UNIVERSITE DE SCIENCES
SOCIALES DE GRENOBLE

UNIVERSITE CLAUDE
BERNARD , LYON I

MEMOIRE DE D.E.S.S.

EN

INFORMATION SCIENTIFIQUE
TECHNIQUE ET ECONOMIQUE

ALGORITHME DE COMPACTAGE
DES
FICHIERS INVERSE

PRESENTE

PAR

ABOUD MOHAMAD ABAD ALI

SOUS

LA DIRECTION

DE

R. CHAPUIS

JUIN 1980

SOMMAIRE

AVANT - PROPOS

Parmi les plus difficiles à résoudre, en informatique, sont les problèmes de fichiers volumineux. Cela peut être manifesté par le long temps de recherche effectuée sur les Bases de données, car il faut accèder aux informations en un temps minimal, pour cette raison les informaticiens et les chercheurs sont de plus en plus intéressés à réduire le volume de stockage des informations sans y apporter aucune réduction de pertinence.

Parmi les meilleures solutions adoptées à cet égard est la création de ce qu'on appelle les fichiers inversés (7) par lesquels on peut avoir une recherche assez efficace par rapport aux pertinence et temps d'accès, Nous travaillons dans ce sens pour réduire l'espace nécessaire à stocker ces fichiers car, en fait, pour stocker les fichiers inversés on a besoin de double l'espace de stockage réservé normalement à un fichier original (12).

C'est en étudiant ici les concepts des fichiers inversés, leurs consequence en mémoire ou sur le support de stockage que nous est apparu le moyen de définir des méthodes de compactage des fichiers inversés afin de faciliter la recherche des informations dans les Bases de données d'une manière qui rend cette recherche plus rapide.

INTRODUCTION

NOTION DU FICHIER

- Définition du fichier.

INTRODUCTION

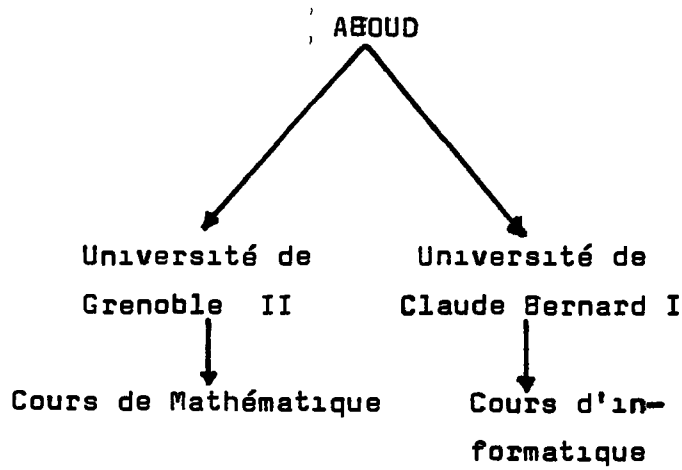
NOTION DU FICHIER

I. DEFINITION DU FICHIER

I.1 - Le fichier

Le fichier est défini comme un ensemble d'enregistrements ayant pour caractéristique des valeurs distinctes afin de prévoir que pour une caractéristique donnée, il puisse n'y avoir aucune valeur présente dans l'enregistrement, on peut imaginer une valeur blanche(10).

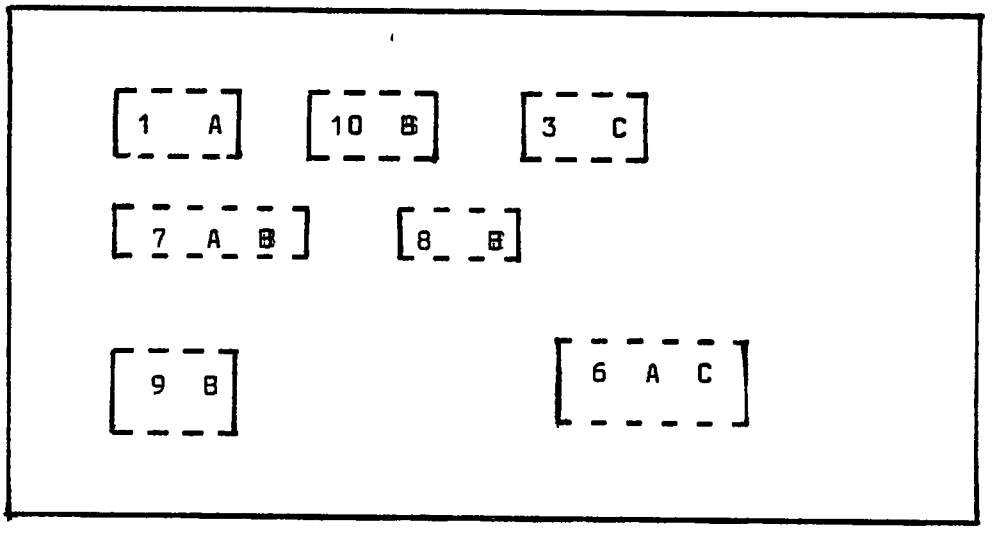
I.2 - Notion du fichier : Il existe deux types des fichiers pour nous. Il y a le fichier arborescent comme la relation qu'il existe entre l'étudiant dans un université et les matières qu'il étudies dans cette université, cette relation est bien illustré au suivant :



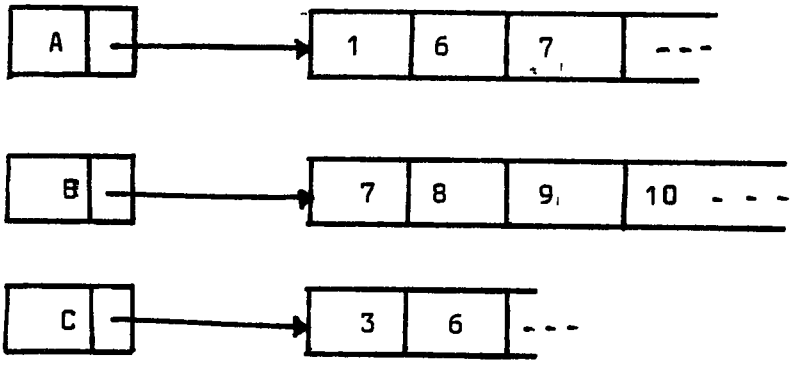
Et il y a le fichier inversé qui nous concerne, C'est un fichier organisé de façon précis pour traiter les informations contenues dans ce fichier. Il a son origine dans le fichier de base d'où il a évolué par un système spécial. Il est composé d'un répertoire des termes index qui ont pour but de décrire le contenu des documents dans le système informatisé. Il y a une correspondance entre chaque terme index et les enregistrements qui sont composés d'une liste d'adresses de ces enregistrements indexés par ces termes sur le support de stockage. Généralement, on utilise le numéro de document comme un pointeur lorsque cettes

listes ont des longueurs variables.

Le fichier original s'est présenté comme dans le schéma suivant:



Tandis que le fichier inversé correspondant est le suivant:



En général , un fichier inversé est utilisé normalement avec le fichier original, il fournie des informations utiles pour accèlerer la recherche, le fichier inversé est composé des listes inversées, c.a.d. les listes de tous les enregistrements ayant des termes index determinés.

PARTIE A

ETUDE SUR LE ROLE DE FICHIER INVERSE ET SON STOCKAGE

- Le role du fichier inversé
- Le stockage du fichier inversé

PARTIE A

ETUDE SUR LE ROLE DE FICHIER INVERSE ET SON STOCKAGE

A.1 - Le role du fichier inversé

Le role du fichier inversé est étudié ici à l'égard de traitement de l'information qui est défini selon Knuth (10) comme l'opération qui a pour but de fournir des informations aux utilisateurs d'un système d'interrogation qui permet d'utiliser les termes index accompagnés par les opérateurs booléens- ET, OU, SAUF - pour construire les questions. Pour réponder à ces questions le système compare la liste des termes index avec les termes dans les questions posées, et quand il trouve une correspondance entre les deux - c.a.d. entre la liste des termes index et les termes dans les questions posées - le système va extraire la liste concernée et la mémoriser, quand le système finit de cet opération -c.a.d. après avoir trouvé la liste correspondante aux termes index - on va éventuellement et à l'aide des programmes spécifiques avoir la liste des numéros de documents qui répondent aux questions posées par l'utilisateur.Cet opération normalement ne prend guère quelques milli-seconds, mais au fur et à mesure de nombre de termes index utilisés dans les questions, cela peut prendre du temps.

En tout cas, le pertinence joue un role important ici spécialement quand on veut réduire le volume de données, on aussi essayer de créer un système de gestion de fichiers spécialement destiné à réduire le temps d'accès aux informations (14).

En ce qui concerne la réponse offert par le système, il n'existe malheureusement pas des cas idéal - c.a.d. il y a toujours un compromis a propos de temps d'accès et de volume de stockage -ce qu'on verra plus tard, on peut constater que l'utilisation de fichiers inversés n'est pas toujours convenable, tout ça dépend de la caractéristique des données à traiter et de temps d'accés prévus par l'utilisateur du système et aussi de la pertinence exigée du système.

En bref, il y a des critères qui affectuent la performance d'organisation de fichiers qui sont:

- Le coût total de stockage
- Le temps moyen d'accès
- Pour une Base de données spécifique
    - Les caractèristiques d'interrogation
    - Spécifications du support utilisé.

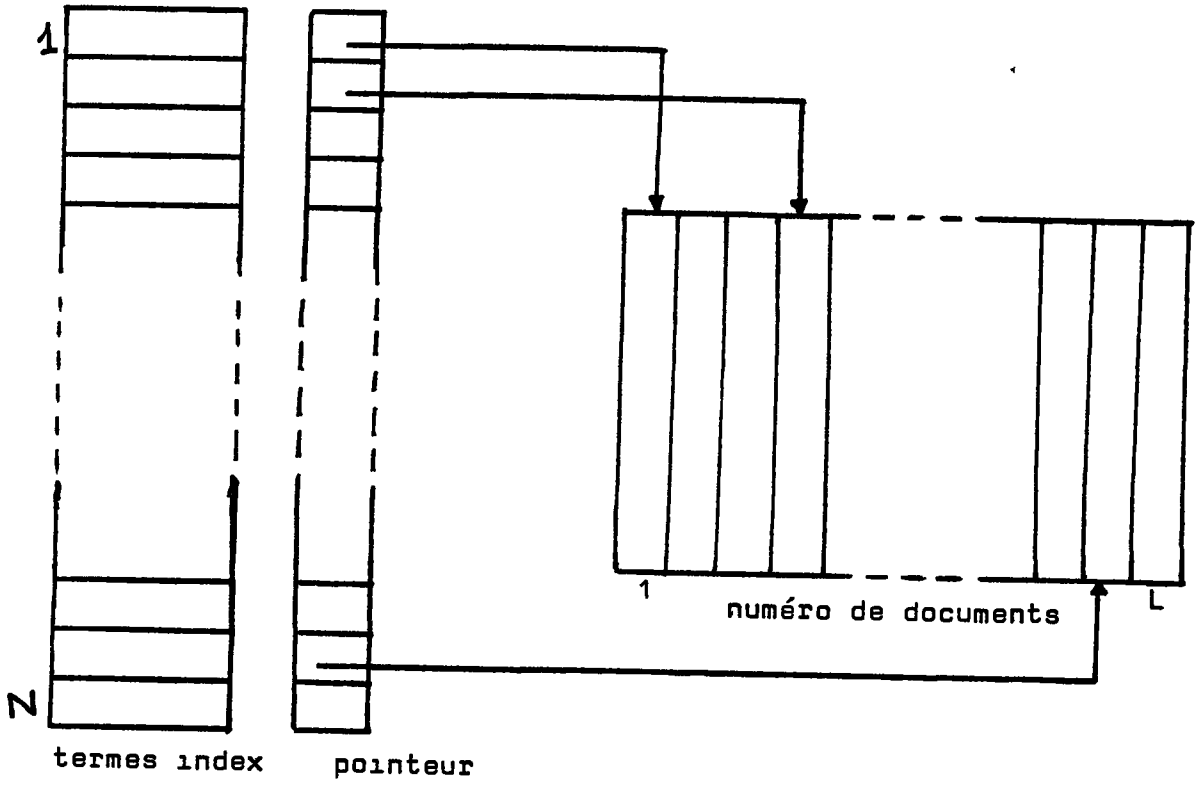
Ces critères selon Cardenas (5) sont à noter en évaluant la performance d'organisation de fichiers, il constate plus qu'en utilisant le fichier inversé on augmente la réponse aux questions des utilisateurs mais le désavantage principal est que le stockage est volumineux (4) et le coût de maintenance et mise à jour sont relativement hauts.

A.2 - Le stockage du fichier inversé

Le fichier inversé est stocké sur le support de manière qu'on peut le consulter d'une façon efficace et pour cela on distingue deux types de méthodes de stockage (12).

A.2.1 - La forme des listes des numéros de documents

Dans cette forme on va considérer le fichier inversé comme des listes des numéros de documents, supposons que nous avons N termes index et L est le nombre total des documents dans toutes les listes, comme dans le schéma suivant :



On note que ce schéma est plus facile à appliquer et à utiliser spécialement quand le fichier n'est pas de taille très volumineuse.

4.2.2 - La forme de vecteur booléen

On la nomme aussi la forme de vecteur binaire, cette forme est plus utilisée dans le cas où le fichier est très grand et afin de réduire l'espace de stockage sur le support pour les listes des numéros de documents qui sont répresentées comme un vecteur binaire selon King (9), qui donne un vecteur binaire zéros pour chaque terme index où la longueur de ce vecteur égal au nombre de documents. Si un document entre dans le système les zéros sont remplacés par les uns - c.a.d. le remplacement aura lieue dans les positions correspondants au numéro de document dans tout les vecteurs binaires de termes index qui sont en possession de ce document entré dans le système- on peut ici citer un example comme le suivant :

Termes index	Numéro de document
A, B	1
C, D, E	2
B, D, F, G	3

On a les termes index A, B, C, D, E, F, G qui ont les vecteurs binaires suivants (avant d'entrer dans le système) :

A	000 0
B	000 0
C	000 0
D	000 0
E	000 0
F	000 0
G	000 0

On aura les vecteurs binaires après entrés dans le système comme suivant :

Termes index	vecteur
A	100 0
B	101 0
C	010 0
D	011 0
E	010 0
F	001 0
G	001 0

En étudiant la table au-dessus on peut remarquer que le vecteur binaire a certains avantages par rapport à la liste de numéro de documents car ici tout les enregistrements ont une longueur fixe parce que le vecteur est égal au nombre de documents, on n'a plus besoin des pointeurs - on peut les remplacés par le calcul d'adresse, ici on a le chance de rajouter des nouveaux documents dans un espace à la fin du vecteur binaire et on peut avoir la facilité d'un seul accès aux enregistrements dans le cas où il n'y a pas de limite sur le volume de stockage sur le support.

Nous voyons que nous avons profité du fait que les opérateurs logiques peuvent se faire par une instruction machine car les ordinateurs de nos jours a le propriété de manipuler les chaines binaires efficacement.

Tout ça nous aide à compacter les données car selon King (9) la distribution d'utilisation des termes index a une fonction logarithmique, ce qui implique que plus le nombre de termes est grand plus la fréquence d'utilisation d'un terme index est faible - c.a.d. il existe beaucoup d'octets dans les enregistrements qui sont les bits zéros-.

PARTIE B

METHODES DE COMPACTAGE DE FICHIERS INVERSES

-Introduction.

-Les différentes méthodes de compactage de fichiers inversés.

PARTIE B

METHODES DE COMPACTAGE DE FICHIERS INVERSES

B.1 - Introduction

Il y a des différentes méthodes pour compacter les fichiers inversés par rapport à ses répresentations, comme nous avons déjà citer les deux formes de stockage de fichiers inversés dans partie A, nous présentons ici les deux formes :

B.1.1 - La forme des listes des numéros de documents

On peut réduire le volume des termes index par le remplacement des index matières(termes index) par certain code abrégé qui exige moins d'espace de stockage.

B.1.2 - La forme de vecteur binaire

Il existe ici des méthodes de compactage de la chaine de bits-zéros en trouvant le codage optimal pour ces bits-zéros, cette principe est plus efficace pour économiser sur le volume de stockage sur le support.

B.2 - Les différentes méthodes de compactage de fichiers inversés

B.2.1 - La méthode de Bradley (3)

Cette méthode est développée en trouvant un code optimal pour compacter les données par "run-length-code", le mot "run" désigne une chaine des bits-zéros qui sont terminés par un bit-un, la longueur de "run" est le nombre bits-zéro,"length " signifie la longueur.

La performance de ce code est ensuite comparée avec la compression théorique qui est basée sur l'information contenue dans la répartition de probabilité du "run-length", "run-length-encoding" est employé à fin de réduire le nombre de bits dans les données binaires redondants comme les images digitales des déssins d ingénieur où uneimage est répresenté par un rayon de deux dimensions en des éléments d'image noirs et blancs. Tout au long d'une certaine direction de rayon,

il apparaît des chaines ou "run" des éléments d'image voisins des mêmes valeurs noirs ou blancs, puisque l'apparition de "run" noir et blanc toujours s'alterne dans un image, dans le processus d' "encoding", il est seulement nécessaire de coder la longueur d'un "run" et non la valeur de ses éléments d'image, alors nous pouvons arbitrairement considérer le "run" typique comme un nombre de zéros suivi par un bit-un pour marquer Ie fin de "run".

Supposons que P est le nombre d'éléments d'image dans le rayon, N est le nombre de "run" dans la direction horizontale. disons que la longueur de chaque de N "run" peut être codé avec un nombre moyen de bits = L_{moy}, dans ce cas, la réduction possible peut être désignée comme un facteur de compression:

$P / N(L_{moy})$ = le facteur de compression .....(*)

alors, pour avoir un facteur de compression haut, la longueur moyenne de chaque "run" codé L_{moy} doît être au minimum.

Si la probabilité P_1 de "run" de longueur 1 est enregistrée d'après un dessin typique, on espère que le meilleur et le plus efficace code à obtenir est du premier-ordre :

$$H = \sum_i P_i \log_2 P_i$$

Disons que "run-length" s ont lieue indépendement, H diminue la limite de L_{moy} de tous les systèmes de "code" basé sur une certaine distribution de P_1, un "length-code" variabl de Huffman peut construire où $L_{moy} = H$, en tout cas, ce n'est pas une solution pratique de problème de code, lorsque l'efficacité de tels "length-codes" variabls est très sensible pour la probabilité P_1 à priori. La compression realisée sur des différentes images avec des différentes distributions de P_1 peut être très pauvre.

Nous voulons maintenant examiner un autre "length-code" variabl qui maintient son efficacité sur une distribution de probabilité, cette flexibilité peut affecter fortement par la différence entre L_{moy} et H pour ce code et pour une distribution typique de probabilités de "run-length" où le code contient 2^n entrées de longueur fixe de n. Les

entrées sont de deux types :

1. Les entrées se présentent comme zéros avec terminaison de un.

2. Les entrées se présentent comme des chaines de zéros consécutives.

Un "run" de longueueur arbitraire peut se présenter par plusieures entrées de code de type 2 et une seule entrée de type 1. Un code typique de cette forme est illustré à la table suivante:

	code index	code d'entrée	code de mot
type 1	1	1	000
	2	01	001
	3	001	010
	4	0001	011
	5	00001	100
type 2	6	5 zéros	101
	7	10 zéros	110
	8	15 zéros	111

(K = le nombre d'entrées terminées par 1).

Dans cette example la longueur de chaque code de mot est m=3, alors pour répresenter une longueur de 30, requiert 9 bits de code correspondant à 15 zéros, 10 zéros, et finalement 4 zéros plus le un de terminaison (code de mots 8,7,et 5).

Nous remarquons que K est aussi le nombre de zéros répresenté par le premier entrée de type 2, en fait les entrées de type 2 répresentent des longueurs de K,2K,3K ..... em donnant la valeur de K et n (la longueur de code de mot), on peut évaluer la longueur maximum de code par m de ces entrées comme

$$R_m = K+(2^n-K)(m-1)K \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (7*)$$

Dans cette example :

R_3 est 15 zéros + 00001 = 35.

La longueur moyenne de code L_{moy} est déterminée par peser la longueur de chaque code (n,2n,3n, ..) avec la probabilité que cette longueur est nécessaire pour coder un "run".

$$L_{moy}\ (n,k) = n\sum_{i=1}^{R_1} P_i + 2n\sum_{i=R+1}^{R_2} P_i + 3n\sum_{i=R_2+1}^{R_3} P_i \ldots\ldots (***)$$

où n est la longueur de chaque code de mot, R_m est le plus long "run" qui est répresenté par m de ces codes de mot (**), et P_i est la probabilité d'un "run" de longueur i. Si L_{moy} (n,k) est minimum par rapport à n et k, la définition optimale pour un code de cette variété sera déterminé.

Un déssin d'ingénieur typique est digitalisé comme 3000 X 4000 points donnant 12 X 10^6 éléments de déssin et 115513 "run" dans une direction horizontale, Le facteur de compression est 13,20 (*). La valeur de H est 5,91 bits donnant une compression maximum de 17,58 pour comparer, supposons qu'un "length-code" fix de $\log_2 3000 \simeq 12$ bits étaient employé pour coder chaque longueur sans regarder sa probabilité on a par (*) que le facteur de compression est cette fois = 8,65 ces résultats sont illustrés à la table suivante :
(La table illustre une comparaison de deux schémas de code avec la performance théorique optimale de tous les codes) :

	facteur de compression	L_{moy}
Théorique	17,58	5,91
k = 48, n = 6 code	13,20	7,87
12 bits, code de longueur fixe	8,65	12,00

Bien que la performance optimale de code n'atteint pas le but théorique, nous savons que cette performance maintient sa valeur pour le "run-length" avec la même distribution de probabilité autant que $\sum_{i=R_m+1}^{R_{m+1}} P_i$ reste le même au (***), L_{moy} restera constant. Lorsque k = 48 et n = 6 on a 4 valeurs de $R_m \leq 3000$ (**).

Tout ce que nous avons besoin est le fait que la probabilité cumulative (non pas les valeurs individuelles de P_i) reste presque la même entre les 5 segments de distribution définie par les 4 valeurs de R_m, ce critère est facilement obtenu, car les fac-similés du même type, comme les déssins d'ingénieur et les matérielles écrites par la machine à écrire, ont une distribution de même forme générale.

Tellement nous avons une méthode pour définir les paramètres de code de longueur variable basée sur une distribution empirique des longueurs de "run", il est certain que la performance de ce code ne sera pas dérangéepar des petits changements dans la fréquence des longueurs de "run".

B.2.2 - La méthode de King (9)

Par sa méthode King a suggeré que le vecteur binaire est un alternative par rapport au stockage de pointeurs de documents dans un fichier inversé, le fichier de vecteur binaire peut fournir, en économisant l'espace de stockage, de temps d'accès minimal et moins d'effort de programmation.

Il a commencé par définir les termes suivants
Sous-vecteur: est une chaine d'octets dans le vecteur binaire,
Sous-vecteur zéro : est un sous-vecteur dont chaque octet contient 8 bits-zéros ,
Sous-vecteur non-zéro : est un sous-vecteur dont chaque octet contient au moins un bit-un.
(Ici on a l'octet qui contient 8 bits c.a.d. on ne peut pas stocker plus de 255 entièrs,de 0 jusqu'à 255.

L'étapes suivantes sont envisagées par King dans sa méthode de compactage :

1 - Le vecteur binaire est partagé entre des séries des sous-vecteurs zéros et non-zéros, ils peuvent avoir une longueur maximale de 255 octets. Les sous-vecteurs de zéro qui sont plus longs que 255 octets, le 256ème octet sera considérer comme non-zéro octet de tels façon qu'on peut diviser le sous-vecteur zéro qui est long.

2 - Chaque sous-vecteur non-zéro est préfixé par deux octets. Le premier contient le nombre d'octet-zéro précedent le sous-vecteur non-zéro de vecteur non compact. Le deuxième octet contient le nombre d'octets dans le sous-vecteur non-zéro.

3 - Le vecteur compact donc comprend seulement les sous-vecteurs non-zéros avec leurs préfixes.

4 - Le vecteur compact sera terminé par un champs de deux octets de zéro binaire.

On peut donner un example en considérant le vecteur binaire suivant :

01100000 / 10000000 / sept octets zéro / 00000001 / 10000000 / ....

les "slashes" indiquent la division du vecteur en octets,ce vecteur remplace la liste de numéros 2, 3, 9, 80 et 81 dans un fichier inversé

standard où chaque numéro de document prend 3 octets de stockage, 15 octets sera nécessaire pour stocker ces numéros, le vecteur compact qui en résulte est le suivant :

00000000/00000010/01100000/10000000/00000111/00000010/00000001/10000000/...

On considére chaque octet dans un vecteur être numéroté séquentiellement à partir d'octet un à gauche , dans le vecteur non compact l'octet un et deux forment un sous-vecteur non-zéro, par conséquence, les premier quatre octets dans le vecteur compact peuvent être considérés comme suivant :

Octet un. Zéro binaire indiquant qu'il n'y a pas d'octet zéro qui précedent ce sous-vecteur .

Octet deux. deux binaire signifie que le sous-vecteur non-zéro suivant à la longueur de 2 octets.

Octets trois, quatre. Octet un et deux du vecteur original.
Octets trois à neuf du vecteur original sont un sous-vecteur zéro, et octets dix et onze sont le deuxième sous-vecteur non-zéro. Donc les 4 deuxième octets du vecteur compact sont intérpretés comme suivant :

Octet cinq. sept binaire indique qu'un sous-vecteur zéro de sept octets a été compacté.

Octet six. deux binaire indique que le deux octets suivants sont un sous-vecteur non-zéro.

Octet sept, huit. Octets dix et onze du vecteur original

Donc le vecteur binaire est réduit de onze octets à huit et le space nécessaire pour enregistrer les numéros de documents dans le fichier inversé standard reste comme 15 octets

Pour examiner plus les différences entre les deux fichiers on a utilisé l'ordinateur de IBM 360/67 et les programmes écrites en PL/1 sur une Base de données où 6.121 documents sont indexés par 1.484 termes index, le nombre total de numéros des documents était 94.542 donnant um moyen d'indexation de 15, termes par document, la taille des enregistrements physiques du fichier de

la liste était 331 octets, le fichier avait besoin de 702.713 octets. Pour le fichier de vecteur binaire non compact, la taille d'enregistrements physique était 1.286 octets pour stocker un nombre maximum de 10.216 documents et quand l'algorithme de compactage était appliqué avec une longueur d'enregistrement physique de 130 octets, l'espace de stockage pour le fichier de vecteur binaire était réduit à 281.450 octets ou 41 % de l'espace nécessaire pour le fichier inversé de la liste.

Des séries de 40 recherches de complexité variantes étaient dirigées avec les deux fichiers, le fonction de "time" de PL/1 a donné la possibilité d'avoir des statistiques de temps, King a trouvé pour 22 questions que le temps de recherche du fichier de vecteur binaire était plus rapide dont une est un facteur de 35, et pour 18 questions où le temps d'accès pour la recherche du fichier standard était au plus 6,17 fois plus rapide. Il a aussi noté que le temps total de recherche au cas du fichier de vecteur binaire était de 0,79 secondes à 9,72 secondes et au cas du fichier standard il était de 0,15 secondes à 202,98 secondes c.a.d. le fichier de vecteur binaire est plus efficace pour "On-line" recherch de type interactif et généralement plus rapide, King a aussi remarqué que le temps de recherche au cas du fichier de vecteur binaire a une relation avec le nombre des termes index dans le question, mais au cas du fichier inversé standard, le temps de recherche a une relation avec le nombre total des numéros de documents dans les termes index trouvés dans la question.

P (la probabilité de bit zéro)

technique	0,05	0,05	0,75	0,90	0,95	0,99
Run-length coding	0,53	0,87	1,15	1,98	2,90	11,24
Golomb	0,35	0,62	1,07	1,88	3,36	12,25
WH1	0,94	0,94	0,94	1,34	1,93	5,38
WH1M	0,94	0,94	0,94	1,69	2,69	9,80
WH2	1,68	0,87	0,88	1,47	2,63	9,52
Thiel et Heaps	0,94	0,94	0,94	1,24	2,04	7,37
King	0,94	0,94	0,93	1,01	1,52	5,13

D'après cette table ; on remarque que le gain de WH2 est supérieur à un lorsque la valeur de P est petit. Les deux premiers codages (c.a.d. celles de "run-length-coding" et de Golomb) donnent des bons résultats quand la valeur de P est grand, Ces propriétés sont combinées dans WH1M et WH2 qui nous semblent être bon pour toutes valeurs de P (la probabilité de bit zéro) .

Nous pouvons aussi constaté que la valeur de gain augmente au cas d'avoir des valeurs hautes de P en presque toute les techniques présentées .

B.2.3 Les autres méthodes

Il y a des autres méthodes pour compacter les fichiers inversés, parmi eux on trouve les méthodes suivantes :

- le codage de Schuegraf ou ce qu'on appelle "run-length-coding".
- le codage de Wedekind et Hârder ou ce qu'on appelle en un mot (WH1).
- le modification de codage de WH1 c.a.d. WH1M.
- le codage de Wedskind et Hârder en deux-mot (WH2) .
- le codage de Thiel et Heaps .

Les techniques mentionnées au-dessus sont baeées sur un modéle stochastique de n états (cas) différents . C'était expliqué en utilisation ds chaine de Markov pour créer le vecteur binaire ; on compare les gains qui résulte de compacter dams chaque tschmique entre eux, le gein est exprimé comme :

$$\frac{\text{vecteur binaire non compact}}{\text{vecteur binaire compact}}$$

On peut constater que les gains sont une fomction de la probabilité de bit zéro . Ils existent aussi des compareisons en considérant Ia longueur optimale de bits nécessaire pour exprimer le code, donc om peut montrer Iss gains en supposant que n = 1 et le longueur du code est égal à k bits .

Dams la table suivant on voit cette comparaison :

CONCLUSION

Dans les systèmes d'interrogation, les techniques différentes que nous avons déjà présentées ici, jouent un role très important par rapport au temps d'accès aux enregistrements pour la recherche interactive et rétrospective spécialement dans le cas où l'on interroge une grande Base de données, la rapidité de réponse dépend fortement sur le volume de stockage de données car quand on a les moyens pour réduire ce volume, on aura certainement la possibilité de diminuer le temps de recherche qui compt beaucoup spécialement quand on interroge une grande Base de données de nos jours .

Les fichiers inversés ne sont pas toujours convenables que les autres types de fichiers comme par example le fichier d'arbre de double-chaine ou le fichier de "multiliste" . On considère aussi le degré de complexité de la question posée au système d'interrogation selon Cardenas (5), ainsi il y a une méthodologie, un modèle et un système programmé pour évaluer à priori les coûts totals de stockage de données et aussi le temps moyen d'accès de plusieures organisations des fichiers par rapport à une Base de données spécifique, il faut aussi prendre compte des caractéristiques des questions et spécifications de dispositif . quand on considère tout ça on peut choisir le structure de son fichier par rapport aussi à ce qu'on veut de son système d'interrogation .

Les fichiers inversés possèdent une inconvenance que le nombre des numéros des documents correspondant aux termes index n'est pas toujours le même et la solution dans ce cas-là n'est pas facilement disponible, car en appliquant la méthode de "fragmentation équifrequente" (13) qui signifie que les termes index soient être découpés en plusieurs éléments pour avoir une combinaison de ces éléments entre eux au lieue d'utiliser les termes index , on ne peut jamais

être certain de ce que le résultat de sa recherche va lui donner, tout ça dépend sur ce qu'on a déjà citer à notre conclusion.

Notre étude des méthodes de compactage des fichiers inversé était plus ou moins concentrée sur les techniques de Bradley (3) et de King (9), car Bradley a dans son traitement expliqué et en grande profondeur l'idée et l'application de son compactage, autrement dit il a donné une application assez riche pour illustrer son intérpretation de son application de déssin d'ingénieur, il en a transformé en poits digitals, ceci nous aide dans l'application de sa méthode de compactage avec une grande rigueur, éliminant, de ce fait tout risque de malapplication.

Quant à la méthode de King (9) , nous la trouvons très intéressante en ce qui concerne son élaboration sur son traitement du compactage de fichier inversé qui éventuellement nous permet de comparer les critères de la performance de son compactage en économisant d'espace de stockage et par conséquence sur la rapidité de recherche réalisée .

Enfin , nous regrettons de ne pas être toujours très récents en ce qui concerne les références utilisées à cause de difficultés de trouver les références souhaitées qui n'étaient pas disponibles au temps de recherche sur ce mémoire .

BIBLIOGRAPHIE

(1) ABOUD M. A.
Some particular methods for the solution of P.D.E.
Mémoire, Université de Technologie, Bagdad, Dec. 1977.

(2) BAHL L. R., KOBAYASHI H.
Image Data Compression by Predictive Coding II, Encoding Algorithms- IBM. J. of Research Developement. 1974, 18, PP. 172-179.

(3) BRADLEY S. D.
Optimizing a scheme for run length encoding- Proceedings of the IEEE, January, 1969, PP. 108-109.

(4) CARDENAS A. F.
Analysis and performance of inverted Data Base structures.
Comm. ACM. 1975, Volume 18, NO. 5, PP. 253-263.

(5) CARDENAS A. F.
Evaluation and Selection of file organization- a Model and System. Comm. ACM., 1973, Volume 16, NO. 9, PP. 540-263.

(6) CHABRE-PECCOUD M.
Le projet abrégé.
Thése, Université de Grenoble III.

(7) DIMSDALE J. J. and Heaps H. S.
File structure for an On-Line catalog of one million Titles, J. of Library Automation. 1973, Volume 6, No. 1 (march), PP. 37-55.

(8) GOLOMB S. W.
Run-length encoding, IEEE Trans. of Information Theory. IT-12, 1966, PP.399-401.

(9) KING D. R.
The binary vector as the basis of an inverted index file, J. of Library Automation, 1974, Volume 7, No. 4; PP.307-314.

(10) KNUTH D. E.
The art of computer programming.
Volume 1, Fundamental Algorithms,
Volume 3, Sorting and Searching.
Addisson-Wesley Publishing Company, Reading Massachusetts USA, 1973.

(11) Lyon J. K.
An Introduction to Data Base.
Wesley Inter-Science, New-York, USA, 1971.

(12) SCHUEGRAF E. J.
Compression of large inverted files with hyperbolic term distribution. Information processing and management, Volume 12, 1976, PP. 377-384.

(13) SCHUEGRAF E. J. and HEAPS H. S.
Selection of equifrequent word fragments for information retrieval. Information Storage & retrieval. 1973, Volume 9, P. 697.

(14) STELLHORN W. H.
An Inverted file Processor for Information Retrieval.
IEEE Trans. on Computers, Volume 26, No. 12, 1977,
PP. 1258-1267.