEMLENGTH)&&(!AllNum(mot)))
          {
             racine=new stem(mot);

             // Set the number of occurences in document
             intree=stems->Find(racine);
             if (!intree)
             {
                stems->Add(racine);
                dlstems->AddAtEnd(racine);
             }
             else intree->number++;
          }

          skip : ;
        }
    }

    free(ligne);
    free(mot);

    fclose(fileh);

    return NULL;
}

void document::CalculateWeight()
{
    float denom;
    float logmul;
    float logmul2;
    int j;
    int status;
    stem * docstem;
    dmstem * intreestem;
    dmstem * dmastem;
    float sum;

    // Calculte the denominator for a doc

    dlstems->MoveAtBegin();
    sum=0.0;
    for(j=1;j<=dlstems->num_elem;j++)
    {
       docstem=dlstems->GetCurrentInfo(status);

       intreestem=new dmstem(docstem->string);
       dmastem=managedBy->stems->Find(intreestem);
       delete intreestem;

       logmul=log((float)managedBy->manageddocs->num_elem/(float)dmastem->numdoc)
           ;
       logmul2=logmul*logmul;
       sum+=(float)((docstem->number*docstem->number)*logmul2);

       dlstems->MoveToNext();
    }
    denom=sqrt(sum);

    // OK, we get it !
    // Now, for each term in the doc, calculate its weight

    dlstems->MoveAtBegin();
    for(j=1;j<=dlstems->num_elem;j++)
    {
       docstem=dlstems->GetCurrentInfo(status);

       intreestem=new dmstem(docstem->string);
       dmastem=managedBy->stems->Find(intreestem);
       delete intreestem;

       logmul=(float)log((double)managedBy->manageddocs->num_elem/(double)dmastem
           ->numdoc);
       docstem->weight=(float)docstem->number*logmul/denom;

       dlstems->MoveToNext();
    }
}

//
// Constructor
//

document::~document()
{
    free(fichier);

    if (stems)
       stems->DeleteAllInfo();

    delete stems;
    delete dlstems;
```

```
}

void document::Display()
{
   if (stems) stems->Display();
}

//
// Class documentManager
//

documentManager::documentManager()
{
   manageddocs=new dlink<document>();
   stems=new tree<dmstem>;
   dlstems=new dlink<dmstem>;
   stopwords=NULL;
   lastID=0;
}

documentManager::~documentManager()
{
   if (manageddocs) manageddocs->DeleteAllInfo();
   if (stopwords) stopwords->DeleteAllInfo();
   if (stems) stems->DeleteAllInfo();

   delete manageddocs;
   delete stopwords;
   delete stems;
   delete dlstems;
}

void documentManager::Add(document* doc)
{
   doc->managedBy=this;
   if (!manageddocs)
      manageddocs=new dlink<document>;
   manageddocs->AddAtEnd(doc);
}

void documentManager::DeleteDocuments()
{
   int i,status;
   int nb=manageddocs->num_elem;
   document * tmpdoc;

   manageddocs->MoveAtBegin();
   for(i=1;i<=nb;i++)
   {
      tmpdoc=manageddocs->GetCurrentInfo(status);
      delete tmpdoc;
      manageddocs->DeleteAtBegin();
   }

   delete manageddocs;
   manageddocs=NULL;
}

void documentManager::FirstPass()
{
   int i, status;
   int nb;
   document * tmpdoc;

   manageddocs->MoveAtBegin();
   nb=manageddocs->num_elem;
   for(i=1;i<=nb;i++)
   {
      tmpdoc=manageddocs->GetCurrentInfo(status);
      cout << "First pass on document " << tmpdoc->fichier << endl;
      tmpdoc->FirstPass();
      manageddocs->MoveToNext();
   }
}

void documentManager::SecondPass()
{
   int i, status;
   document * tmpdoc;

   cout << manageddocs->num_elem << " documents should be processed." << endl;

   manageddocs->MoveAtBegin();
   i=1;
   while(i<=manageddocs->num_elem)
   {
      tmpdoc=manageddocs->GetCurrentInfo(status);
      cout << "Second pass on document " << tmpdoc->fichier << endl;
      tmpdoc->SecondPass();
      tmpdoc->CalculateWeight();
      tmpdoc->Store();
      manageddocs->MoveToNext();
      i++;
   }
}

void documentManager::EleminateStems()
{
   int i;
   int status;
   dmstem * thestem;

   cout << "Eliminating stems." << endl;
   cout << "Before elimination : " << dlstems->num_elem << endl;

   if (stems) delete stems;
   i=1;
   dlstems->MoveAtBegin();
   while (i<=dlstems->num_elem)
   {
      thestem=dlstems->GetCurrentInfo(status);
      if ((thestem->number==1)||(thestem->numdoc==1))
      {
         dlstems->DeleteCurrentInfo();
         dlstems->DeleteCurrent();
      }
      else {
            dlstems->MoveToNext();
            i++;
           }
   }

   stems=new tree<dmstem>;
   dlstems->MoveAtBegin();
   for(i=1; i<=dlstems->num_elem;i++)
   {
      stems->Add(dlstems->GetCurrentInfo(status));
      dlstems->MoveToNext();
```

```
    }

    cout << "After elimination : " << dlstems->num_elem << endl;

}

void documentManager::CalculateWeight()
{
    float denom;
    float logmul;
    float logmul2;
    int i,j;
    int status;
    document * tmpdoc;
    stem * docstem;
    dmstem * intreestem;
    dmstem * dmastem;
    float sum;

    // Calculate the weight of all terms in all docs

    // Iterate through all the docs
    manageddocs->MoveAtBegin();
    for(i=1;i<=manageddocs->num_elem;i++)
    {
        tmpdoc=manageddocs->GetCurrentInfo(status);

        tmpdoc->Retrieve();

        cout << "Calculating weight for doc " << tmpdoc->fichier << endl;

        // Calculte the denominator for a doc

        tmpdoc->dlstems->MoveAtBegin();
        sum=0.0;
        for(j=1;j<=tmpdoc->dlstems->num_elem;j++)
        {
            docstem=tmpdoc->dlstems->GetCurrentInfo(status);

            intreestem=new dmstem(docstem->string);
            dmastem=stems->Find(intreestem);
            delete intreestem;

            logmul=log((float)manageddocs->num_elem/(float)dmastem->numdoc);
            logmul2=logmul*logmul;

            sum+=(float)((docstem->number*docstem->number)*logmul2);

            tmpdoc->dlstems->MoveToNext();
        }
        denom=sqrt(sum);

        // OK, we get it !
        // Now, for each term in the doc, calculate its weight

        tmpdoc->dlstems->MoveAtBegin();
        for(j=1;j<=tmpdoc->dlstems->num_elem;j++)
        {
            docstem=tmpdoc->dlstems->GetCurrentInfo(status);

            intreestem=new dmstem(docstem->string);
            dmastem=stems->Find(intreestem);
            delete intreestem;

            logmul=(float)log((double)manageddocs->num_elem/(double)dmastem->numdoc
                );
            docstem->weight=(float)docstem->number*logmul/denom;

            tmpdoc->dlstems->MoveToNext();
        }

        // All the terms have been assigned a weight.
        // Move to next doc

        tmpdoc->Update();

        manageddocs->MoveToNext();
    }
}

void documentManager::Display()
{
    cout << "Displaying stems of doc manager" << endl;
    if (stems) stems->Display();
    cout << "End of display stems of doc manager" << endl;
}

void documentManager::ReadStopWord(char * fich)
{
    if (stopwords)
    {
        stopwords->DeleteAllInfo();
        delete stopwords;
    }

    stopwords=new tree<mystring>();

    FILE * fileh;
    char * mot;
    int longueur;
    mystring * tmp;

    mot=(char *)malloc(40);
    fileh=fopen(fich, "rb");

    while (!ReadLine(fileh, mot,longueur))
    {
        tmp=new mystring(mot);
        stopwords->Add(tmp);
    }
    free(mot);

    fclose(fileh);
}
```

```
#include "irnews.h"
#include "splitnws.h"
#include "ir.h"
#include "clusters.h"
#include "dlink.h"
#include "mystring.h"

#define DROPDATABASE 1
#define DONOTDROPDATABASE 0

#define INIFILE "c:\\vincent\\dev\\irnews.ini"

#define ALTHYP 1
#define COMPTEXT 2
#define LINANN 3
#define SECMISC 4
#define SECUNIX 5
#define SYSINTEL 6
#define MEMOIRE 7

dlink<mystring> * groups;
dlink<mystring> * groupsdesc;

int DatabaseStartup(void)
{
   char choice;

   init : printf("IR-NEWS - Automatic Construction of a Hypertext system from
       news articles\n\n");
   printf("Would you like to keep the existing database or start from scratch ?
       (E/S)");
   choice=getchar();getchar();

   if ((choice!='E')&&(choice!='S')) goto init;

   return (choice=='E') ? DONOTDROPDATABASE : DROPDATABASE;
}

int Menu(void)
{
   int choice;

   init : printf("IR-NEWS - Automatic Construction of a Hypertext system from
       news articles\n\n");
   printf("1 - Split news files\n");
   printf("2 - Calculate stems weight\n");
   printf("3 - Generate clusters\n");
   printf("4 - Generate hypertext (HTML)\n");
   printf("5 - All four in one go\n");
   printf("6 - Quit\n");

   choice=getchar();

   if ((choice<'1')||(choice>'6')) goto init;

   return (choice-'0');
}

void ReadIniFile()
{
   FILE * inifile;
   char aline[1025];

   inifile=fopen(INIFILE, "rt");
   while (fgets(aline, 1024, inifile))
   {
      groups->AddAtEnd(new mystring(strtok(aline, ",")));
      groupsdesc->AddAtEnd(new mystring(strtok(NULL ,"\n")));
   }

   fclose(inifile);
}

void main()
{
   mystring * g;
   mystring * gd;
   int i;
   int status;

   NWS_CreateDatabase(DatabaseStartup());

   groups=new dlink<mystring>;
   groupsdesc=new dlink<mystring>;

   ReadIniFile();

   while (true)
   {
      switch(Menu()) {
      case 1 : {

               groups->MoveAtBegin();
               groupsdesc->MoveAtBegin();

               i=1;
               while (i<=groups->num_elem)
               {
                  g=groups->GetCurrentInfo(status);
                  gd=groups->GetCurrentInfo(status);

                  SplitFile(g->string, i);

                  groups->MoveToNext();
                  groupsdesc->MoveToNext();

                  i++;
               }
               };break;

      case 2 : {
         ir_main();
               };break;
      case 3 : {
         Clustering();
               };break;
      case 4 : {
         OrganizeAuthors();

               groups->MoveAtBegin();
               groupsdesc->MoveAtBegin();

               i=1;
               while (i<=groups->num_elem)
               {
                  g=groups->GetCurrentInfo(status);
                  gd=groups->GetCurrentInfo(status);
```

```
            OrganizeAuthorsNewsgroup(i, gd->string);

            groups->MoveToNext();
            groupsdesc->MoveToNext();

            i++;
         }

         groups->MoveAtBegin();
         groupsdesc->MoveAtBegin();

         i=1;
         while (i<=groups->num_elem)
         {
            g=groups->GetCurrentInfo(status);
            gd=groups->GetCurrentInfo(status);

            OrganizeNewsGroup(i, gd->string);

            groups->MoveToNext();
            groupsdesc->MoveToNext();

            i++;
         }

         groups->MoveAtBegin();
         groupsdesc->MoveAtBegin();

         i=1;
         while (i<=groups->num_elem)
         {
            g=groups->GetCurrentInfo(status);
            gd=groups->GetCurrentInfo(status);

            OrganizeNewsgroupBySubject(i, gd->string);

            groups->MoveToNext();
            groupsdesc->MoveToNext();

            i++;
         }

         OrganizeNewsgroupUsingClusters();
         };break;
case 5 : {
         groups->MoveAtBegin();
         groupsdesc->MoveAtBegin();

         i=1;
         while (i<=groups->num_elem)
         {
            g=groups->GetCurrentInfo(status);
            gd=groups->GetCurrentInfo(status);

            SplitFile(g->string, i);

            groups->MoveToNext();
            groupsdesc->MoveToNext();

            i++;
         }

   ir_main();
         Clustering();

         OrganizeAuthors();

               groups->MoveAtBegin();
               groupsdesc->MoveAtBegin();

               i=1;
               while (i<=groups->num_elem)
               {
                  g=groups->GetCurrentInfo(status);
                  gd=groups->GetCurrentInfo(status);

                  OrganizeAuthorsNewsgroup(i, gd->string);

                  groups->MoveToNext();
                  groupsdesc->MoveToNext();

                  i++;
               }

               groups->MoveAtBegin();
               groupsdesc->MoveAtBegin();

               i=1;
               while (i<=groups->num_elem)
               {
                  g=groups->GetCurrentInfo(status);
                  gd=groups->GetCurrentInfo(status);

                  OrganizeNewsGroup(i, gd->string);

                  groups->MoveToNext();
                  groupsdesc->MoveToNext();

                  i++;
               }

               groups->MoveAtBegin();
               groupsdesc->MoveAtBegin();

               i=1;
               while (i<=groups->num_elem)
               {
                  g=groups->GetCurrentInfo(status);
                  gd=groups->GetCurrentInfo(status);

                  OrganizeNewsgroupBySubject(i, gd->string);

                  groups->MoveToNext();
                  groupsdesc->MoveToNext();

                  i++;
               }

               OrganizeNewsgroupUsingClusters();
               };break;
      case 6 : exit(0);
   }
  }
}
```

```
#include <string.h>

#include <iostreams.h>

#include <stdlib.h>

#include "stddefs.h"

#include "mystring.h"

int mystring::operator<(mystring& right)
{
   return (strcmp(string, right.string)<0) ? true : false;
}

int mystring::operator>(mystring& right)

{

   return (strcmp(string, right.string)>0) ? true : false;

}

int mystring::operator==(mystring& right)

{

   return (strcmp(string, right.string)==0) ? true : false;

}

mystring::mystring()

{

   string=NULL;

}

mystring::mystring(char * is)

{

   string=(char *)malloc(strlen(is)+1);

   strcpy(string, is);

}

mystring::~mystring()

{

   if (string) free(string);

}

// Function definition

ostream& operator<<(ostream& s, mystring& n)

{
   if (n.string)
      return s << (char *) n.string << endl;
      else return s << "Empty string";
};

ostream& operator<<(ostream& s, mystring * n)
{
   if (n->string)
      return s << (char *) n->string;
      else return s << "Empty string";
}
```

```
#include "irnews.h"

#define ALTHYP 1
#define COMPTEXT 2
#define LINANN 3
#define SECMISC 4
#define SECUNIX 5
#define SYSINTEL 6

#define DROPDATABASE 1
#define DONOTDROPDATABASE 0

#define OUTPUTNEWS "d:\\vincent\\dev\\nws\\out\\%08i.nws"
#define OUTPUTHTML "d:\\vincent\\dev\\nws\\html\\%08i.htm"
#define OUTPUTHTM2 "d:\\vincent\\dev\\nws\\html\\%08i.ht2"
#define OUTPUTAUTH "d:\\vincent\\dev\\nws\\html\\aut%05i.htm"
#define OUTPUTSUBJECT "d:\\vincent\\dev\\nws\\html\\sub%05i.htm"
#define OUTPUTINDEXAUTHOR "d:\\vincent\\dev\\nws\\html\\indauth.htm"
#define OUTPUTINDEXNEWS "d:\\vincent\\dev\\nws\\html\\ind%05i.htm"

#define ARTICLEDELIMITER "========\n"

// Definition des variables locales

int artnum;
FILE * datefile;

// #define __HOME__

#ifdef __HOME__

char AARTICLE[100]="\"file:///d:\\vincent\\dev\\nws\\html\\%08i.htm\"";
char ASUBJIND[100]="\"file:///d:\\vincent\\dev\\nws\\html\\sub%05i.htm\"";
char AAUTHORI[100]="\"file:///d:\\vincent\\dev\\nws\\html\\aut%05i.htm\"";
char AAAUTIND[100]="\"file:///d:\\vincent\\dev\\nws\\html\\indauth.htm\"";
char AARTIIND[100]="\"file:///d:\\vincent\\dev\\nws\\html\\ind%05i.htm\"";
char AAUTHIND[100]="\"file:///d:\\vincent\\dev\\nws\\html\\nws%05i.htm\"";
char AHYPSYS[100]="\"file:///c:\\vincent\\dev\\index.htm\"";

#else

char AARTICLE[100]="\"http://www.info.fundp.ac.be/~vdh/news/%08i.html\"";
char ASUBJIND[100]="\"http://www.info.fundp.ac.be/~vdh/news/sub%05i.html\"";
char AAUTHORI[100]="\"http://www.info.fundp.ac.be/~vdh/news/aut%05i.html\"";
char AAAUTIND[100]="\"http://www.info.fundp.ac.be/~vdh/news/indauth.html\"";
char AARTIIND[100]="\"http://www.info.fundp.ac.be/~vdh/news/ind%05i.html\"";
char AAUTHIND[100]="\"http://www.info.fundp.ac.be/~vdh/news/nws%05i.html\"";
char AHYPSYS[100]="\"http://www.info.fundp.ac.be/~vdh/news/index.html\"";

#endif

// Removes char at the beginning of a string...

char * Skip(char * aline, int num)
{
   char * tmpline;
   char * orig;
   int i;

   tmpline=(char *)malloc(strlen(aline)+1);
   orig=tmpline;

   strcpy(tmpline, aline);
   for(i=1; i<=num; i++)
      tmpline++;

   strcpy(aline, tmpline);

   free(orig);

   return aline;
}

// Replace avery CR by a NULL character

void CRToNull(char * aline)
{
   unsigned char * tmpline;

   tmpline=aline;

   while (*tmpline)
   {
      *tmpline=(*tmpline=='\n') ? 0 : *tmpline;
      tmpline++;
   }
}

// Replace all chars in string to comply with HTML
// < replace by (
// > replace by )
// & replace by @

void TextToHTML(char * aline)
{
   char * achar;

   achar=aline;
   while (*achar)
   {
      switch (*achar)
      {
         case '<' : *achar='(';break;
         case '>' : *achar=')';break;
         case '&' : *achar='@';break;
      };
      achar++;
   }
}

void DateToDateTime(char * sdate, char * date, char * time)
{
   char * calendar[]={"JAN", "FEB", "MAR", "APR", "MAY", "JUN", "JUL", "AUG",
       "SEP", "OCT", "NOV", "DEC"};

   char tmpdate[1025];

   char * day;
   char * month;
   char * year;

   int i;
   int found;

   // Work on a copy of date string
   strcpy(tmpdate, sdate);
```

```
   day=strtok(tmpdate, " ");

   month=strtok(NULL, " ");

   i=0;
   found=false;
   while ((!found) && (i<12))
   {
      if (strcmp(calendar[i], month)==0)
         found=true;
      else i++;
   }
   i++;

   if (!found)
      exit(1);

   year=strtok(NULL, " ");

   if (atoi(year)<1900)
      sprintf(year, "%i", atoi(year)+1900);

   sprintf(date, "%s-%02i-%02i", year, i, atoi(day));

   strcpy(time, strtok(NULL, " "));
}

void FixSubject(char * subject, char * & fixs)
{
   char * thesub;
   char * origsub;

   if (fixs) free(fixs);

   thesub=(char *)malloc(strlen(subject)+1);
   origsub=thesub;

   strcpy(thesub, subject);

   doit :

   if ((strncmp(thesub, "RE:", 3)==0) || (strncmp(thesub, "RE :", 4)==0))
   {
      thesub=strtok(thesub, ":");
      thesub=strtok(NULL, "\0");
      while (*thesub==' ')
         thesub++;

      strcpy(origsub, thesub);
      thesub=origsub;

      goto doit;
   }

   fixs=(char *)malloc(strlen(thesub)+1);

   strcpy(fixs, thesub);

   free(origsub);
}

// Split a news file in multiple article files
// News file is generated using Free Agent 1.0
// Article separator is "==========\n"
```

```
void SplitFile(char * filename, int nwsgrp)
{
   FILE * infile;
   FILE * outfile;

   char aline[1025];
   char * anewline;

   char outfilename[255];

   int inheader;
   int inbody;
   int newarticle;

   char from[1025];
   char sfrom[1025];
   char subject[1025];
   char ssubject[1025];
   char mess_id[1025];
   char references[1025];
   char date[1025];
   char sdate[1025];

   char * areference;
   char * fixs;
   int refcount;

   char sd[11];
   char st[9];

   fixs=NULL;

   printf("Beginning SplitFile(%s)\n", filename);

   infile=fopen(filename, "rt");

   // Skip first line as it as an article delimiter

   if (!fgets(aline, 1024, infile))
   {
      fclose(infile);
      printf("End of processing %s. EOF reached.\n", filename);
      return;
   }

   if (strcmp(ARTICLEDELIMITER,aline)!=0)
   {
      printf("File %s not in correct format.\n", filename);
      fclose(infile);
      return;
   }

   // For all the articles in file...

   do
   {

      // We are now in the header of an article

      strcpy(from, "\0");
      strcpy(sfrom, "\0");
      strcpy(subject, "\0");
      strcpy(ssubject, "\0");
      strcpy(date, "\0");
```

```
strcpy(sdate, "\0");
strcpy(mess_id, "\0");
strcpy(references, "\0");
newarticle=true;

inheader=true;
inbody=true;
fgets(aline, 1024, infile);

while (inheader)
{
   strupr(aline);

   if (strncmp(aline, "FROM:", 5)==0)
   {
      strcpy(from, aline);
      strcpy(sfrom, aline);
      Skip(sfrom, 6);
      CRToNull(sfrom);
   }

   if (strncmp(aline, "SUBJECT:", 8)==0)
   {
      strcpy(subject, aline);
      strcpy(ssubject, aline);
      Skip(ssubject, 9);
      CRToNull(ssubject);
   }

   if (strncmp(aline, "MESSAGE-ID:", 11)==0)
   {
      strcpy(mess_id, aline);
      Skip(mess_id, 12);
      CRToNull(mess_id);
   }

   if (strncmp(aline, "REFERENCES:", 11)==0)
   {
      strcpy(references, aline);
      Skip(references, 12);
      CRToNull(references);
   }

   if (strncmp(aline, "DATE:", 5)==0)
   {
      strcpy(date, aline);
      strcpy(sdate, aline);
      Skip(sdate, 6);

      if (*sdate>'9')
         Skip(sdate, 5);

      CRToNull(sdate);

      DateToDateTime(sdate, sd, st);
   }

   if (strncmp(aline, "\n", 1)==0)
       inheader=false;

   if (inheader)
      fgets(aline, 1024, infile);
}
```

```
// We should check the database for the article

if (!NWS_CheckArticle(mess_id))
{
   // We have got a new article. Save it database

   artnum++;

   if ((artnum % 20)==0)
      printf("*\n");

   // Save the References

   refcount=0;
   areference=strtok(references, " \0");
   while (areference)
   {
      refcount++;
      NWS_AddArticleReference(mess_id, areference, refcount);
      areference=strtok(NULL, " \0");
   }

   FixSubject(ssubject, fixs);

   TextToHTML(fixs);

   NWS_AddArticle(artnum, mess_id, sfrom, ssubject, fixs, sd, st, nwsgrp,
       refcount);

   NWS_Commit();
}
else
{
   // Article already in database
   printf("Article %s in DB\n", mess_id);
   newarticle=false;
}

// Process the body of article accordingly

if (newarticle)
{
   sprintf(outfilename, OUTPUTNEWS, artnum);
   outfile=fopen(outfilename, "wt");
   if ferror(outfile)
   {
      printf("Error on file in output !\n");
      exit(1);
   }

   fprintf(outfile, "%s\n\n", ssubject);

}

// Iterate till the end of body...

inbody=true;
anewline=fgets(aline, 1024, infile); // Get a new line

while ((anewline) && (inbody)) // Not at EOF && still in body...
{
   if (strcmp(ARTICLEDELIMITER,aline)==0)
      inbody=false; // Reached an article delimiter ?
```

```
        if (inbody && newarticle) // Line in article body && it's a new article
           fprintf(outfile, "%s", aline);

        if (inbody) // Still in body. Get a new line.
           anewline=fgets(aline, 1024, infile);
     }

     if (newarticle)
        fclose(outfile);
  }
  while (anewline);

  fclose(infile);

  printf("Ending SplitFile(%s)\n", filename);
}

void OutputFooterHTML(FILE * fp)
{
  fprintf(fp, "<HR>For more information on this <A HREF=");
  fprintf(fp, AHYPSYS);
  fprintf(fp, ">hypertext system</A>, contact : ");
  fprintf(fp, "<A HREF=\"http://www.info.fundp.ac.be/~vdh/\">Vincent
      D'Haeyere</A>\n");
  fprintf(fp, "</BODY></HTML>\n");
}

void OutputArticleReferenced(FILE * outfile, char * artid, int refcount)
{
  int status;
  int artnum;
  int refcount2;
  char reference[100];
  char artsubject[255];
  char artfrom[255];
  char artdate[11];
  char arttime[9];

  int numberart;
  int authornum;

  int i;
  int present;

  fprintf(outfile, "<BR>In answer to %i article(s) : (if present in
      database)<P>\n<UL><FONT SIZE=\"1\">", refcount);

  for(i=1;i<=refcount; i++)
  {
     NWS_GetReferencedArticle(artid, i, reference);

     NWS_GetArticleByID(reference, artsubject, artfrom, artdate, arttime,
         refcount2, artnum, present);

     if (present)
     {
        TextToHTML(artfrom);
        TextToHTML(artsubject);

        fprintf(outfile, "<LI><A HREF=");
        fprintf(outfile, AARTICLE, artnum);
        fprintf(outfile, ">%s</A> (%s - %s) by ", artsubject, artdate,
            arttime);
        NWS_GetAuthorInfo(artfrom, numberart, authornum);

        if (numberart>1)
        {
           fprintf(outfile, "<A HREF=");
           fprintf(outfile, AAUTHORI, authornum);
           fprintf(outfile, ">%s</A>\n", artfrom);
        }
        else
        {
           fprintf(outfile, "<A HREF=");
           fprintf(outfile, AARTICLE, artnum);
           fprintf(outfile, ">%s</A>\n", artfrom);
        }
     }
  }

  fprintf(outfile, "</FONT></UL><HR><P>\n");
}

void OutputArticleHTML(FILE * outfile, FILE * infile, char * artid, char *
    artfrom, char * artsubject, int refcount, int nwsgrp, char * nwsgrpdesc, int
    precedant, int suivant)
{
  char aline[1025];
  int numberart;
  int authornum;

  NWS_GetAuthorInfo(artfrom, numberart, authornum);

  TextToHTML(artsubject);
  TextToHTML(artfrom);

  fprintf(outfile, "<HTML><TITLE>%s in %s</TITLE><BODY><FONT SIZE=\"1\">",
      artsubject, nwsgrpdesc);

  if (precedant)
  {
     fprintf(outfile, "<A HREF=");
     fprintf(outfile, AARTICLE, precedant);
     fprintf(outfile, ">Previous article</A> - ");
  }

  fprintf(outfile, "<A HREF=");
  fprintf(outfile, AARTIIND, nwsgrp);
  fprintf(outfile, ">Index of articles </A> - ");

  if (suivant)
  {
     fprintf(outfile, "<A HREF=");
     fprintf(outfile, AARTICLE, suivant);
     fprintf(outfile, ">Next Article</A> - ");
  }

  fprintf(outfile, "<A HREF=");
  fprintf(outfile, AAUTHIND,nwsgrp);
  fprintf(outfile, ">Index of authors</A> - ");

  fprintf(outfile, "<A HREF=");
  fprintf(outfile, ASUBJIND, nwsgrp);
  fprintf(outfile, ">Index by subject</A>\n");

  fprintf(outfile, "</FONT>");
```

```
    fprintf(outfile, "<H3>Newsgroup : %s</H3><P><H3>Subject : %s</H3><P>",
        nwsgrpdesc, artsubject);

    if (numberart>1)
    {
       fprintf(outfile, "<H3>From : <A HREF=");
       fprintf(outfile, AAUTHORI, authornum);
       fprintf(outfile, ">%s</A></H3> (%i other articles)\n", artfrom,
           (numberart-1));
    }
    else fprintf(outfile, "<H3>From : %s</H3>\n", artfrom);

    if (refcount)
       OutputArticleReferenced(outfile, artid, refcount);

    fprintf(outfile, "<P>\n");

    fprintf(outfile, "<P><PRE>");

    while (fgets(aline, 1024, infile))
    {
       TextToHTML(aline);
       fprintf(outfile, "%s", aline);
    }

    fprintf(outfile, "</PRE>\n<MARKER>\n");

    OutputFooterHTML(outfile);
}

void OutputArticleIndexHTML(FILE * indfile, char * artfrom, char * artsubject,
     int anum)
{
    int numberart;
    int authornum;
    int artnum;

    fprintf(indfile, "<LI><A HREF=");
    fprintf(indfile, AARTICLE, anum);
    fprintf(indfile, ">%s</A> - <FONT SIZE=\"2\">", artsubject);

    NWS_GetAuthorInfo(artfrom,numberart, authornum);

    if (numberart>1)
    {
       fprintf(indfile, "<A HREF=");
       fprintf(indfile, AAUTHORI, authornum);
       fprintf(indfile, ">%s</A>", artfrom);
    }
    else
    {
       NWS_GetAuthorOnlyArticle(artfrom, artnum);

       fprintf(indfile, "<A HREF=");
       fprintf(indfile, AARTICLE, artnum);
       fprintf(indfile, ">%s</A>", artfrom);
    }

    fprintf(indfile,"</FONT>\n");
}

void OrganizeNewsgroupBySubject(int nwsgrp, char * nwsgrpdesc)
{
    char * artid;
    char * artsubject;
    char * precartsubject;

    char * subfname;
    FILE * outfile;

    int status;
    int artnum;
    int counter=0;

    artid=(char *)malloc(101);
    artsubject=(char *)malloc(255);
    precartsubject=(char *)malloc(255);
    subfname=(char *)malloc(170);

    sprintf(subfname, OUTPUTSUBJECT, nwsgrp);

    outfile=fopen(subfname, "wt");

    fprintf(outfile,"<HTML><TITLE>Index of %s by subject</TITLE>\n", nwsgrpdesc);
    fprintf(outfile,"<H1>Index of %s organised by subject</H1>\n<UL>\n",
        nwsgrpdesc);

    NWS_OpenBySubject(nwsgrp, status);

    NWS_FetchBySubject(artsubject, artid, artnum, status);

    while (status)
    {
       if (strcmp(artsubject, precartsubject)==0)
       {
          // Same subject as previous
          counter++;

          if ((counter%10)==0) fprintf(outfile, "<BR>");

          fprintf(outfile, "<A HREF=");
          fprintf(outfile, AARTICLE, artnum);
          fprintf(outfile, ">%i</a> \n", counter);
       }
       else
       {
          // Another subject
          counter=1;
          fprintf(outfile, "<BR><LI>%s<BR>\n", artsubject);
          fprintf(outfile, "<A HREF=");
          fprintf(outfile, AARTICLE, artnum);
          fprintf(outfile, ">%i</a> \n",counter);

          strcpy(precartsubject, artsubject);
       }

       NWS_FetchBySubject(artsubject, artid, artnum, status);
    }

    fprintf(outfile, "</UL><BR>\n");

    OutputFooterHTML(outfile);

    NWS_CloseBySubject();

    fclose(outfile);

    free(artid);
```

```
    free(artsubject);
    free(precartsubject);
    free(subfname);
}

void OrganizeNewsGroup(int nwsgrp, char * nwsgrpdesc)
{
    int anum1;
    int anum2;
    int status1;
    int status2;
    int refcount1;
    int refcount2;
    char artid1[101];
    char artid2[101];
    char artfrom1[255];
    char artfrom2[255];
    char artsubject1[255];
    char artsubject2[255];

    int artprecedant;

    char htmlfilename[150];
    char articlefilename[150];
    char indexfilename[150];

    FILE * htmlfile;
    FILE * infile;
    FILE * indfile;

    printf("Beginning of OrganizeNewsGroup(%s)\n", nwsgrpdesc);

    sprintf(indexfilename, OUTPUTINDEXNEWS, nwsgrp);
    indfile=fopen(indexfilename, "wt");

    fprintf(indfile, "<HTML><TITLE>Index of articles in %s, classified by
        date</TITLE>\n<BODY>", nwsgrpdesc);
    fprintf(indfile, "<H1>Index of articles in %s</H1><HR><UL>\n", nwsgrpdesc);

    NWS_OpenNewsGroup(nwsgrp, status1);

    artprecedant=0;
    NWS_ArticleFetch(anum1, artid1, artfrom1, artsubject1, refcount1, status1);

    if (status1)
        NWS_ArticleFetch(anum2, artid2, artfrom2, artsubject2, refcount2,
            status2);
    else
    {
        NWS_CloseNewsGroup();
        return;
    }

    // For all article (but the last) in newsgroup...

    while (status2)
    {
        // Work on this article...

        sprintf(htmlfilename, OUTPUTHTML, anum1);
        sprintf(articlefilename, OUTPUTNEWS, anum1);

        htmlfile=fopen(htmlfilename, "wt");
        infile=fopen(articlefilename, "rt");

        OutputArticleHTML(htmlfile, infile, artid1, artfrom1, artsubject1,
            refcount1, nwsgrp, nwsgrpdesc, artprecedant, anum2);

        OutputArticleIndexHTML(indfile, artfrom1, artsubject1, anum1);

        artprecedant=anum1;

        // End of work on this article

        fclose(infile);
        fclose(htmlfile);

        anum1=anum2;
        refcount1=refcount2;
        strcpy(artid1, artid2);
        strcpy(artfrom1, artfrom2);
        strcpy(artsubject1, artsubject2);

        NWS_ArticleFetch(anum2, artid2, artfrom2, artsubject2, refcount2,
            status2);
    }

    // For the last article in newsgroup

    sprintf(htmlfilename, OUTPUTHTML, anum1);
    sprintf(articlefilename, OUTPUTNEWS, anum1);

    htmlfile=fopen(htmlfilename, "wt");
    infile=fopen(articlefilename, "rt");

    OutputArticleHTML(htmlfile, infile, artid1, artfrom1, artsubject1, refcount1,
        nwsgrp, nwsgrpdesc, artprecedant, 0);

    OutputArticleIndexHTML(indfile, artfrom1, artsubject1, anum1);

    fclose(infile);
    fclose(htmlfile);

    // Thats all folk's...

    NWS_CloseNewsGroup();

    fprintf(indfile, "<UL><P>\n");

    OutputFooterHTML(indfile);

    fclose(indfile);

    printf("Ending of OrganizeNewsGroup(%s)\n", nwsgrpdesc);
}

void OrganizeNewsgroupUsingClusters()
{
    char aline[1024];
    char * theline;
    int artnumtotal;
    int artincluster;
    int i;
    int status;
    int present;
    int cluster;
    char artfrom[255];
    char artsubject[255];
    FILE * infile;
```

```
   FILE * outfile;
   char filein[170];
   char fileout[170];
   int getout;
   int numberart;
   int authornum;
   int artdocid;
   int artdocid2;

   artnumtotal=NWS_GetNumArticle();

   for(i=1;i<=artnumtotal; i++) //artnumtotal
   {
      printf("Working on article N: %i\n", i);

      NWS_GetDocIDForArticle(i, artdocid);

      cluster=NWS_GetClusterForArticle(artdocid);

      artincluster=NWS_GetClusterCountButItself(cluster, artdocid);

      if (artincluster)
      {
         sprintf(filein, OUTPUTHTML, i);
         sprintf(fileout, OUTPUTHTM2, i);

         infile=fopen(filein, "rt");
         outfile=fopen(fileout, "wt");

         theline=fgets(aline, 1024, infile);
         getout=false;
         while ((theline)&&(!getout))
         {
            if (strcmp(aline, "<MARKER>\n")==0)
              getout=true;
            else
            {
              fprintf(outfile, "%s", aline);
              theline=fgets(aline, 1024, infile);
            }
         }

         if ((getout)&&(!theline))
         {
            printf("Error in HTML file...\n");
            exit(0);
         }

         fprintf(outfile, "<HR>See also the following articles that are
             (presumably) related to a similar subject.<P><UL><FONT
             SIZE=\"1\">\n");

         NWS_OpenNewsByCluster(cluster, artdocid, status);

         NWS_FetchArticleByCluster(artdocid2, status);

         while (status)
         {
            NWS_GetArticleByDocID(artdocid2, artnum);

            NWS_GetArticleByNum(artnum, artsubject, artfrom, present);

            NWS_GetAuthorInfo(artfrom,numberart, authornum);
            TextToHTML(artfrom);
            TextToHTML(artsubject);

            fprintf(outfile, "<LI><A HREF=");
            fprintf(outfile, AARTICLE, artnum);
            fprintf(outfile, ">%s</A> - ", artsubject);

            if (numberart>1)
            {
               fprintf(outfile, "<A HREF=");
               fprintf(outfile, AAUTHORI, authornum);
               fprintf(outfile, ">%s</A>\n", artfrom);
            }
            else fprintf(outfile, "%s<P>\n", artfrom);

            NWS_FetchArticleByCluster(artdocid2, status);
         }

         NWS_CloseNewsByCluster();

         fprintf(outfile, "</FONT></UL>\n");

         while (fgets(aline, 1024, infile))
            fprintf(outfile, "%s", aline);

         fclose(infile);
         fclose(outfile);
      }
   }
}

void OutputAuthorArticleIndexHTML(char * artfrom, int numberart, int authnum)
{
   char filename[170];
   FILE * authfile;

   int status;

   int artnum;
   int refcount;
   char artsubject[255];
   char artdate[11];
   char arttime[9];

   sprintf(filename, OUTPUTAUTH, authnum);
   authfile=fopen(filename, "wt");

   fprintf(authfile, "<HTML><TITLE>Index of %i articles written by
       %s</TITLE><BODY>", numberart, artfrom);
   fprintf(authfile, "%s has written %i articles.<P>", artfrom, numberart);

   NWS_OpenArticleFrom(artfrom, status);

   NWS_ArticleFromFetch(artnum, artsubject, artdate, arttime, refcount, status);

   while (status)
   {
      TextToHTML(artsubject);

      fprintf(authfile, "%s - %s - <A HREF=", artdate, arttime);
      fprintf(authfile, AARTICLE, artnum);
      fprintf(authfile, ">%s</A><P>\n", artsubject);

      NWS_ArticleFromFetch(artnum, artsubject, artdate, arttime, refcount,
```

```
            status);
      }

      NWS_CloseArticleFrom();

      fclose(authfile);
}

void OrganizeAuthors()
{
      char artfrom[255];
      char artsubject[255];
      int artnum;
      int numberart;
      int authornum;
      int status;

      char filename[170];
      char indexname[170];

      FILE * authfile;
      FILE * indexfile;

      printf("Beginning of OrganizeAuthors\n");

      NWS_Commit();

      NWS_InsertInAuthorTable();

      NWS_Commit();

      authornum=0;

      NWS_OpenAuthors(status);

      sprintf(indexname, OUTPUTINDEXAUTHOR);
      indexfile=fopen(indexname, "wt");

      fprintf(indexfile, "<HTML><TITLE>Index of authors</TITLE><BODY><H1>Index of
          authors</H1><HR><P>");

      NWS_AuthorFetch(artfrom, numberart, status);
      while (status)
      {
         if (numberart>1)
         {
            authornum++;

            NWS_AuthorInFile(authornum, artfrom);

            TextToHTML(artfrom);

            OutputAuthorArticleIndexHTML(artfrom, numberart, authornum);

            fprintf(indexfile, "<A HREF=");
            fprintf(indexfile, AAUTHORI, authornum);
            fprintf(indexfile, ">%s</A> - %i articles<P>\n", artfrom, numberart);
         }
         else
         {
            NWS_GetAuthorOnlyArticle(artfrom, artnum);

            TextToHTML(artfrom);
            fprintf(indexfile, "%s - <A HREF=", artfrom);
            fprintf(indexfile, AARTICLE, artnum);
            fprintf(indexfile, ">1 article</A><P>\n");
         }

         NWS_AuthorFetch(artfrom, numberart, status);
      }

      OutputFooterHTML(indexfile);

      fclose(indexfile);

      NWS_CloseAuthors();

      NWS_Commit();

      printf("End of OrganizeAuthors\n");
}

void OrganizeAuthorsNewsgroup(int nwsgrp, char * nwsgrpdesc)
{
      char artfrom[255];
      int numberart;
      int totalnumberart;
      int authornum;
      int anum;

      FILE * indnwsgrp;
      char indexname[170];

      int status;

      NWS_OpenAuthorsNewsgroup(nwsgrp, status);

      DB_CheckError();

      sprintf(indexname, "d:\\vincent\\dev\\nws\\html\\nws%05i.htm", nwsgrp);
      indnwsgrp=fopen(indexname, "wt");

      fprintf(indnwsgrp, "<HTML><TITLE>Index of authors for %s.</TITLE><BODY>",
          nwsgrpdesc);
      fprintf(indnwsgrp, "<H1>Index of authors for %s</H1><HR>\n", nwsgrpdesc);

      NWS_AuthorNewsgroupFetch(artfrom, numberart, status);
      while (status)
      {
         NWS_GetAuthorInfo(artfrom, totalnumberart, authornum);

         TextToHTML(artfrom);

         if (authornum)
         {
            fprintf(indnwsgrp, "<A HREF=");
            fprintf(indnwsgrp, AAUTHORI, authornum);
            fprintf(indnwsgrp, ">%s</A> - %i article(s)", artfrom, numberart);
         }
         else
         {
            NWS_GetAuthorOnlyArticle(artfrom, anum);

            fprintf(indnwsgrp, "%s - <A HREF=", artfrom);
            fprintf(indnwsgrp, AARTICLE, anum);
            fprintf(indnwsgrp, ">1 article</A>");
         }
```

```
      if (totalnumberart>numberart)
         fprintf(indnwsgrp, "<FONT SIZE=\"1\">(%i of %i articles in
            DB)</FONT><P>\n", numberart, totalnumberart);
      else fprintf(indnwsgrp, "<P>\n");

      NWS_AuthorNewsgroupFetch(artfrom, numberart, status);
   }

   NWS_CloseAuthorsNewsgroup();

   DB_CheckError();

   OutputFooterHTML(indnwsgrp);

   fclose(indnwsgrp);
}

void split_main()
{
   printf("News Splitter\n*************\n\n");

   artnum=NWS_GetNumArticle();
   printf("%i article(s) in database.\n", artnum);

   // datefile=fopen("d:\\vincent\\dev\\nws\\out\\date.out", "wt");

   SplitFile("c:\\vincent\\dev\\nws\\althyp.txt", ALTHYP);
   SplitFile("c:\\vincent\\dev\\nws\\comptext.txt", COMPTEXT);
   SplitFile("c:\\vincent\\dev\\nws\\linann.txt", LINANN);
   SplitFile("c:\\vincent\\dev\\nws\\secmisc.txt", SECMISC);
   SplitFile("c:\\vincent\\dev\\nws\\secunix.txt", SECUNIX);
// SplitFile("c:\\vincent\\dev\\nws\\sysintel.txt", SYSINTEL);

// fclose(datefile);

   OrganizeAuthors();

   OrganizeAuthorsNewsgroup(ALTHYP, "alt.hypertext");
   OrganizeAuthorsNewsgroup(COMPTEXT, "comp.text");
   OrganizeAuthorsNewsgroup(LINANN, "comp.linux.announce");
   OrganizeAuthorsNewsgroup(SECMISC, "comp.secutiry.misc");
   OrganizeAuthorsNewsgroup(SECUNIX, "comp.security.unix");
// OrganizeAuthorsNewsgroup(SYSINTEL, "comp.sys.intel");

   OrganizeNewsGroup(ALTHYP, "alt.hypertext");
   OrganizeNewsGroup(COMPTEXT, "comp.text");
   OrganizeNewsGroup(LINANN, "comp.linux.announce");
   OrganizeNewsGroup(SECMISC, "comp.security.misc");
   OrganizeNewsGroup(SECUNIX, "comp.security.unix");
// OrganizeNewsGroup(SYSINTEL, "comp.sys.intel");
}
```

```
#include <stdio.h>
#include "stdfunc.h"
#include "stddefs.h"

int ReadLine(FILE * fileh, char * line, int & length)
{
   unsigned char * pb=(unsigned char *)line;
   int ch;

   length=0;

   ch=getc(fileh);
   while ((ch!=EOF)&&(ch!=13))
   {
      *pb=tolower(ch);
      if (*pb==9)
         *pb=32;
      if ((*pb>=32)&&(*pb<=127))
      {
         pb++;
         length++;
      }
      ch=getc(fileh);
   }

   *pb=0;

   return (ch==EOF) ? true : false;
}

int IsDelimiter(char ch)
{
   return (!IsAlphaNum(ch));
}

int IsAlphaNum(char ch)
{
   return ( ((ch>='0') && (ch<='9')) || ((ch>='a') && (ch<='z')) ) ? true :
       false;
}

int IsAlpha(char ch)
{
   return ( ((ch>='a') && (ch<='z')) ) ? true : false;
}

int AllNum(char * s)
{
   int an=true;
   char * tmp=s;

   while (*tmp && an)
   {
    an=(((*tmp>='0') && (*tmp<='9')) ? true : false);
    tmp++;
   }
   return an;
}

int GetNextWord(char * mot, char * & lastpos)
{
   while (IsDelimiter(*lastpos) && (*lastpos!='\0') )
      lastpos++;
   if (!*lastpos)
      return false; // end of string reached.

   do
   {
      *mot++=*lastpos++;
   }
   while (IsAlphaNum(*lastpos) && (*lastpos!='\0'));

   *mot='\0';

   return true;
}
```

```
#include <iostream.h>

#include "stemc.h"

stem::stem()

{

   string=NULL;

   number=1;

}

stem::~stem()

{

}

stem::stem(char * mot):mystring(mot)

{

   number=1;

   weight=0.0;

}

// Function definition

ostream& operator<<(ostream& s, stem& n)

{
   if (n.string)
      return s << (char *) n.string << ':' << n.number <<'(' << n.weight << ')'
          << endl;
      else return s << "Empty stem" << endl;
};

dmstem::dmstem()

{

   string=NULL;

   number=1;

   lastdoc=-1;

   numdoc=0;

}

dmstem::dmstem(char * mot):mystring(mot)

{

   number=1;

   lastdoc=-1;

   numdoc=0;

}

dmstem::~dmstem()

{

}

// Function definition

ostream& operator<<(ostream& s, dmstem& n)

{
   if (n.string)
   {
      s << (char *) n.string << ':' << n.number << " in " << n.numdoc << " docs
          " << endl;
      return s;
   }
   else return s << "Empty stem" << endl;
};

ostream& operator<<(ostream& s, dmstem * n)
{

   if (n->string)

   {

      s << (char *) n->string << ':' << n->number << " in " << n->numdoc << "
          docs " << endl;

      return s;

   }

   else return s << "Empty stem" << endl;

}
```

```
c:\vincent\dev\nws\althyp.txt,alt.hypertext
c:\vincent\dev\nws\secmisc.txt,comp.security.misc
c:\vincent\dev\nws\secunix.txt,comp.security.unix
c:\vincent\dev\nws\comptext.txt,comp.text
c:\vincent\dev\nws\linann.txt,comp.os.linux.announce
```

```
#
# Borland C++ IDE generated makefile
#
.AUTODEPEND

#
# Borland C++ tools
#
IMPLIB  = Implib
BCC     = Bcc +BccW16.cfg
TLINK   = TLink
TLIB    = TLib
BRC     = Brc
TASM    = Tasm
#
# IDE macros
#

#
# Options
#
IDE_LFLAGS =  -LC:\BC4\LIB
IDE_RFLAGS =  -IC:\BC4\INCLUDE
LLATW16_irnewsdexe =  -Twe -LC:\BC4\LIB;C:\OCELOT; -x
RLATW16_irnewsdexe =  -31 -IC:\BC4\INCLUDE;C:\OCELOT;C:\VINCENT\DEV;
BLATW16_irnewsdexe =
LEAT_irnewsdexe = $(LLATW16_irnewsdexe)
REAT_irnewsdexe = $(RLATW16_irnewsdexe)
BEAT_irnewsdexe = $(BLATW16_irnewsdexe)
CLATW16_ddbddbocelotbdbmswdlib =  -ml -WS
LLATW16_ddbddbocelotbdbmswdlib =  -Twe
RLATW16_ddbddbocelotbdbmswdlib =  -31
BLATW16_ddbddbocelotbdbmswdlib =
CEAT_ddbddbocelotbdbmswdlib = $(CEAT_irnewsdexe) $(CLATW16_ddbddbocelotbdbmswdli
    b)
LEAT_ddbddbocelotbdbmswdlib = $(LEAT_irnewsdexe) $(LLATW16_ddbddbocelotbdbmswdli
    b)
REAT_ddbddbocelotbdbmswdlib = $(REAT_irnewsdexe) $(RLATW16_ddbddbocelotbdbmswdli
    b)
BEAT_ddbddbocelotbdbmswdlib = $(BEAT_irnewsdexe) $(BLATW16_ddbddbocelotbdbmswdli
    b)

#
# Dependency List
#
Dep_irnews = \
   irnews.exe

irnews : BccW16.cfg $(Dep_irnews)
  echo MakeNode irnews

Dep_irnewsdexe = \
   stdfunc.obj\
   document.obj\
   stemc.obj\
   mystring.obj\
   dbirnwsc.obj\
   clusters.obj\
   splitnws.obj\
   ir.obj\
   irnews.obj

irnews.exe : $(Dep_irnewsdexe)
  $(TLINK)   @&&|
 /v $(IDE_LFLAGS) $(LEAT_irnewsdexe) +
C:\BC4\LIB\c0wl.obj+
stdfunc.obj+
document.obj+
stemc.obj+
mystring.obj+
dbirnwsc.obj+
clusters.obj+
splitnws.obj+
ir.obj+
irnews.obj
$<,$*
..\..\ocelot\dbmsw.lib+
C:\BC4\LIB\import.lib+
C:\BC4\LIB\mathwl.lib+
C:\BC4\LIB\cwl.lib
irnews.def
|

stdfunc.obj :  stdfunc.cpp
  $(BCC)   -c $(CEAT_irnewsdexe) -o$@ stdfunc.cpp

document.obj :  document.cpp
  $(BCC)   -c $(CEAT_irnewsdexe) -o$@ document.cpp

stemc.obj :  stemc.cpp
  $(BCC)   -c $(CEAT_irnewsdexe) -o$@ stemc.cpp

mystring.obj :  mystring.cpp
  $(BCC)   -c $(CEAT_irnewsdexe) -o$@ mystring.cpp

dbirnwsc.obj :  dbirnwsc.cpp
  $(BCC)   -c $(CEAT_irnewsdexe) -o$@ dbirnwsc.cpp

clusters.obj :  clusters.cpp
  $(BCC)   -c $(CEAT_irnewsdexe) -o$@ clusters.cpp

splitnws.obj :  splitnws.cpp
  $(BCC)   -c $(CEAT_irnewsdexe) -o$@ splitnws.cpp

ir.obj :  ir.cpp
  $(BCC)   -c $(CEAT_irnewsdexe) -o$@ ir.cpp

irnews.obj :  irnews.cpp
  $(BCC)   -c $(CEAT_irnewsdexe) -o$@ irnews.cpp

# Compiler configuration file
BccW16.cfg :
   Copy &&|
-R
-v
-vi
-X-
-H
-IC:\BC4\INCLUDE
-H=irnews.csm
-ml
-WS
-IC:\BC4\INCLUDE;C:\OCELOT;C:\VINCENT\DEV;
-Og
-Ot
-O-a
-Z
-O
```

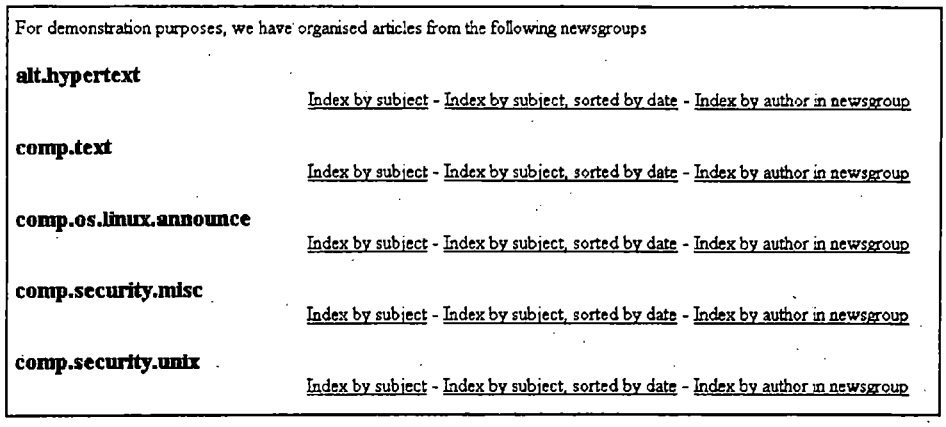
```
-Oe
-Ol
-Ob
-OW
-Om
-Op
-Oi
-Ov
-4
| $@
```

Annexe B :
Utilisation d'un système hypertexte construit par notre prototype

Dans les pages suivantes, nous présentons un exemple d'utilisation de notre système hypertexte. Nous rappelons qu'un système hypertexte construit par ce programme est accessible jusqu'à la défense de ce mémoire à l'URL suivante : `http://www.info.fundp.ac.be/~vdh/news/index.html`. La disponibilité du système à une date ultérieure est fortement improbable pour des raisons d'administration système.

Première étape

L'utilisateur utilise pour la première fois le système hypertexte. Il consulte l'index des index des groupes de discussion.


For demonstration purposes, we have organised articles from the following newsgroups

alt.hypertext
Index by subject - Index by subject, sorted by date - Index by author in newsgroup

comp.text
Index by subject - Index by subject, sorted by date - Index by author in newsgroup

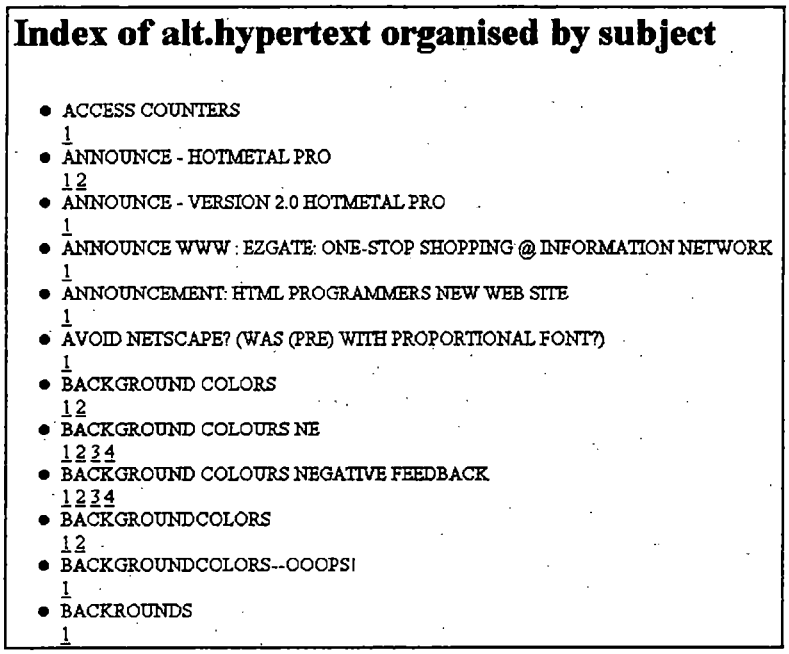
comp.os.linux.announce
Index by subject - Index by subject, sorted by date - Index by author in newsgroup

comp.security.misc
Index by subject - Index by subject, sorted by date - Index by author in newsgroup

comp.security.unix
Index by subject - Index by subject, sorted by date - Index by author in newsgroup

Deuxième étape

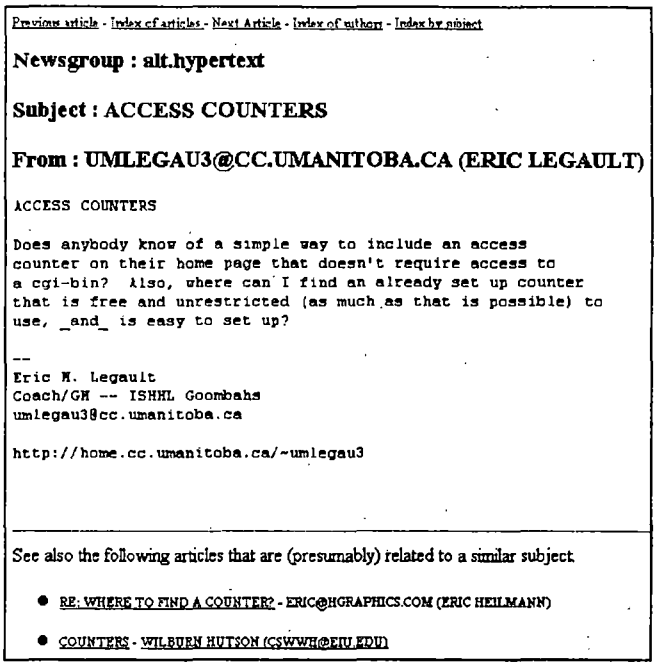
Il décide de consulter la liste des articles du groupe *alt.hypertext* et sélectionne l'ancre hypertexte *Index by subject*. Il obtient l'écran suivant.


Index of alt.hypertext organised by subject

- ACCESS COUNTERS
  1
- ANNOUNCE - HOTMETAL PRO
  1 2
- ANNOUNCE - VERSION 2.0 HOTMETAL PRO
  1
- ANNOUNCE WWW : EZGATE: ONE-STOP SHOPPING @ INFORMATION NETWORK
  1
- ANNOUNCEMENT: HTML PROGRAMMERS NEW WEB SITE
  1
- AVOID NETSCAPE? (WAS (PRE) WITH PROPORTIONAL FONT?)
  1
- BACKGROUND COLORS
  1 2
- BACKGROUND COLOURS NE
  1 2 3 4
- BACKGROUND COLOURS NEGATIVE FEEDBACK
  1 2 3 4
- BACKGROUNDCOLORS
  1 2
- BACKGROUNDCOLORS--OOOPS!
  1
- BACKROUNDS
  1

Troisième étape

L'utilisateur décide de consulter l'article portant sur les compteurs d'accès à une page (*access counters*). Il sélectionne le seul article consacré à ce sujet.



Previous article - Index of articles - Next Article - Index of authors - Index by subject

Newsgroup : alt.hypertext

Subject : ACCESS COUNTERS

From : UMLEGAU3@CC.UMANITOBA.CA (ERIC LEGAULT)

```
ACCESS COUNTERS

Does anybody know of a simple way to include an access
counter on their home page that doesn't require access to
a cgi-bin?  Also, where can I find an already set up counter
that is free and unrestricted (as much as that is possible) to
use, _and_ is easy to set up?

--
Eric M. Legault
Coach/GM -- ISHHL Goombahs
umlegau3@cc.umanitoba.ca

http://home.cc.umanitoba.ca/~umlegau3
```

See also the following articles that are (presumably) related to a similar subject

- RE: WHERE TO FIND A COUNTER? - ERIC@HGRAPHICS.COM (ERIC HEILMANN)
- COUNTERS - WILBURN HUTSON (CSWWH@EIU.EDU)

Quatrième étape

L'utilisateur prend connaissance de l'article. Il est intéressé par le sujet et décide de consulter un des articles présenté au bas de la page. Ces articles font partie de la même classe issue du processus de *clustering*. Il choisit l'article « Re : w*here to find a counter* ».

RE: WHERE TO FIND A COUNTER?

Kevin Lyles (klyles@rs6000.cmp.ilstu.edu) wrote:
: I would like to know where I can find a counter to put on my Homepage to see
: how many people travel there....I've seen it before somewhere on someone's
: page but that server is not accepting anymore counters. Any suggetions?

: klyles@rs6000.cmp.ilstu.edu

Try http://www.cybercon.com/webtools/

--
****
* ERIC HEILMANN, Owner *
* Heilmann Graphics - Desktop and Online Publishing *
* 6167 Jarvis Avenue, Suite #319, Newark, CA 94560 (510) 792-4974 *
* eric@hgraphics.com *
* Check out our Home Page at http://www.hgraphics.com/ *
****

See also the following articles that are (presumably) related to a similar subject.

- ACCESS COUNTERS - UMLEGAU3@CC.UMANITOBA.CA (ERIC LEGAULT)
- COUNTERS - WILBURN HUTSON (CSWWH@EIU.EDU)

Cinquième étape

L'utilisateur a le renseignement qu'il désire. Il a consulté l'URL donné en référence dans l'article. Il décide de parcourir le système et consulte l'article suivant dans l'ordre chronologique (ancre *Next Article*).

Previous article - Index of articles - Next Article - Index of authors - Index by subject

Newsgroup : alt.hypertext

Subject : BACKGROUND COLORS

From : WILBURN HUTSON (CSWWH@EIU.EDU)

(1 other articles)

BACKGROUND COLORS

Not long ago I think I remember finding a command other than the '(BODY BKGROUND=#??????)' which would change the screen color. It used the more familiar RGB NOTATION. Now I cannot find that little gem anywhere--- can anyone help.

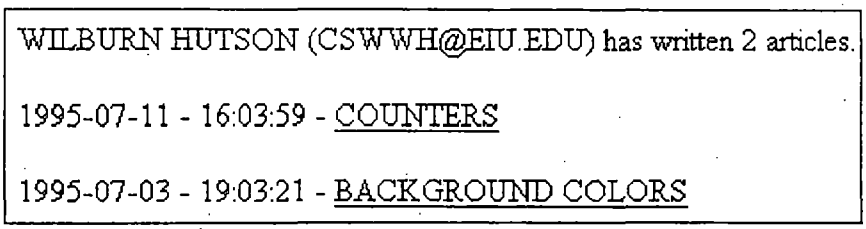
See also the following articles that are (presumably) related to a similar subject.

- RE: BACKGROUND COLORS - GRGREEN@SHADOW.NET (GRGREEN)

For more information on this hypertext system, contact : Vincent D'Haeyere

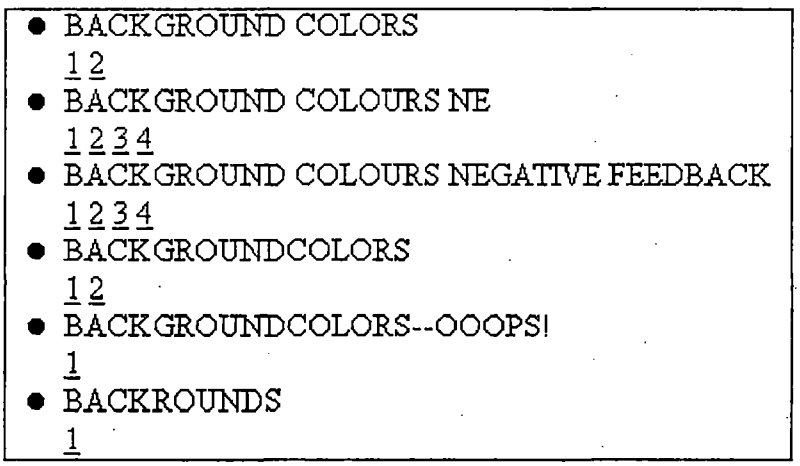
Sixième étape

L'utilisateur lit l'article en question. Il se rend compte que l'auteur a écrit un autre article. Il consulte l'index des articles écrits par cette personne.



Septième étape

L'utilisateur décide d'obtenir des informations complémentaires sur la couleur du fond d'écran. Au lieu de sélectionner l'article concerné, il consulte l'index par sujets. Il constate que 16 articles parlent des couleurs de fond. Il en visionne un au hasard.



Huitième étape

From : GRGREEN@SHADOW.NET (GRGREEN)

(1 other articles)
In answer to 2 article(s) : (if present in database)

- BACKGROUND COLORS (1995-07-03 - 19:03:21) by WILBURN HUTSON (CSWWH@EIU.EDU)

```
RE: BACKGROUND COLORS

In article (3te77f$90k@winx03.informatik.uni-wuerzburg.de), Lonestar (i219@stio1.fh-wuerzburg.de) says:

)(body bkground=(your.gif)) or (body background=(your.gif.again)) is used for background
)IMAGES, not for COLORS !
)
)To change the color, use: (body bgcolor=#(your_color_in_hex_code))
)
Type color can be changed in the same fashion by adding the following
_within_ the above command:

        TEXT="######"

where each ## is a 2 byte hex number representing R,G,@B respectively

Gerald Green (grgreen@shadow.net)
```

See also the following articles that are (presumably) related to a similar subject.

- BACKGROUND COLORS - WILBURN HUTSON (CSWWH@EIU.EDU)

Ici, une constatation s'impose. L'article proposé par le *clustering* est l'article auquel celui-ci répond (cfr. haut de la page WWW). Pourquoi cela ? Une raison probable est que cet article reprend une grande partie du texte original. C'est donc normal que la similarité entre les deux articles soit importante et qu'ils se retrouvent dans le même *cluster*. Mais, d'un autre point de vue, l'index des articles comporte 16 articles consacrés à la couleur de fond des pages WWW. Il apparaît donc que le processus de *clustering* n'est pas assez fin pour détecter ce genre de similitude.

Annexe C : Exemple de *clustering* sur 30 documents

Les pages suivantes reprennent 30 documents qui serviront de base à un exemple de *clustering*. Ensuite, les classes déterminées par l'algorithme seront présentées.

Les documents

Document 1

```
Newsgroups: alt.hypertext
Path:
news.fundp.ac.be!chaos.kulnet.kuleuven.ac.be!Belgium.EU.net!EU.net!news.sprintlink.net!howland.reston.ans.net!v
ixen.cso.uiuc.edu!news.cts.eiu.edu!news
From: Wilburn Hutson <cswwh@eiu.edu>
Subject: BACKGROUND COLORS
Content-Type: text/plain; charset=us-ascii
Message-ID: <DB5M9L.DzD@news.cts.eiu.edu>
Sender: news@news.cts.eiu.edu (news maintenance)
Content-Transfer-Encoding: 7bit
Organization: University Print Center
Mime-Version: 1.0
Date: Mon, 3 Jul 1995 19:03:21 GMT
X-Mailer: Mozilla 1.1N (Windows; I; 16bit)
Lines: 4

Not long ago I think I remember finding a command other than the '<BODY BKGROUND=#??????>'
which would change the screen color. It used the more familiar RGB NOTATION.  Now I cannot find
that little gem anywhere--- can anyone help.
```

Document 2

```
Newsgroups: alt.hypertext
Path:
news.fundp.ac.be!chaos.kulnet.kuleuven.ac.be!Belgium.EU.net!EU.net!news.sprintlink.net!cs.utexas.edu!uwm.edu!ll
l-winken.llnl.gov!uop!pacbell.com!gw2.att.com!oucsboss!oak.cats.ohiou.edu!gh295688
From: gh295688@oak.cats.ohiou.edu (Gregory Michael Hanek)
Subject: Re: backgroundcolors
X-Nntp-Posting-Host: oak.cats.ohiou.edu
Message-ID: <DB5zBL.IuI@boss.cs.ohiou.edu>
Sender: postmaster@oak.cats.ohiou.edu
X-Nntp-Posting-Date: Mon Jul  3 19:45:19 1995
Organization: Ohio University, Athens Ohio, USA
Date: Mon, 3 Jul 1995 23:45:20 GMT
Lines: 19

>html 3.0 allows you to set backgroundcolors. Does anybody know where I
>can find a table of the hex-codes that you have to use to set the
>background color?
>
>Thank you
>
>Martijn de Waal

Dorian Dowse and Gavin Sade have created a nifty colorpicker/hex converter if
you have access to a 256 color mac. Point your browser to http://firehouse.com/
to find the latest, greatest version of Colorhex. Let em know what you think!

Cheers,
--
        Greg Hanek  KB8VRR  gh295688@oak.cats.ohiou.edu
    (My opinions are my opinions, unless I've stolen them.)
       "I've got an attention span of -oh, that's nice."
```

Document 3

```
Newsgroups: alt.hypertext
Path:
news.fundp.ac.be!chaos.kulnet.kuleuven.ac.be!Belgium.EU.net!EU.net!news.sprintlink.net!howland.reston.ans.net!n
ews.moneng.mei.com!uwm.edu!lll-winken.llnl.gov!uop!pacbell.com!gw2.att.com!oucsboss!oak.cats.ohiou.edu!gh295688
From: gh295688@oak.cats.ohiou.edu (Gregory Michael Hanek)
Subject: Re: backgroundcolors--ooops!
X-Nntp-Posting-Host: oak.cats.ohiou.edu
Message-ID: <DB60Fz.JKB@boss.cs.ohiou.edu>
Sender: postmaster@oak.cats.ohiou.edu
X-Nntp-Posting-Date: Mon Jul  3 20:09:33 1995
Organization: Ohio University, Athens Ohio, USA
Date: Tue, 4 Jul 1995 00:09:35 GMT
Lines: 23

>>html 3.0 allows you to set backgroundcolors. Does anybody know where I
>>can find a table of the hex-codes that you have to use to set the
>>background color?
>>
>>Thank you
>>
>>Martijn de Waal
>
>Dorian Dowse and Gavin Sade have created a nifty colorpicker/hex converter if
>you have access to a 256 color mac. Point your browser to http://firehouse.com/
>to find the latest, greatest version of Colorhex. Let em know what you think!

Do let em know what you think, after actually finding the correct URL, which is
```

http://www.firehorse.com/ :). My apologies for the brain-fart. Fingers working, cerebrum not.

Cheers,
--
Greg Hanek KB6VRR gh295688@oak.cats.ohicu.edu
(My opinions are my opinions, unless I've stolen them.)
"I've got an attention span of -oh, that's nice."

Document 4

Path: news.fundp.ac.be!chaos.kulnet.kuleuven.ac.be!Belgium.EU.net!EU.net!gatech!news.fsu.edu!psi!kacmar
From: kacmar@psi.cs.fsu.edu (Chuck Kacmar)
Newsgroups: alt.hypertext
Subject: SIGLINK newsletter
Date: 6 Jul 1995 09:20:31 GMT
Organization: Florida State University Computer Science Department
Lines: 22
Message-ID: <3tg9sv$kih@news.fsu.edu>
Reply-To: kacmar@cs.fsu.edu
NNTP-Posting-Host: psi.cs.fsu.edu

Just a reminder that the SIGLINK newsletter is soliciting
short articles for a special issue on digital libraries.
Submissions should arrive by July 17. The focus of the issue
is on hypertext/hypermedia aspects of digital libraries. If
your digital library system uses hypertext elements in its
operation (e.g., WWW), please send a few short paragraphs
describing this functionality and your project to:

Dr. Chuck Kacmar
Department of Computer Science
Florida State University
203 Love Building
Tallahassee, Florida
32306-4019

email: kacmar@cs.fsu.edu
phone: (904) 644-9661
fax: (904) 644-0058

Document 5

Path:
news.fundp.ac.be!chaos.kulnet.kuleuven.ac.be!Belgium.EU.net!EU.net!Austria.EU.net!newsfeed.ACO.net!news.tuwien.ac.at!news
From: Helmut Wimmer <hwimmer@igw.tuwien.ac.at>
Newsgroups: alt.hypertext
Subject: History of Hypertext/Internet
Date: 6 Jul 1995 11:58:16 GMT
Organization: Technical University of Vienna
Lines: 13
Message-ID: <3tgj4o$dq3@news.tuwien.ac.at>
NNTP-Posting-Host: labmac04.iguw.tuwien.ac.at
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
X-Mailer: Mozilla 1.1N (Macintosh; I; 68K)
X-URL: news:alt.hypertext

I am currently writing a dissertation about the history and
development of Hypertext and Internet.

It would be very helpful to know if there is any SPECIFIC literature
about these two themes.

I already know some internet-sites about the history of
internet, but it would be interesting to know if there
was something like a timetable (esp. for hypertext).

Thanx to everybody !

Document 6

Path:
news.fundp.ac.be!chaos.kulnet.kuleuven.ac.be!Belgium.EU.net!EU.net!sun4nl!news.nic.surfnet.nl!howland.reston.ans.net!newsjunkie.ans.net!news.lava.net!usenet
From: "Charles C. Goodin" <goodin@lava.net>
Newsgroups: alt.hypertext
Subject: What does "Tweaking" a GIF mean?
Date: 7 Jul 1995 09:27:30 GMT
Organization: LavaNet, Inc.
Lines: 7
Message-ID: <3tium2$e80@malasada.lava.net>
NNTP-Posting-Host: dialup017.lava.net
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
X-Mailer: Mozilla 1.2b2 (Windows; I; 16bit)

Alright, I know it sounds stupid but what does "tweaking" a GIF mean? I
figure it is more stupid not to know what it means and not ask.

Thanks!

Charles C. Goodin

Document 7

Path:
news.fundp.ac.be!chaos.kulnet.kuleuven.ac.be!Belgium.EU.net!EU.net!Austria.EU.net!newsfeed.ACO.net!swidir.switch.ch!univ-lyon1.fr!jussieu.fr!oleane!dish.news.pipex.net!pipex!howland.reston.ans.net!newsjunkie.ans.net!news.lava.net!usenet
From: "Charles C. Goodin" <goodin@lava.net>
Newsgroups: alt.hypertext
Subject: Looking for Cheap Web FTP Space
Date: 7 Jul 1995 11:31:45 GMT
Organization: LavaNet, Inc.
Lines: 10
Message-ID: <3tj5v1$jvi@malasada.lava.net>
NNTP-Posting-Host: dialup017.lava.net
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
X-Mailer: Mozilla 1.2b2 (Windows; I; 16bit)

Does anyone know of cheap FTP space? It would be for a non-commercial
page and I need about 5 megs, maybe 7 megs.

Thanks.

Charles C. Goodin
Astral Projection Home Page

http://www.lava.net/~goodin/astral.html

Document 8

Newsgroups: alt.hypertext
Path:
news.fundp.ac.be!chaos.kulnet.kuleuven.ac.be!Belgium.EU.net!EU.net!sun4nl!freya.let.rug.nl!p4305bb.let.rug.nl!s0809845
From: s0809845@let.rug.nl (E.C. Smit)
Subject: textstructures in cyberspace
Sender: news@let.rug.nl (News system at let.rug.nl)
Message-ID: <s0809845.1.2FFD3313@let.rug.nl>
Date: Fri, 7 Jul 1995 13:13:55 GMT
Lines: 16
Nntp-Posting-Host: p4305bb.let.rug.nl
Organization: Faculteit der Letteren, Rijksuniversiteit Groningen, NL
X-Newsreader: Trumpet for Windows [Version 1.0 Rev A]

Hello,

Could anyone help me to find good literature on textstructures in hypertext? I
am busy to write my final paper where I am investigating how existing
brochures could be placed into any form of digital text. This text needs to
be arranged according consumer information search structures. Please help me,
I'm stuck.

My personal e-mail adress is:
E.C.Smit@let.rug.nl

P.s. suggestions on good html, http and hypertext literature are also very
welcome. But keep in mind that I am not a very technical person. Thank you!

Greetings, Lisette

Document 9

Path:
news.fundp.ac.be!chaos.kulnet.kuleuven.ac.be!Belgium.EU.net!EU.net!Germany.EU.net!howland.reston.ans.net!swrinde!elroy.jpl.nasa.gov!decwrl!purdue!mozo.cc.purdue.edu!pasture.ecn.purdue.edu!laird
From: laird@pasture.ecn.purdue.edu (Kyler Laird)
Newsgroups: comp.infosystems.www.authoring.images,comp.infosystems.www.providers,comp.human-factors,alt.hypertext
Subject: Re: text -> html conversion
Date: 6 Jul 1995 15:28:50 GMT
Organization: Purdue University
Lines: 28
Message-ID: <3tgvfi$7bn@mozo.cc.purdue.edu>
References: <3r09im$286@engnews2.Eng.Sun.COM> <3sot1j$rus@nntpd.lkg.dec.com> <3t0sai$t8e@calum.csclub.uwaterloo.ca> <3t2cji$f52@mozo.cc.purdue.edu> <3tf3j6$5ao@calum.csclub.uwaterloo.ca>
NNTP-Posting-Host: pasture.ecn.purdue.edu
X-Newsreader: NN version 6.5.0 (NOV)
Xref: news.fundp.ac.be comp.infosystems.www.authoring.images:2214 comp.infosystems.www.providers:32923 comp.human-factors:6069 alt.hypertext:6485

papresco@calum.csclub.uwaterloo.ca (Paul Prescod) writes:

```
>>Text formatting makes markup guessing
>>easy in many instances, though.  It's easy to make guesses like,
>>"Mark the largest text as level one headers (<H1>)."

>If you are
>encoding important information then "guessing" may not be good enough.  My
>post merely pointed out that there is no general purpose way to convert
>text to HTML because I can invent -=!NEW!=- conventions for text markup
>in any document.  Any important converstion should be manually checked
>and fixed.

No.  "Guessing" becomes "Knowing" when working with a fixed format.
If I want to convert 7000 man pages to HTML, it's probably worth my
effort to come up with an automatic way to do it.  There won't be
any "-=!NEW!=- conventions" because the formatting for a man page
is already pretty well set.  There also won't be any need to
manually check and fix the output if the converter was written well.

I didn't realize that we were talking about converting arbitrary
formats to HTML.  Obviously any format conversion will fail if the
input format changes more than the conversion tool can handle.  My
guess is that's why one of the responses was something like, "Sure,
just put '<pre> </pre>' around it."  (Of course, you'd also need to
escape the special characters...)

--kyler
```

Document 10

```
Path:
news.fundp.ac.be!chaos.kulnet.kuleuven.ac.be!Belgium.EU.net!EU.net!Germany.EU.net!howland.reston.ans.net!news-
ela.megaweb.com!newstf01.news.aol.com!uunet!in1.uu.net!anshar.shadow.net!usenet
From: grgreen@shadow.net (grgreen)
Newsgroups: alt.hypertext
Subject: Re: BACKGROUND COLORS
Date: 8 Jul 1995 02:27:08 GMT
Organization: Shadow Information Services Inc.
Lines: 17
Message-ID: <3tkqds$466@anshar.shadow.net>
References: <DB5M9L.DzD@news.cts.eiu.edu> <3te77f$90k@winx03.informatik.uni-wuerzburg.de>
NNTP-Posting-Host: ppp-mia-41.shadow.net
X-Newsreader: WinVN version 0.82

In article <3te77f$90k@winx03.informatik.uni-wuerzburg.de>, Lonestar <i219@stio1.fh-wuerzburg.de> says:

><body bkground=(your.gif)> or <body background=(your.gif.again)> is used for background
>IMAGES, not for COLORS !
>
>To change the color, use: <body bgcolor=#(your_color_in_hex_code)>
>
Type color can be changed in the same fashion by adding the following
_within_ the above command:

     TEXT="######"

where each ## is a 2 byte hex number representing R,G,&B respectively

Gerald Green (grgreen@shadow.net)
```

Document 11

```
Path:
news.fundp.ac.be!chaos.kulnet.kuleuven.ac.be!Belgium.EU.net!EU.net!Germany.EU.net!howland.reston.ans.net!news-
ela.megaweb.com!newstf01.news.aol.com!uunet!in1.uu.net!news.cybercom.com!usenet
From: Ed Culp <Eculp@cybercom.com>
Newsgroups: alt.hypertext
Subject: Help with this Drop-down
Date: 8 Jul 1995 02:42:14 GMT
Organization: T.G.o.T.
Lines: 55
Message-ID: <3tkra6$ksv@crow.cybercomm.net>
NNTP-Posting-Host: raven.cybercomm.net
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
X-Mailer: Mozilla 1.1N (Windows; I; 16bit)

The following will activate a drop-down type menu, I need someone out
there to tell me what the 'GO' statement calls.  Cannot get it to run,
but assum it uses something like ISMAP uses when active points are hit
with the mouse.

Under "post" action, the action='go' is what I need to know about

Hope for help!

-------------

<CENTER>
<form method="post" action="go">
```

```
<input type="submit" value="Go to:">
<select name="select">
<option selected>About The Gamesters
<option>----------------------
<option>VGA Planets Information
<option>Other VGAP Sites
<option>FTP Sites
<option>Our Games
<option>Programs Needed
<option>Internet Play
<option>VGAP BBS's
<option>----------------------
<option>Star Trek Page
</select>
<input type="hidden" name="[VGA Planets Information]"
value="http://tgot-p3.htm">
<input type="hidden" name="[Other VGAP Sites]"
value="http://sites.html">
<input type="hidden" name="[FTP Sites]"
value="http://ftpsites.html">
<input type="hidden" name="[Our Games]"
value="http://ourgames.html">
<input type="hidden" name="[Programs Needed]"
value="http://needed.html">
<input type="hidden" name="[Internet Play]"
value="http://netplay.html">
<input type="hidden" name="[VGAP BBS's]"
value="http://bbs-vgap.html">
<input type="hidden" name="[Star Trek Page]"
value="http://st-trek.html">
<input type="hidden" name="[About The Gamesters]"
value="http://tgot-p2.html">
</CENTER>

----------------

Thanks for looking,

Ed
Eculp@cybercom.com
```

Document 12

```
Path:
news.fundp.ac.be!chaos.kulnet.kuleuven.ac.be!Belgium.EU.net!EU.net!howland.reston.ans.net!newsjunkie.ans.net!ne
ws.lava.net!usenet
From: "Charles C. Goodin" <goodin@lava.net>
Newsgroups: alt.hypertext
Subject: In-Line Animations Without CGI bin?
Date: 10 Jul 1995 08:44:48 GMT
Organization: LavaNet, Inc.
Lines: 13
Message-ID: <3tqpa0$r54@malasada.lava.net>
NNTP-Posting-Host: dialup010.lava.net
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
X-Mailer: Mozilla 1.2b2 (Windows; I; 16bit)

Is there a way to do in-line animations (I believe its called server
push) without using the CGI bin?  I don't mean a slide show.  I want to
animation to run without changing the URL.

Thank you.

Charles C. Goodin

P.S.  I made a slide show to speed up the processing of a rather big page
with a background GIF at:

http://emporium.turnpike.net/G/goodin/furyu/
```

Document 13

```
Path:
news.fundp.ac.be!chaos.kulnet.kuleuven.ac.be!Belgium.EU.net!EU.net!news.sprintlink.net!holonet!colossus.holonet
.net!nwcs!patrick.west
From: patrick.west@nwcs.org (Patrick West)
Newsgroups: alt.hypertext
Subject: winhelp.exe
Date: Sat,  8 Jul 1995 06:01:00 GMT
Message-ID: <9507112331333576@nwcs.org>
Organization: NWCS Online * Oregon USA * 503-655-3927
Distribution: world
Lines: 15

Hi Folks,

I don't know if this is the place or not, but the listing reads
hypertext.
```

I working on a project of creating some widows help files. I have the windows viewer, winhelp.exe, as does anyone with Windows. There is a freeware dos viewer (from Ziff Davis) for these and (freeware?) Mac viewer (from Microsoft). Does anyone knew of a viewer for amiga or unix ?

Patrick West
patrick.west@nwcs.org
---
* POW 1.1 On Trial * Error Backup not found: (A)bort (R)etry (P)C CARE

Document 14

Path:
news.fundp.ac.be!chaos.kulnet.kuleuven.ac.be!Belgium.EU:net!EU.net!howland.reston.ans.net!vixen.cso.uiuc.edu!news.uoregon.edu!news.bc.net!unixg.ubc.ca!vanbc.wimsey.com!fonorola!infoshare!passport.ca!news
From: Earl Curley <psychic@passport.ca>
Newsgroups: alt.hypertext
Subject: Re: *** To someone who knows their stuff! ***
Date: 11 Jul 1995 02:12:00 GMT
Organization: Psychic Quest BBS
Lines: 25
Message-ID: <3tsmlg$4ll@forged.passport.ca>
References: <950707170752.9336@news.nbnet.nb.ca>
NNTP-Posting-Host: dial047.passport.ca
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: quoted-printable
X-Mailer: Mozilla 1.1N (Macintosh; I; 68K)
To: mikehaze@nbnet.nb.ca
X-URL: news:950707170752.9336@news.nbnet.nb.ca

mikehaze@nbnet.nb.ca (Mike Hayes) wrote:
>
>To someone who knows what their talking about!
>
>Could some someone please tell what "HTML coding" is exactly (not what it
>stands for) and a brief decription of "UNIX"!
>Any help that you could give would be greatly appreciated.
>
> Mike Haze
> <mikehaze@nbnet.nb.ca>

Mike, html coding is the language used to construct a web page for the Internet. A UNIX system is what some may have refered to yea=
rs ago as a super-computer althought in todays modern technology it's simply another computer with a programming language of it's ow=
n. If you're interested in designing your own html page you can easily find many links to professional people who have spent their =
time and energy constructing html encoded pages.

Regards

Earl Curley
psychic@passport.ca
http://www.passport.ca/~psychic/

Document 15

Path:
news.fundp.ac.be!chaos.kulnet.kuleuven.ac.be!Belgium.EU.net!EU.net!news.sprintlink.net!howland.reston.ans.net!sol.ctr.columbia.edu!ucsnews!newshub.sdsu.edu!usenet
From: bostrom@mail.sdsu.edu (Tim Bostrom)
Newsgroups: alt.hypertext
Subject: Re: What does "Tweaking" a GIF mean?
Date: Mon, 10 Jul 1995 04:13:34 GMT
Organization: Network Services @ San Diego State University
Lines: 15
Message-ID: <3tq97b$eaq@pandora.sdsu.edu>
References: <3tium2$e80@malasada.lava.net>
Reply-To: bostrom@mail.sdsu.edu
NNTP-Posting-Host: annex2p31.sdsu.edu
X-Newsreader: Forte Free Agent 1.0.82

"Charles C. Goodin" <goodin@lava.net> wrote:

>Alright, I know it sounds stupid but what does "tweaking" a GIF mean? I
>figure it is more stupid not to know what it means and not ask.

>Thanks!

>Charles C. Goodin

Isn't 'tweaking' just a fancy way of saying Image Processing? I would imagine that it means resizing, cropping, and adding effects and what not. Is this wrong? Somebody please correct me if so!

bostrom@mail.sdsu.edu

Document 16

```
    Path: news.fundp.ac.be!idefix.CS.kuleuven.ac.be!Belgium.EU.net!EU.net!howland.reston.ans.net!news-
ela.megaweb.com!newstf01.news.aol.com!newsbf02.news.aol.com!not-for-mail
    From: voxtalent@aol.com (VoxTalent)
    Newsgroups: comp.text.pdf,alt.hypertext,comp.text.sgml,alt.fan.mozilla
    Subject: Re: The Crux of the Problem (was Re: PDF versus HTML)
    Date: 10 Jul 1995 00:30:54 -0400
    Organization: America Online, Inc. (1-800-827-6364)
    Lines: 12
    Sender: root@newsbf02.news.aol.com
    Message-ID: <3tqadu$jd2@newsbf02.news.aol.com>
    References: <19950708T213343Z@naggum.no>
    Reply-To: voxtalent@aol.com (VoxTalent)
    NNTP-Posting-Host: newsbf02.mail.aol.com
    Xref: news.fundp.ac.be comp.text.pdf:1325 alt.hypertext:6516 comp.text.sgml:5933 alt.fan.mozilla:588

    > I don't know any figures, but I'll bet you can find a
    > population large enough that 95% of it have never used the
    > Internet.

    Um...like, say, the Earth's? :)

    David Lawrence      <a href="mailto:voxtalent@aol.com>Comments?</a>
    <a href="http://www.access.digex.net/~oltdavid">My humble abode</a>
    ========
     And this fall from Addison-Wesley:  "Learn HTML on the Macintosh"
     by Dave Mark & David Lawrence - a great holiday stocking stuffer!

    Document 17

    Path:
news.fundp.ac.be!chaos.kulnet.kuleuven.ac.be!Belgium.EU.net!EU.net!howland.reston.ans.net!agate!news.mindlink.n
et!news
    From: "Garfield C. Johnson" <VCI@mindlink.bc.ca>
    Newsgroups: alt.hypertext
    Subject: Re: Background Colours Negative Feedback
    Date: Mon, 10 Jul 95 14:14:12 PDT
    Organization: MIND LINK! - British Columbia, Canada
    Lines: 40
    Message-ID: <NEWTNews.805411119.29794.garfield@mindlink.net.deep.mindlink.net>
    References: <DB3zD6.GFy@knoware.nl>
    NNTP-Posting-Host: line035.nwm.mindlink.net
    Mime-Version: 1.0
    Content-Type: TEXT/PLAIN; charset=US-ASCII
    X-Newsreader: NEWTNews & Chameleon -- TCP/IP for MS Windows from NetManage

    In article <DB3zD6.GFy@knoware.nl>, <martijn.de.waal@let.uva.nl> writes:

    > From: Martijn de Waal <martijn.de.waal@let.uva.nl>
    > Subject: backgroundcolors

    > Date: Sun, 2 Jul 1995 21:51:05 GMT

    > Organization: Knoware Nederland

    > html 3.0 allows you to set backgroundcolors. Does anybody know where I
    > can find a table of the hex-codes that you have to use to set the
    > background color?
    >
    > Thank you
    >
    > Martijn de Waal

    Note/??? for others out there.

    I am seeing considerable early negative comments re coloured backgrounds and
    their impact on the clarity of text etc.

    Hope about you?

    Cheers
    Garfield
    >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
    Garfield C. Johnson
    VCI@mindlink.bc.ca
    Virtual Connections Inc.
    Developers of Electronic Commerce
    "Bringing Small Businesses To The World"
    The HUMAN ELEMENT Network[tm] http://mindlink.net/vci
    "Because Results Are Really About People"
    <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

    Document 18

    Path:
news.fundp.ac.be!chaos.kulnet.kuleuven.ac.be!Belgium.EU.net!EU.net!news.sprintlink.net!news1.databank.com!usene
t
    From: michael@databank.com (mike smith)
    Newsgroups: alt.hypertext
    Subject: Re: Background Colours Negative Feedback
    Date: Tue, 11 Jul 95 02:05:13 GMT
    Organization: Databank InterNet News Host
```

```
Lines: 31
Message-ID: <3tsm8p$cvk_001@news1.databank.com>
References: <DB3zD6.GFy@knoware.nl> <NEWTNews.805411119.29794.garfield@mindlink.net.deep.mindlink.net>
NNTP-Posting-Host: vva2-ts8.databank.net
X-Newsreader: News Xpress Version 1.0 Beta #4

>Note/??? for others out there.
>
>I am seeing considerable early negative comments re coloured backgrounds and
>their impact on the clarity of text etc.
>
>Hope about you?
>
>Cheers
>Garfield
>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>
>Garfield C. Johnson
>VCI@mindlink.bc.ca
>Virtual Connections Inc.
>Developers of Electronic Commerce
>"Bringing Small Businesses To The World"
>
>The HUMAN ELEMENT Network[tm] http://mindlink.net/vci
>"Because Results Are Really About People"
><<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
>
True! People don't seem to check out their backgrounds with multiple clients
and hardware. The majority of the backgrounds seem to wind up being the prime
focus since the text is unreadable.

Cheers, Mike.
*-----------------------------------*
| http://www.charisma.com/~michael/ |
*-----------------------------------*
```

Document 19

```
Path:
news.fundp.ac.be!chaos.kulnet.kuleuven.ac.be!Belgium.EU.net!EU.net!howland.reston.ans.net!news.sprintlink.net!n
ntp.earthlink.net!tomb.earthlink.net!user
From: tomb@earthlink.net (Tom Burton)
Newsgroups: alt.hypertext
Subject: Re: Background Colours Negative Feedback
Date: Mon, 10 Jul 1995 22:08:44 -0800
Organization: Earthlink Network, Inc.
Lines: 29
Message-ID: <tomb-1007952208440001@tomb.earthlink.net>
References: <DB3zD6.GFy@knoware.nl> <NEWTNews.805411119.29794.garfield@mindlink.net.deep.mindlink.net>
NNTP-Posting-Host: tomb.earthlink.net

In article
<NEWTNews.805411119.29794.garfield@mindlink.net.deep.mindlink.net>,
"Garfield C. Johnson" <VCI@mindlink.bc.ca> wrote:

> I am seeing considerable early negative comments re coloured backgrounds and
> their impact on the clarity of text etc.
>

Ditto. Also I've seen some where the screen appears completely different
on a Mac or on one or another PC. All running Netscape 1.0! For now, I'm
avoiding using background color except in a few places (and checking those
on as many platforms as possible!).

The downside is that black-and-white line drawings show up best against a
white background, whereas tables (in Netscape) show up best against the
(default) gray.

Sigh....

Tom Burton

--
*****
Check out my WWW page at http://www.earthlink.net/free/tomb/webdocs
The opinions expressed are my own (as if anyone cares!).

My general response: "What part of "no" didn't you understand?"
*****
```

Document 20

```
Path: news.fundp.ac.be!chaos.kulnet.kuleuven.ac.be!Belgium.EU.net!EU.net!gatech!bloom-
beacon.mit.edu!news.kei.com!simtel!news.sprintlink.net!newsboy.utelfla.com!news.iag.net!usenet
From: Darrell Cadwallader <webmaster@iag.net>
Newsgroups: alt.hypertext
Subject: Re: Text colors
Date: 11 Jul 1995 15:32:19 GMT
Organization: Internet Access Group
Lines: 23
```

Message-ID: <3tu5i3$c7r@news.iag.net>
References: <3tssfj$76o@usenet.INS.CWRU.Edu>
NNTP-Posting-Host: darrell.iag.net
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
X-Mailer: Mozilla 1.1N (Windows; I; 16bit)

fjm3@po.CWRU.Edu (Fred J. Mongenel) wrote:
>
>Does anyone know if it is possible to change the color of
>text in a HTML document with a tag? I would like to have a
>dark background with white text. I have looked in help
>documents on the WEB, but haven't been able to find anything
>on a tag the controls color of text. Thanx in advance.

HoTMetaL PRO allows you to set colors for all tagged elements, but it will
not allow you to use the nifty 3.0 stuff like centering text, background
colors, font sizing, no borders on clickable links, etc.

The best way to get around this is to use all CAPS for your html tags and
space everything neatly,. i.e.

<A HREF = "mypage.html"><IMG SRC = "gifs/this-gif.gif" ALIGN = TOP></A>

<CENTER><H1><FONT SIZE = "7">T</FONT>his is the title...</H1></CENTER>

As you can see, it is very easy to distinguish the difference between tags
and text.

Document 21

Path:
news.fundp.ac.be!chaos.kulnet.kuleuven.ac.be!Belgium.EU.net!EU.net!news.sprintlink.net!cs.utexas.edu!swrinde!elroy.jpl.nasa.gov!usc!news.cerf.net!nntp-server.caltech.edu!cithe501.cithep.caltech.edu!kelsey
From: kelsey@cithe501.cithep.caltech.edu (Michael Kelsey)
Newsgroups: alt.hypertext
Subject: Re: Text colors
Date: 11 Jul 1995 16:34:36 GMT
Organization: HEP, California Institute of Technology
Lines: 78
Distribution: world
Message-ID: <3tu96s$dje@gap.cco.caltech.edu>
References: <3tssfj$76o@usenet.INS.CWRU.Edu> <3tu5i3$c7r@news.iag.net>
NNTP-Posting-Host: cithe502.cithep.caltech.edu

In article <3tu5i3$c7r@news.iag.net>, Darrell Cadwallader <webmaster@iag.net> writes:
|> fjm3@po.CWRU.Edu (Fred J. Mongenel) wrote:
|> >
|> >Does anyone know if it is possible to change the color of
|> >text in a HTML document with a tag? I would like to have a
|> >dark background with white text. I have looked in help
|> >documents on the WEB, but haven't been able to find anything
|> >on a tag the controls color of text. Thanx in advance.

That's because HTML is designed to be independent of the end user. You,
the author, can't imagine what kind of display device the ultimate user of
your information will have. They might have a monochrome monitor; they
might be using a speech synthesizer and voice recognition system because
they are blind, or work somwhere where they can't sit in front of a screen
with a mouse.

Things like text colors are, in standard HTML, configured by the end user
if she has a display device for which color is meaningful.

There are non-standard, proprietary markup languages, derived from HTML,
which allow you to try to control things like colors. Of course, if the
user of your information isn't using the "correct" vendor's browser, or if
they are using a different version of that browser, or if they have set up
their configuration files appropriately, or if they are using that browser
with a monochrome monitor, they won't see what you're trying to show them.
They'll still see the information (if you've marked it up correctly), but
they won't see the formatting details you thought you were demanding.

|> HoTMetaL PRO allows you to set colors for all tagged elements, but it will
|> not allow you to use the nifty 3.0 stuff like centering text, background
|> colors, font sizing, no borders on clickable links, etc.

You mean HoTMetaL PRO won't generate markup with the ALIGN attribute?
It won't generate the BACKGROUND attribute to BODY to allow the user to
specify a background graphic? It won't generate the <BIG> or <SMALL>
character markup containers for font sizing?

Broders on clickable links is a purely browser issue. Not all browsers are
graphical, not all of them draw borders or underlines (Arena, for example,
makes all links look like little 3D buttons).

It's true that there are some proprietary, non-standard markup languages,
based on HTML, which do these things in different ways. But why would you
expect HoTMetaL to generate non-standard, proprietary markup in place of

```
HTML, when it is trying to capture the widest possible audience, who have
user agents/browsers completely different from what the author might have?

|> The best way to get around this is to use all CAPS for your html tags and
|> space everything neatly,. i.e.

This is good advice for someone writing raw HTML.  But what does it have to
do with how that HTML, or non-standard proprietary alternatives, is
rendered on a browser?

|> <A HREF = "mypage.html"><IMG SRC = "gifs/this-gif.gif" ALIGN = TOP></A>
|>
|> <CENTER><H1><FONT SIZE = "7">T</FONT>his is the title...</H1></CENTER>
|>
|> As you can see, it is very easy to distinguish the difference between tags
|> and text.

Again, what doesn't this have to do with either rendering or marking up?
HTML, and even the proprietary alternatives to HTML, are not
case-sensitive.  The rendering will be the same whether you use upper case
or lower case, whether you put spaces around your equal signs or not (which
I find a bit difficult to read, but that's my opinion only).

Of course, the markup you've shown above is not HTML, but a proprietary
language derived from HTML and sharing a few of its features, so other
users should be very careful about using it as an example.

                                                -- Mike Kelsey
--
[ My opinions are not endorsed by SLAC, Caltech, or the US government ]
 "I've seen things you people wouldn't believe.  Attack ships on fire
  off the shoulder of Orion. I've watched C-beams glitter in the dark
  near the Tannhauser Gate.  All these memories will be lost in time,
  like tears in rain."                                  -- Roy Baty
```

Document 22

```
Path:
news.fundp.ac.be!chaos.kulnet.kuleuven.ac.be!Belgium.EU.net!EU.net!news.sprintlink.net!gatech!psinntp!psinntp!p
sinntp!news.nyc.pipeline.com!not-for-mail
From: dkolmar@grovestocktn.com (Doug Kolmar)
Newsgroups: alt.hypertext
Subject: Scripts for Windows
Date: 11 Jul 1995 14:45:54 -0400
Organization: The Pipeline
Lines: 10
Message-ID: <3tugt2$ksf@pipe2.nyc.pipeline.com>
NNTP-Posting-Host: pipe2.nyc.pipeline.com

Does anyone have suggestions on what scripting language I can use to write
CGI type scripts in a Windows environment - is there a bourne shell script
equivalent?

Your suggestions are appreciated.

Regards.
--
Doug Kolmar
dkolmar@grovestocktn.com
```

Document 23

```
Path:
news.fundp.ac.be!chaos.kulnet.kuleuven.ac.be!Belgium.EU.net!EU.net!news.sprintlink.net!uunet!in1.uu.net!newshos
t.marcam.com!zip.eecs.umich.edu!panix!not-for-mail
From: rloewy@panix.com (Ron Loewy)
Newsgroups: alt.hypertext
Subject: Re: Scripts for Windows
Date: 11 Jul 1995 17:52:17 -0400
Organization: HyperAct, Inc.
Lines: 26
Message-ID: <3turqh$93g@panix2.panix.com>
References: <3tugt2$ksf@pipe2.nyc.pipeline.com>
NNTP-Posting-Host: panix2.panix.com

In article <3tugt2$ksf@pipe2.nyc.pipeline.com>,
Doug Kolmar <dkolmar@grovestocktn.com> wrote:
>Does anyone have suggestions on what scripting language I can use to write
>CGI type scripts in a Windows environment - is there a bourne shell script
>equivalent?
>
>Your suggestions are appreciated.
>
>Regards.
>--
>Doug Kolmar
>dkolmar@grovestocktn.com

There is PERL for NT. So your scripts can be rather portable.

You can also use VB or Delphi.
```

If you are interested in Delphi components (CGI Comp. that is), you can
connect to our page at http://www.hyperact.com and choose the
"Developer's Resources" link.

Ron.
--
Ron Loewy, HyperAct, Inc. +1 (319) 351 8413 | rloewy@hyperact.com
Author of HLPDK/PA, CatMake, Interactive Help and PASTERP.
Visit our home page at http://www.hyperact.com .

Document 24

Path:
news.fundp.ac.be!chaos.kulnet.kuleuven.ac.be!Belgium.EU.net!EU.net!news.sprintlink.net!news.primenet.com!ip084
From: redt@primenet.com (Wilbur C. Tory)
Newsgroups: alt.hypertext
Subject: Convert html to text
Date: Tue, 11 Jul 95 23:43:08 GMT
Organization: Primenet
Lines: 7
Distribution: world
Message-ID: <3tvav5$ivv@dump.primenet.com>
NNTP-Posting-Host: ip084.phx.primenet.com
X-Newsreader: News Xpress Version 1.0 Beta #2

Is is possible to "strip out" the hypertext markup in a HTML document
using an editor of some type and transmit those documents to a reader
who does not use a HTML reader?

Short docs I have manually stripped out the characters, but a long document
makes this extremely difficult.

Document 25

Newsgroups: alt.hypertext
Path:
news.fundp.ac.be!chaos.kulnet.kuleuven.ac.be!Belgium.EU.net!EU.net!howland.reston.ans.net!vixen.cso.uiuc.edu!ne
ws.uoregon.edu!news.bc.net!news.mindlink.net!vanbc.wimsey.com!fonorola!news!news
From: you@somehost.somedomain (G Wheeler)
Subject: Listserv of HTML newsgroups?
Sender: news@magi.com
Message-ID: <DBL4F2.LrJ@magi.com>
Date: Wed, 12 Jul 1995 03:59:26 GMT
Nntp-Posting-Host: magi02p14.magi.com
Mime-Version: 1.0
Organization: Your Organization
X-Newsreader: WinVN 0.99.2
Lines: 4

Could someone please let me know of any listserv newsgroups for hypertext
and especially for HTML. By listserv , I mean that the articles on
HTML are sent directly to my email address.

Document 26

Path:
news.fundp.ac.be!chaos.kulnet.kuleuven.ac.be!Belgium.EU.net!EU.net!Austria.EU.net!newsfeed.ACO.net!swidir.switc
h.ch!univ-lyon1.fr!in2p3.fr!lyon.cemagref.fr!coulet
From: coulet@lyon.cemagref.fr (p. coulet)
Newsgroups: alt.hypertext
Subject: hypermedia documents design
Date: Wed, 12 Jul 1995 10:52:31 LOCAL
Organization: CEMAGREF
Lines: 20
Message-ID: <coulet.16.00134975@lyon.cemagref.fr>
NNTP-Posting-Host: pchh68.lyon.cemagref.fr
X-Newsreader: Trumpet for Windows [Version 1.0 Rev B final beta #4]

Hi every body!

I start a phd in a French Lab in the field of hypermedia
document design and I'm looking for all kind of information
concerning a methodology for the hypermedia documents
design and management.

Thanks,

Pierre COULET
coulet@lyon.cemagref.fr Cemagref de Lyon (France)
tel: (+33) 72-20-87-80 Division Hydrologie Hydraulique
fax: (+33) 78 47 78 75 U.R. Informatique

Document 27

```
    Path:
news.fundp.ac.be!chaos.kulnet.kuleuven.ac.be!Belgium.EU.net!EU.net!howland.reston.ans.net!agate!holonet!colossu
s.holonet.net!kincyb.com!kevshih
    Subject: Background Colours Ne
    Newsgroups: alt.hypertext
    From: kevshih@kincyb.com
    Date: Wed, 12 Jul 95 08:00:07 EST
    Message-ID: <7916163872601@kincyb.com>
    Organization: Hotel California
    Lines: 26

     ==> Quoting In:vci@mindlink.bc.ca to ** All ** <==-

     > html 3.0 allows you to set backgroundcolors. Does anybody know where I
     > can find a table of the hex-codes that you have to use to set the
     > background color?
     >
    check out this address, it should help you:

           http://www.infi.net/wwwimages/colorindex.html

    it lists many different colors by name with the option of webbing to another
    page for a visual listing.  it also includes the hex designations.

    Hope it helps!

    Take Care,

    kev

    ... Answers: $1, Short: $5, Correct: $25, dumb looks are still free.
    ___ Blue Wave/QWK v2.12

    --
    Hotel California, Los Angeles (310)407-1300

    Document 28

    Newsgroups: alt.hypertext
    Path:
news.fundp.ac.be!chaos.kulnet.kuleuven.ac.be!Belgium.EU.net!EU.net!news.sprintlink.net!cs.utexas.edu!utnut!nott
!cunews!superior!kmennie
    From: kmennie@superior.carleton.ca (Kia Mennie)
    Subject: Re: How to connect Newsgroups to WEB SITE
    X-Nntp-Posting-Host: superior.carleton.ca
    Message-ID: <DBptyG.49E@cunews.carleton.ca>
    Sender: news@cunews.carleton.ca (News Administrator)
    Organization: Carleton Department of Semantics and Obfuscation, Ottawa
    X-Newsreader: TIN [version 1.2 PL2]
    References: <3u2869$fgf@boris.eden.com> <3u4tlr$m6o@malasada.lava.net>
    Date: Fri, 14 Jul 1995 17:01:28 GMT
    Lines: 12

    Quoting Charles C. Goodin (goodin@lava.net):
    > <A HREF="news: alt.dreams.lucid">alt.dreams.lucid newsgroup</a>
                ^

    <nitpick>
    Don't leave a space there. I think Netscape might accept it, but Lynx
    won't.
    </nitpick>

    --
    <a href = "http://www.eng.carleton.ca/~kmennie/mennie/">.............</a>
          `I think he figured `omelette' was some sort of euphemism.'

    Document 29

    Path:
news.fundp.ac.be!chaos.kulnet.kuleuven.ac.be!Belgium.EU.net!EU.net!howland.reston.ans.net!newsjunkie.ans.net!ne
ws.lava.net!usenet
    From: "Charles C. Goodin" <goodin@lava.net>
    Newsgroups: alt.hypertext
    Subject: Re: How to connect Newsgroups to WEB SITE
    Date: 15 Jul 1995 03:55:54 GMT
    Organization: LavaNet, Inc.
    Lines: 19
    Message-ID: <3u7e8a$1a6@malasada.lava.net>
    References: <3u2869$fgf@boris.eden.com> <3u4tlr$m6o@malasada.lava.net> <DBptyG.49E@cunews.carleton.ca>
    NNTP-Posting-Host: dialup017.lava.net
    Mime-Version: 1.0
    Content-Type: text/plain; charset=us-ascii
    Content-Transfer-Encoding: 7bit
    X-Mailer: Mozilla 1.2b2 (Windows; I; 16bit)

    Thanks Kia.  I am spoiled by Netscape!

    Charles C. Goodin

    kmennie@superior.carleton.ca (Kia Mennie) wrote:
    >Quoting Charles C. Goodin (goodin@lava.net):
```

```
>> <A HREF="news: alt.dreams.lucid">alt.dreams.lucid newsgroup</a>
>
>
><nitpick>
>Don't leave a space there. I think Netscape might accept it, but Lynx
>won't.
></nitpick>
>
>--
><a href = "http://www.eng.carleton.ca/~kmennie/mennie/">.....</a>
>       `I think he figured `omelette' was some sort of euphemism.'


   Document 30

   Path:
news.fundp.ac.be!chaos.kulnet.kuleuven.ac.be!Belgium.EU.net!EU.net!news.sprintlink.net!uunet!in1.uu.net!newsfla
sh.concordia.ca!canopus.cc.umanitoba.ca!top.MTS.Net!worldlinx.com!onlink3.onlink.net!usenet
   From: bigs@onlink.net
   Newsgroups: alt.hypertext
   Subject: www homepage development
   Date: 17 Jul 1995 18:56:12 GMT
   Organization: Alternative Mediums
   Lines: 8
   Message-ID: <3ueboc$abo@onlink3.onlink.net>
   Reply-To: bigs@onlink.net
   NNTP-Posting-Host: nlkd1_onlink5.onlink.net
   X-Newsreader: AIR News 3.X (SPRY, Inc.)

   What would be the best software to write a WWW homepage.  I am new to this so I could use all the help
   that I can get..Please reply to bigs@onlink.net

   Thanks..
   :================:
   :bigs@onlink.net :
   :================:
```

La classification

Pour cette collection de documents, il y a 18 classes qui ont été découvertes par l'algorithme de *clustering*. Pour chaque classe, le nombre de documents et leur inertie sont repris ci-dessous, ainsi que la liste des documents.

```
Number of clusters : 18
```

Clusters initiaux

```
Cluster 1 : (1,0.400000) -> 11
Cluster 2 : (1,0.400000) -> 27
```

Autres clusters découverts

```
Cluster 3 : (2,0.494450) -> 1 10
Cluster 4 : (3,0.548082) -> 2 3 18
Cluster 5 : (1,0.400000) -> 4
Cluster 6 : (2,0.361609) -> 5 8
Cluster 7 : (3,0.396880) -> 6 12 15
Cluster 8 : (1,0.400000) -> 7
Cluster 9 : (2,0.463808) -> 9 25
Cluster 10 : (1,0.400000) -> 13
Cluster 11 : (1,0.400000) -> 14
Cluster 12 : (1,0.400000) -> 16
Cluster 13 : (1,0.400000) -> 17
```

```
Cluster 14 : (3,0.387158) -> 19 20 28
Cluster 15 : (2,0.498426) -> 21 22
Cluster 16 : (2,0.647437) -> 23 24
Cluster 17 : (1,0.400000) -> 26
Cluster 18 : (2,0.687909) -> 29 30
```