Université d'Ottawa
CSI 2531 – Examen final (version francophone)
Professeur: Iluju Kiringa

15 avril 2003
9:h30-12h30
Durée: 3 hs

Livre fermé et sans calculatrices

Nom de famille: _____

Prénom: _____

Numéro d'étudiant: _____

Il y a 23 questions et un total de 100 points.

Cet examen doit contenir 27 pages,
incluant cette page couverture.

1 – à 2 – Disques magnétiques	/ 4
3 – Bandes magnétiques	/ 2
4 – Mémoires tampon ("Buffering")	/ 2
5 – Compression Huffman	/ 2
6 – Hachage: distribution des enregistrements	/ 2
7 – à 8 - Récuperer l'espace dans les fichiers	/ 2
9 – Traitement coséquentiel	/ 2
10 – à 11 - Tri de larges fichiers	/ 4
12 – Propriétés des arbres B	/ 2
13 – Propriétés des arbres B+	/ 2
14 – Effacements dans les arbres B	/ 2
15 – Hachage	/ 8
16 – Hachage extensible: insertions	/ 8
17 – Hachage extensible: effacements	/ 8
18 – Arbres B: insertions	/ 8
19 – Arbres B+ préfixe simple	/ 4
20 – Arbre B+: effacements	/ 8
21 – Indexes secondaires: listes inversées	/ 6
22 – Hachage: insertion (pseudocode)	/ 12
23 – Programme de décompression Lempel-Ziv	/ 12
Total	/ 100

... suite

1 à 2 – Disques magnétiques — 4 points

Considerez un disque magnétique avec les caracteristiques suivantes:

```
Nombre d'octets par secteur = 512
Nombre de secteurs par piste = 100
Nombre de pistes par cylindre = 8
Nombre de cylindres = 100
```

ainsi que un fichier contenant 8,000 enregistrements, chacun ayant 256 octets.

1 Disques - 1ère partie

Combier de cylindres sont requis pour stocker le fichier sur disque en supposant que le disque est vide au départ et est utilisé de manière optimale?

A. 5

B. 10

C. 15

D. 20

E. 40

2 Disques - 2ème partie

Supposez que ayions la même organisation que dans la question précédente et que les enregistrements sont écrits séquentiellement dans un disque initiallement vide en remplissant les pistes d'un cylindre avant d'aller au cylindre suivant.
Combien de recherches sur disque ("seeks") sont-elles nécessaires afin de récuperer (dans l'ordre) le 5ème, 50ème, 500ème, 2500ème, 2800ème et 3000ème enregistrement?

A. 2

B. 3

C. 4

D. 5

E. 6

... suite

3 Bandes magnétiques — 2 points

Soit une bande magnétique avec les caracteristiques suivantes:

- densité: 2000 bpi (bytes per inch – octets par pouce)
- espace inter-bloc: 0.2 in (pouce)
- vitesse = 100 ips (inches per second – pouces par seconde)

Supposez que l'on veut stocker un fichier de 10,000,000 enregistrements en utilisant un facteur de bloc de 1,000. Chaque enregistrement a 100 octets.
Combien d'espace est nécessaire pour stocker ce fichier entièrement?
Quel est le taux de transmission nominal de la bande?

A. Espace: 700 pouces; taux de transmission nominal : 400,000 octets/sec

B. Espace: 5,020 pouces; taux de transmission nominal : 20,000,000 octets/sec

C. Espace: 502,000 pouces; taux de transmission nominal : 400,000 octets/sec

D. Espace: 50,200 pouces; taux de transmission nominal : 200,000 octets/sec

E. Espace: 502,000 pouces; taux de transmission nominal : 200,000 octets/sec

4 Mémoires tampon ("Buffering") — 2 points

Considerez les stratégies de mémoire tampon ("buffering") vues en classe.
Laquelle des affirmations suivantes est **fausse**?

A. **"Buffer pooling"** implique une réserve ("pool") de mémoires tampon disponibles de laquelle des mémoires tampon sont puisées selon les besoins.

B. **"Move mode"** implique un déplacement de données entre une mémoire tampon d'entrée/sortie et une mémoire tampon d'un programme.

C. **"Locate mode"** elimine le besoin de transferrer des données entre une mémoire tampon d'entrée/sortie et une mémoire tampon d'un programme.

D. Quand **"double buffering"** est utilisé, deux mémoires tampon sont impliquées et le système opère sur une mémoire tampon pendant que l'autre est entrain d'être remplie ou vidée.

E. Dans **"move mode"** et **"locate mode"**, un programme est capable d'opérer directement sur les donnés dans la mémoire tampon d'entré/sortie.

... suite

5 Compression Huffman — 2 points

Il y a plusieurs arbres de Huffman valides pour le même fichier si l'on considère des choix arbitraires des sous-arbres gauches et droits. Chacun de ces choix conduit à un code Huffman différent.

Marquez ci-bas la réponse qui **ne correspond pas** à un code Huffman valide pour le fichier contenant les caractères suivants: `aabbbc`

A. `a:10, b:0, c:11`

B. `a:00, b:1, c:01`

C. `a:11, b:0, c:10`

D. `a:10, b:11, c:0`

E. `a:01, b:1, c:00`

... suite

6 Hachage: distribution des enregistrements — 2 points

Soit r le nombre de clés devant être stockées dans un tableau de hachage avec N adresses.

Soit $p(i)$ la probabilité que une adresse donnée ait un nombre i d'enregistrements.

Le tableau suivant montre une approximation de $p(i)$ pour diverses valeurs de r/N utilisant la distribution de Poisson:

r/N	0.50	0.75	1.00	1.50
p(0)	0.61	0.48	0.37	0.22
p(1)	0.30	0.35	0.37	0.33
p(2)	0.08	0.13	0.18	0.25
p(3)	0.01	0.03	0.06	0.13
p(4)	-	0.01	0.02	0.07

Considerez les deux scénarios suivants:

(1) $r = 1000$, $N = 2000$, sans buckets

(2) $r = 1000$, $N = 1000$, avec des buckets de taille 2

Calculez le **nombre attendu d'enregistrements débordants** (i.e. le nombre d'enregistrements qui devront être stockés en dehors de leurs adresses domiciliaires) pour chacun des deux scénarios ci-dessus:

A. (1) 100 (2) 75

B. (1) 180 (2) 80

C. (1) 180 (2) 360

D. (1) 200 (2) 100

E. (1) 200 (2) 150

... suite

7 à 8 - Récuperer l'espace dans les fichiers — 2 points

Considerez le fichier suivant qui stocke de l'information au sujet des présidents des universités canadiennes; par exemple, Edwards est le président de l'Université d'Ottawa (UofO). Le fichier est stocké dans un format à **longueur variable** utilisant un indicateur de longueur. Il y a 5 enregistrements dans ce fichier:

```
         LH ---> -1
Record#: 0              1            2            3                  4
         13Edwards|UofO|11Bates|UofT|11Wills|UofA|16Masteriano|UofW|12Chavez|UofM|
```

`LH` est un pointeur vers le premier élément de la liste `AVAIL LIST`. La valeur -1 indique la fin de la liste. Une barre verticale | est un séparateur de champ. L'indicateur de longueur occupe 2 octets stockés au début du fichier. Ainsi, l'enregistrement 0 occupe 13 octets, sans compter l'indicateur de longueur lui-même. Nous comptons le décalage en octets à partir de 0; par exemple, la lettre "U" dans l'enregistrement 1 a un décalage en octets de 23. Enfin, nous organisons la liste `AVAIL LIST` comme vu en classe pour un format variable.

7 Récuperer de l'espace dans les fichiers – Question I

Dans cette question, les pointillés indiquent des caractères effacés; on place "*" dans le premier champ de l'enregistrement effacé; et le décalage en octets du prochain enregistrement disponible est placé dans le second champ.

Après l'effacement des enregistrements 1 (Bates) et 3 (Masteriano), notre fichier sera:

A.

```
LH ---> 41
0              1            2            3                4
13Edwards|UofO|11*.....|-1..11Wills|UofA|16*.........|15..|12Chavez|UofM|
```

B.

```
LH ---> 3
0              1            2            3                4
13Edwards|UofO|11*.....|-1..11Wills|UofA|16*.........|15..|12Chavez|UofM|
```

C.

```
LH ---> 41
0              1            2            3                4
13Edwards|UofO|11*.....|15..11Wills|UofA|16*.........|-1..|12Chavez|UofM|
```

D.

```
LH ---> 15
0              1            2            3                4
13Edwards|UofO|11*.....|-1..11Wills|UofA|16*.........|15..|12Chavez|UofM|
```

E.

```
LH ---> -1
0              1            2            3                4
13Edwards|UofO|11*.....|41..11Wills|UofA|16*.........|15..|12Chavez|UofM|
```

... suite

8 Récuperer l'espace dans les fichiers – Question II

Supposez la même organisation de fichier que dans la question précédente. Laquelle des affirmations suivantes est **fausse**?

A. Nous ne pouvons pas utiliser le RRN ("Relative Record Number") à la place du décalage en octets pour acceder directement aux enregistrements.

B. Pendant l'insertion d'un nouvel enregistrement, le premier élément libre (celui vers lequel pointe `LH`) doit être utilisé.

C. Utiliser "Worst Fit" peut conduire à de la fragmentation.

D. Nous pouvons utiliser "Best Fit" comme stratégie de remplacement pour le fichier.

E. Nous pouvons utiliser la stratégie "Worst Fit".

9 Traitement coséquentiel — 2 points

Combien de **comparaisons de clés** et d'**opérations d'écriture** sont requises pour faire l'**appariment** ("match") des noms dans les deux listes suivantes, en supposant que les listes d'entrée sont stockées dans des fichiers sur disque?

```
List1:  Banana, Eliza, Finamo, Kant, Ludwig, Pietro, Zaraeu
List2:  Anastasios, Balancius, Furlan, Zoro
```

A. nombre de comparaisons = 28 nombre d'opérations d'écriture = 10

B. nombre de comparaisons = 10 nombre d'opérations d'écriture = 11

C. nombre de comparaisons = 10 nombre d'opérations d'écriture = 0

D. nombre de comparaisons = 28 nombre d'opérations d'écriture = 7

E. nombre de comparaisons = 6 nombre d'opérations d'écriture = 0

... suite

10 à 11 - Tri de larges fichiers — 4 points

Soit

- un fichier avec 1,000,000,000 enregistrements, chacun ayant 1,000 octets,
- 500,000,000 octets de mémoire principal disponible.

Supposez que vous voulez trier ce fichier en utilisant une procédure de fusion qui divise le fichier en "runs" et fusionne ces runs de manière co-séquentielle.

10 Tri de larges fichiers – Question I

En combien de runs devez vous diviser le fichier?

A. 20,000

B. 2

C. 10,000

D. 200

E. 2,000

11 Tri de larges fichiers – Question II

Combien d'enregistrements peuvent simultanément tenir en mémoire pendant l'étape de fusion de l'algorithme?

A. 50,000

B. 500,000

C. 500,000,000

D. 1,000

E. Aucune des réponses ci-haut n'est correcte

... suite

12 Propriétés des arbres B — 2 points

Considerez un fichier index de 2,000,000 clés. Quel est le nombre maximum de niveaux ("levels") que l'index en arbre B d'ordre 20 aura?

A. 7

B. 8

C. 2

D. 100

E. Aucun des nombres ci-haut

13 Propriétés des arbres B+ — 2 points

Laquelle des affirmations suivantes au sujet des arbres B+ est **fausse**?

A. Un arbre B+ consiste en un "index set" et un "sequence set".

B. Un arbre B+ consiste en un "index set" organisé comme un arbre B.

C. Un arbre B+ préfixe simple utilise les plus petits séparateurs possibles.

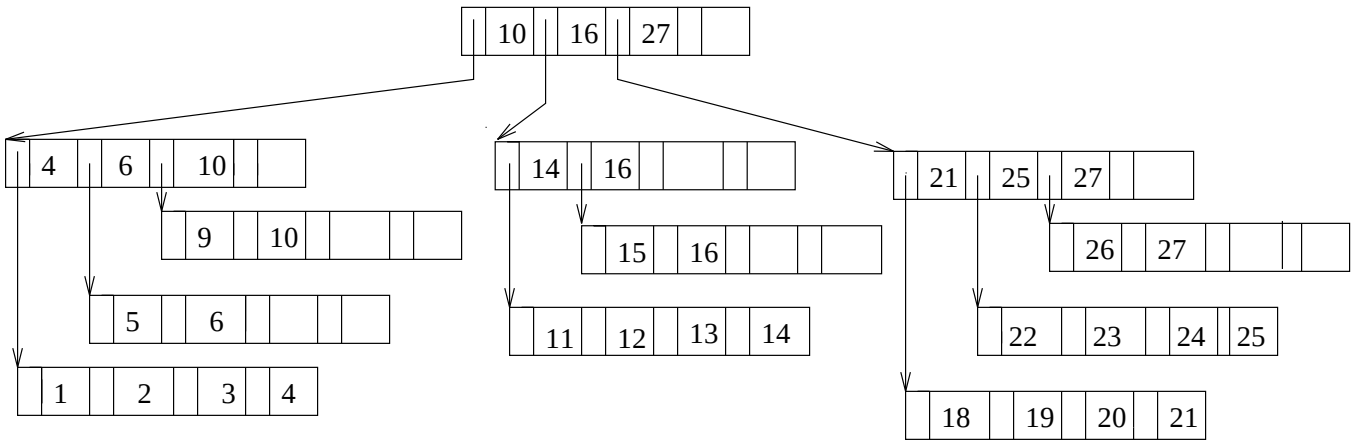
D. Les séparateurs sont dérivés à partir des clés des enregistrements situés de part et d'autre des frontières des blocs du "sequence set".

E. L'"index set" d'un arbre B+ préfixe simple est un arbre dont les noeuds sont des séparateurs.

... suite

14 Effacements dans les arbres B — 2 points

Considerez l'arbre B (d'ordre 4) suivant:



Considerez l'effacement des clés 9, 16 et 1 (dans cet ordre).

Determinez si les affirmations suivantes sont VRAIES ou FAUSSES:

(1) Après l'effacement de la clé 9, deux noeuds sont fusionnés.

(2) Après l'effacement de 16 et 1, la structure de l'arbre change.

(3) Après l'effacement de 9, 16 et 1 (dans cet ordre), l'arbre contient toujours 3 niveaux.

Les quelles des affirmations ci-dessus sont VRAIES?

A. (1) seulement

B. (2) seulement

C. (1) et (2) seulement

D. Toutes sont vraies

E. (1) et (3) seulement

... suite

15 Hachage — 8 points

Considerez les paires (clé/valeur de hachage) suivantes:

clé	A	D	B	E	G	H	F
valeur de hachage (adresse domiciliaire)	7	6	7	5	4	4	4

Dans les prochaines 4 parties de cette question, vous allez montrer la table de hachage ("hash table") qui résulte de l'insertion des clés ci-dessus dans l'ordre indiqué.

Chaque table de hachage a 8 adresses; la première adresse est 0 et la dernière est 7.

Dans chaque partie, une organisation de hachage différente est requise. Nous ne dessinons (partiellement) la table que pour la partie A.

Partie A — 2 points Hachage: débordement progressif ("progressive overflow")
Dans cette partie, utilisez le débordement progressif ("linear probing") sans buckets:

0	
1	
2	
3	
4	
5	
6	
7	

Partie B — 2 points Hachage: débordement progressif avec buckets
Ici, utilisez le débordement progressif ("linear probing") et des **buckets de taille 2**.
Le nombre d'adresse de hachage est toujours 8 (la première adresse est 0 et la dernière est 7).

... suite

Partie C — 2 points Hachage: débordement progressif chainé ("chained progressive overflow")
Ici, utilisez le débordement progressif chainé sans buckets.

Partie D — 2 points Hachage: "scatter tables"
Ici, utilisez une "scatter table".

... suite

16 Hachage extensible: insertions — 8 points

Effectuez les insertions suivantes (dans l'ordre donné ci-bas) dans une table de hachage vide au départ.

clé: k	$h(k)$	représentation binaire de $h(k)$ (avant l'inversion des bits)
A	0	0000
D	2	0010
B	1	0001
E	5	0101
G	1	0001
H	1	0001
F	5	0101
K	6	0110
L	2	0010

Vous devez utiliser des buckets de taille 3.

Vous pouvez utiliser le verso de cette page pour faire un brouillon.
Montrez ci-bas la structure de la table de hachage extensible après chaque séquence de 3 insertions, c'est-à-dire: après B, après H et après L.

1) La table de hachage extensible après l'insertion de A, D, B:

... suite

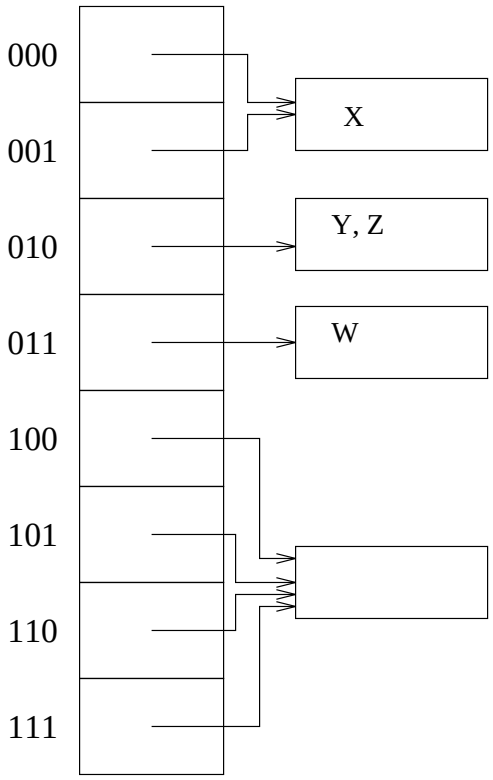
2) La table de hachage extensible après l'insertion de E, G, H:

3) La table de hachage extensible après l'insertion de F, K, L:

... suite

17 Hachage extensible: effacements — 8 points

Considerez la table de hachage extensible suivante:



Vous devez utiliser des buckets de taille 2.

Dessinez une table de hachage extensible après chacune des effacements suivants:

A. Effacer Z

B. Effacer W

Pour chaque effacement, **montrez seulement l'étape finale ainsi que chacune des étapes intermédiaires où la table de hachage ("directory") change de taille**.

... suite

... suite

18 Arbres B: insertions — 8 points

Considerez la séquence de clés suivante:

```
3, 19, 4, 20, 1, 13, 16, 6, 2, 23, 14, 7, 21, 18
```

Construisez un arbre B d'ordre 4 par des insertions successives de clés dans l'ordre donné ci-haut.

Vous pouvez utiliser le verso de cette page-ci comme brouillon et veuillez ne montrer ci-bas (et à la page suivante) que les étapes intermédiaires suivantes:

- une étape d'insertion de votre choix qui cause une simple division;
- une étape d'insertion de votre choix qui cause une division récursive qui va jusqu'à causer la division de la racine;
- l'arbre final obtenu.

1) Division simple choisie:

... suite

2) Division recursive choisie causant une division de la racine:

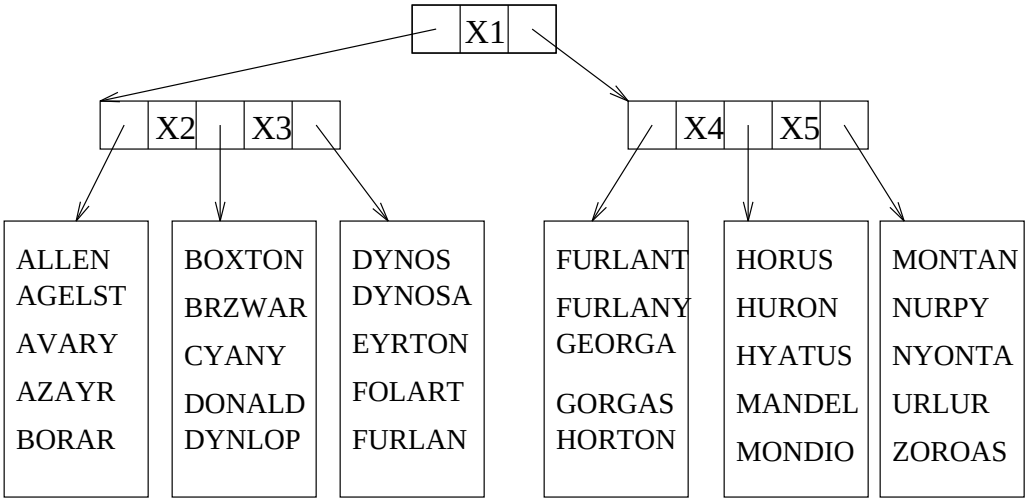
3) Arbre B final:

... suite

19 Arbres B+ préfixe simple — 4 points

Considerez l'arbre B+ préfixe simple suivant qui contient des clés pour une portion d'étudiants du programme CSI de SITE.

Le "sequence set" contient les clés en blocs de taille 5. L"'index set" en arbre B+ est d'ordre 3.



Donnez les valeurs des plus petits séparateurs qui sont représentées par les inconnues suivantes:

X1=

X2=

X3=

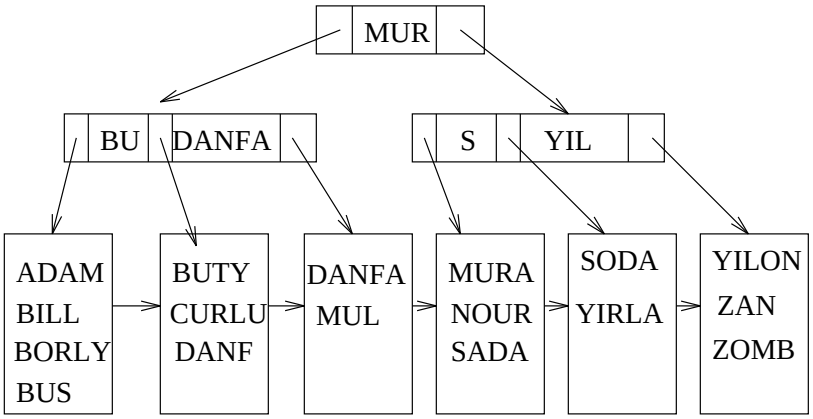
X4=

X5=

... suite

20 Arbre B+: effacements — 8 points

Considerez l'arbre B+ préfixe simple donné ci-bas et dont l'"index set" est d'**ordre 3** et le "sequence set" a des **blocs de taille 4**.



Dans cette question, en cas de choix à faire entre la fusion et la redistribution, vous devez toujours **donner priorité à la fusion**.

Utilisez l'espace ci-bas pour montrer comment l'arbre B+ préfixe simple donné change après avoir effacé
`DANFA, SODA, MURA, NOUR, ZAN, ZOMB, BORLY,` dans cet ordre.

Vous n'avez besoin de ne montrer l'abre en question que après l'effacement de `NOUR` et `BORLY`.

L'arbre B+ préfixe simple donné après l'effacement de `NOUR`:

... suite

L'arbre B+ préfixe simple donné après l'effacement de `BORLY`:

... suite

21 Indexes secondaires: listes inversées — 6 points

Supposez que nous voulons administrer la collection suivante d'enregistrements d'étudiants:

```
2087743 Jenkins   Francesca Ottawa
2251841 Duran     Diana     Toronto
2173098 McKone    Ryan      Toronto
2240890 Adelstein Katharine Quebec
2180384 Choi      Maria     Shawinigan
2231301 Hemmings  Anne      Montreal
2225639 Taylor    Sarah     Toronto
2225657 Jeppesen  Abigale   Ottawa
```

Les champs du fichier de donnés ci-haut contiennent respectivement: le numéro d'identification, le nom de famille, le prénom ainsi que la ville d'origine.

Veuillez trouver ci-bas un fichier d'index primaire ainsi que un fichier d'indexe secondaire organisé en utilisant la méthode des listes inversées ("inverted lists"):

Index primaire

	clé primaire	réf. vers les données
0	2087743	0
1	2173098	3
2	2180384	4
3	2225639	6
4	2225657	7
5	2231301	5
6	2240890	3
7	2251841	2

Index secondaire

	clé secondaire	réf. vers `Id List`
0	Montreal	5
1	Ottawa	0
2	Quebec	3
3	Shawinigan	4
4	Toronto	2

Fichier `Id List`

0	2087743	7
1	2251841	2
2	2173098	6
3	2240890	-1
4	2180384	-1
5	2231301	-1
6	2225639	-1
7	2225657	-1

Chaque élément de l'**index secondaire** pointe vers une liste de clés primaires qui lui correspondent et qui sont stockées dans le fichier **Id List**. Notez que chacun de ces index est trié.

Montrez le contenu des 3 fichiers après l'insertion (dans l'ordre donné) des 3 enregistrements suivants dans le fichier des données:

```
2351741 Malik     Anna      Toronto
2246757 Jorg      Derek     Toronto
1987744 Laberge   Jean      Gatineau
```

... suite

Contenu des 3 fichiers après les 3 insertions données ci-dessus:

Index primaire

	clé primaire	réf. vers les données
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		

Index secondaire

	clé secondaire	réf. vers `Id List`
0		
1		
2		
3		
4		
5		
6		
7		

Fichier `Id List`

0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		

... suite

22 Hachage: insertion (pseudocode) — 12 points

Donnez le pseudocode pour l'**insertion** dans un fichier de hachage utilisant le débordement progressif (aussi apellé "linear probing"). Le fichier de hachage peut contenir des pierres tombales ("tombstones"), à savoir, une indication d'un enregistrement qui a existé mais qui n'existe plus.

Vous pouvez utiliser les suggestions suivantes pour votre pseudocode:

- Un enregistrement du fichier de hachage contient 2 champs, à savoir la `clé` (entier), et la `référence` (entier).
  La `clé` contient:

  un entier positif, si elle représente une clé réelle;

  `0`, si cette position est libre;

  `-1`, si la position contient une pierre tombale.

- Votre pseudocode doit donner des détails au sujet des opérations de manipulation de fichier de bas niveau. Par exemple:
  `Ouvrir le fichier logique Y en mode lecture et l'associer avec le fichier physique X;`
  `Positionner le pointeur de lecture du fichier Z au décalage octet N;`
  `Lire (clé,référence) du fichier W;` (la lecture est effectué à partir de la position de lecture courante)
  etc.

- La donnée à inserer dans le fichier de hachage est `(newkey, newreference)`.

Completez le pseudocode qui se trouve à la page suivante.

... suite

```
######################### PSEUDOCODE: ############################
Variables globales et constantes a utiliser:
   HASHSIZE: nombre d'adresses de hachage
   ``hashindex.txt'': fichier physique contenant un nombre HASHSIZE
                      d'enregistrements (RRNs de 0 a HASHSIZE-1)
   RECSIZE: taille en octets de chaque enregistrement de ``hashindex.txt''
   hashfile: nom logique a associer a ``hashindex.txt''

int HashFunction(int key); // retourne l'adresse domiciliaire de la cle ``key''

int HashSearch(int key);
             // retourne le RRN de la position de la cle ``key'' dans le
             // fichier ``hashfile'', si ``key'' est presente, ou -1 sinon

procedure HashInsert(int newkey, int newreference)
{
   Ouvrir hashfile en mode lecture & ecriture, en l'associant a ``hashindex.txt''









    Fermer hashfile;
}
```

... suite

23 Programme de décompression Lempel-Ziv — 12 points

Ecrivez un programme C++ qui, étant donné un fichier comprimé en Lempel-Ziv, produit le fichier **décomprimé**. Notez bien que vous devez DECOMPRIMER, et non comprimer.

Rappelez vous que le fichier comprimé consiste en une séquence de plusieurs paires (nombre, lettre). Vous n'avez pas besoin de programmer les détails sur la manière d'obtenir ces paires. Vous n'avez qu'à utiliser la procédure `ReadPairsFromFileToArrays` qui lit les paires (nombre, lettre) du fichier d'entrée vers des tableaux `numbers` and `letters`. Utilisez le verso de cette page-ci si nécessaire.

```
#include <fstream>
#include <iostream>
using namespace std;
#define MAXSIZE 32512

void ReadPairsFromFileToArrays(istream & inputfile, int numbers[], char letters[]);
    // Cette procedure vous est donnee (utilisez la sans l'implementer)

int main() {
   istream input; ostream output; // Fichiers d'entree et sortie
   int numbers[MAXSIZE]; char letters[MAXSIZE]; // Tableaux avec l'info comprimee
   input.open(``compressed.txt'',ios::in|ios::binary); // Ouvrir le fichier comprime
   ReadPairsFromFileToArrays(input,numbers,letters); // Lire les paires









 }
```

... End Of Examen final (version francophone)