GIS3 — Structures de Données — Épreuve d'Avril 2014
Durée 2 heures — Aucun document autorisé

---

1 Exercices et questions de cours (10 points)

Question 1 [1 pt]. L'algorithme A a une complexité en temps en $O(n)$. L'algorithme B a une complexité en temps en $\Omega(n^2)$. Peut-on affirmer que, quand n tend vers l'infini, A est plus rapide que B ?

1.1 Listes chaînées

On considère les déclarations C suivantes :

```
struct maillon {
    double valeur;
    struct maillon* next;
};
#define NIL (struct maillon*)0
struct liste {
    int nbelem;
    struct maillon* tete;
};
```

Question 2 [2 pts]. Écrire une fonction C paramétrée par une liste L, un double d et qui ajoute d en queue de L.

Question 3 [1 pt]. On souhaite que le temps d'exécution de l'ajout en queue ne dépende pas du nombre de maillons présents dans la liste. Comment modifier l'implantation des listes pour obtenir ce résultat (donner la structure C avec une phrase d'explication) ?

Question 4 [1 pt]. Quelle structure de données étudiée en cours pourrait tirer bénéfice de la modification ci-dessus ?

1.2 Tables de hachage

On considère l'insertion des clés $\square$, $\bigcirc$, $\triangle, *$ dans une table de hachage de $N = 12$ alvéoles. La table est gérée avec la technique du double hachage. Les valeurs de hachage des différents éléments sont les suivantes :

	h_1	h_2
$\square$	3	5
$\bigcirc$	7	7
$\triangle$	7	9
$*$	4	3

Question 5 [1 pt]. Insérer les clés dans la table. Donner un schéma de la table résultat. Indiquer les éventuelles collisions.

Question 6 [1 pt]. La fonction h_2 retourne un nombre impair dans tous les cas. Est-ce suffisant pour trouver un alvéole libre s'il en existe au moins un dans la table ?

1.3 Files de priorité

On considère le tableau $T = \{8, 10, 17, 31, 19, 80\}$.

Question 7 [1 pt]. Donner une représentation graphique de ce minimier.

Question 8 [1 pt]. Défiler un élément à partir du minimier de la question précédente. Quel élément sort ? Dessiner le minimier obtenu.

Question 9 [1 pt]. Enfiler 12. Dessiner le minimier obtenu.

2 Problème (10 points)

Les questions s'appliquent au document préparatoire joint ci-dessous.

Soit A une matrice creuse de dimension $m \times n$, contenant p coefficients non nuls.

On s'intéresse pour commencer au format DOK, implanté avec un arbre binaire de recherche. Les valeurs des nœuds (les coefficients de A) doivent être du type suivant.

```
struct coeff {
    int i; /* indice de ligne */
    int j; /* indice de colonne */
    double aij; /* valeur de la matrice, ligne i et colonne j */
};
```

La matrice elle-même est représentée par un pointeur du type `struct abr_matrix*`. On ne stocke pas dans la structure les entiers m, n et p.

Question 10 [1 pt]. Donner la déclaration du type `struct abr_matrix`.

Pour convertir une matrice du format `struct abr_matrix` vers le format de Yale, on utilise une variante de l'algorithme qui affiche les éléments d'un ABR par ordre croissant. On souhaite donc que les coefficients de A soient affichés par ligne croissante et, pour une même ligne, par colonne croissante. Il faut donc choisir avec soin le test qui indique si un coefficient (i, j, a_{ij}) est inférieur à un coefficient $(i', j', a_{i'j'})$.

Question 11 [1 pt]. Écrire une fonction C `est_inferieur`, paramétrée par deux `struct coeff` a et b, qui retourne `true` si $a < b$ et `false` sinon (voir paragraphe ci-dessus).

Question 12 [1 pt]. Donner la déclaration d'une structure `struct Yale_matrix` permettant de représenter une matrice A de dimension $m \times n$ suivant le format de Yale. La structure doit comporter les dimensions m et n de la matrice, le nombre p d'éléments non nuls ainsi que les trois tableaux V, I et J. Les tableaux doivent pouvoir être alloués dynamiquement.

On s'intéresse maintenant à la fonction suivante, qui initialise Y avec ses autres paramètres, m, n, p et B. C'est un constructeur pour le type `struct Yale_matrix`, ce qui implique qu'aucun champ de Y n'est initialisé.

```
void init_Yale_abr_matrix
        (struct Yale_matrix* Y, int m, int n, int p, struct abr_matrix* B)
```

La partie délicate consiste à remplir les trois tableaux à partir de B.

On conseille d'écrire une fonction auxiliaire, récursive, qui implante une variante de l'algorithme qui affiche les éléments d'un ABR par ordre croissant. Comme cette fonction est conçue pour énumérer les coefficients dans l'ordre de remplissage des tableaux, il est assez facile de remplir V et J.

Pour le tableau I, on conseille d'utiliser la fonction récursive pour *compter* le nombre d'éléments non nuls de chaque ligne. Ainsi, après exécution de la fonction récursive, $I[i+1]$ est égal au nombre d'éléments non nuls de la ligne i, pour tout $0 \leq i < m$. Sur l'exemple, on obtiendrait :

$$I = [0,\ 2,\ 2,\ 3,\ 1]\,.$$

Un simple balayage de I suffit alors pour terminer l'initialisation et obtenir, sur l'exemple :

$$I = [0,\ 2,\ 4,\ 7,\ 8]\,.$$

Question 13 [3 pts]. Écrire le constructeur.

On s'intéresse maintenant au format DOK, implanté par table de hachage. Une fois la table remplie, on souhaite, ici aussi, pouvoir énumérer les coefficients de la matrice A dans l'ordre de remplissage des tableaux du format de Yale.

Parmi les différentes variantes d'implantation (hachage double, hachage simple avec listes désordonnées, avec listes triées), certaines rendent cette énumération plus facile que d'autres.

Question 14 [2 pts]. Quelle variante vous paraît la meilleure ? Les fonctions de hachage doivent-elles satisfaire des contraintes particulières ? Préciser les détails du pseudo-code ci-dessous, qui énumère les coefficients de la matrice dans l'ordre de remplissage des tableaux du format de Yale :

for i, indice de ligne, variant de 0 à $m-1$ do
  [code qui énumère les `struct coeff` de la ligne i par colonne croissante]
end do

Question 15 [2 pts]. Donner la déclaration du type `struct hash_matrix` correspondant à votre structure de données. Le tableau des éléments de la table de hachage peut être conçu de taille fixe, en supposant qu'on dispose d'une borne `PMAX` sur p. Spécifier cette structure de données.

La représentation la plus simple d'une matrice $m \times n$ est un tableau à deux dimensions. Dans de nombreuses applications, les matrices sont *creuses*, c'est-à-dire, qu'elles sont très grandes mais n'ont que très peu d'éléments non nuls. On s'intéresse à des structures de données (des *formats*) qui représentent des matrices en ne stockant en mémoire que les éléments non nuls.

Il y a deux grands groupes de formats : ceux pour lesquels il est facile d'ajouter un nouvel élément mais où les parcours (nécessaires pour implanter les algorithmes d'algèbre linéaire) sont compliqués ; ceux pour lesquels l'ajout d'un nouvel élément est compliqué mais où les parcours sont plus simples.

Typiquement, les bibliothèques numériques dédiées aux matrices creuses utilisent un format du premier groupe pour construire la matrice, puis convertissent la matrice dans un format du deuxième groupe avant d'exécuter les algorithmes d'algèbre linéaire. Voir [1].

Le format DOK fait partie du premier groupe : on stocke les éléments non nuls a_{ij} d'une matrice A dans un dictionnaire (table de hachage, arbre binaire de recherche ...). La clé est formée du couple d'indices (i, j).

Le format de Yale fait partie du deuxième groupe. Trois tableaux sont utilisés pour représenter une matrice A. On l'illustre sur l'exemple ci-dessous (remarque : les indices de A commencent à 0 et pas à 1) :

$$A = \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 10 & 20 & 0 & 0 & 0 & 0 \\ 0 & 30 & 0 & 40 & 0 & 0 \\ 0 & 0 & 50 & 60 & 70 & 0 \\ 0 & 0 & 0 & 0 & 0 & 80 \end{pmatrix} \end{array} . \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array}$$

Notons p le nombre d'éléments non nuls (ici, $p = 8$). Le premier tableau, V, de dimension p, contient la suite des valeurs non nulles, énumérées par ligne. Sur l'exemple,

$$V = [10, 20, 30, 40, 50, 60, 70, 80]\,.$$

Le deuxième tableau, I, est de dimension $m+1$. Pour tout indice de ligne $0 \leq i < m$, l'entier $I[i]$ donne l'indice, dans V, du premier élément non nul de la ligne i de A. Le dernier élément de I est égal à p. La différence $I[i+1] - I[i]$ entre deux éléments consécutifs de I donne le nombre d'éléments non nuls de la $i$ème ligne de A. Sur l'exemple,

$$V = \begin{array}{ccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ [10, & 20, & 30, & 40, & 50, & 60, & 70, & 80] & \end{array}$$

$$I = [0, 2, 4, 7, 8]$$

Enfin, le troisième tableau, J, est de dimension p. Pour tout indice $0 \leq k < p$, l'entier $J[k]$ donne l'indice de colonne du réel $V[k]$. Sur l'exemple,

$$J = [\underline{0,\ 1},\ \underline{1,\ 3},\ \underline{2,\ 3,\ 4},\ \underline{5}]$$

Il y a $I[1] - I[0] = 2$ éléments non nuls sur la ligne 0. Ils sont sur les colonnes 0 et 1

Il y a $I[2] - I[1] = 2$ éléments non nuls sur la ligne 1, situés sur les colonnes 1 et 3

Références

[1] Wikipedia. Sparse matrix. `http://en.wikipedia.org/wiki/Sparse_matrix`, 2004.