Examen juin 2010 - 2nde session

Durée : 2H

Documents autorisés: supports et notes manuscrites de cours et de TD/TME
– Barême indicatif –

Note importante : Les questions peuvent toutes être traitées indépendamment les unes des autres, au besoin en s'appuyant sur l'existence des fonctions des questions précédentes.

Exercice 1 (4 points) – Question de cours

Soit la séquence d'entiers suivante : $\mathbf{3}, \mathbf{10}, \mathbf{12}, \mathbf{8}, \mathbf{4}, \mathbf{1}$.

Q 1.1 Arbre Binaire de Recherche
• En supposant que les entiers sont insérés dans l'ordre de la séquence, dessiner l'arbre binaire de recherche issu de cette séquence (il n'est pas demandé d'équilibrer l'arbre).
• Equilibrer l'arbre obtenu à la question précédente en proposant une rotation ou une double rotation permettant d'atteindre un ABR équilibré. Dessinez le ou les arbres correspondants.

Q 1.2 Tas
• En supposant que les entiers sont insérés dans l'ordre de la séquence dans un tas manipulant un minimum, dessiner ce tas.
• Dessiner le tas obtenu après la suppression du minimum dans le tas de la question précédente.

Exercice 2 (2 points) – Utilisation de structures

On désire implémenter une structure de données abstraite nommée `table` (ou `map`) qui est définie comme un ensemble de valeurs indexées par une chaîne de caractères que l'on nomme alors `clé`. On suppose ici que la clé permet d'identifier un élément de manière unique (on dit que la clé est discriminante). Ainsi, connaissant la clé, il est possible de retrouver l'élément. Une `table` peut ainsi facilement représenter un annuaire. Dans ce cas, le tableau suivant donne toutes les informations sur la table :

Clé	Infos	
Dupont	4, place Jussieu	0685674539
Wuillemin	18, rue Sébastien	0456379413
Segura	98, place Monge	0145683498

Q 2.1 Indiquer quelle structure de données permet d'implémenter efficacement cette structure de données abstraites `table`. Justifier votre réponse.

Q 2.2 Est-ce que le fait que la clé soit discriminante permet une gestion particulière de cette structure ? Dans le cas où la clé serait non-discriminante, votre proposition reste-t-elle valable ? efficace ?

Q 2.3 L'annuaire devient très volumineux avec beaucoup de personnes de même nom. Est-ce que votre réponse aux questions précédentes reste pertinente ?

Exercice 3 (6 points) – Multiplication et Matrices Creuses

Dans ce qui suit, nous supposerons la valeur N définie comme suit :

```
1 #define  N 50
```

Soit $\mathcal{M}_N$ l'ensemble des matrices carrées (N, N).
$\forall A \in \mathcal{M}_N$ on indique par $A[i, j]$ l'élément (i, j) (avec $i, j \in \{0, \cdots, N-1\}$).

Q 3.1 Beaucoup de représentations d'une matrice de $\mathcal{M}_N$ sont possibles en C. En voici deux parmi les plus classiques :

```
1 /* implementation A*/
2 typedef float Matrice[N][N];
```

```
1 /* implementation B*/
2 typedef float Matrice[N*N];
```

Quelle est la différence en terme d'implémentation ? En supposant qu'un `float` prend 8 octets en mémoire et un pointeur 4 octets, quelle est la taille d'un objet de ces 2 types ? Qu'en concluez-vous ?

Q 3.2 Afin de ne plus avoir à se poser de questions sur l'implémentation, on propose de mettre en place une couche d'abstraction sous la forme de 2 fonctions :

```
1 float get(Matrice m,int i,int j); /* retourne la valeur de la matrice m en [i,j] */
2 void set(Matrice m,int i,int j,float f); /* affecte f dans la matrice m en [i,j] */
```

- Pourquoi la matrice n'est-elle pas passée par pointeur dans les deux fonctions ?
- Proposer un code pour ces 2 fonctions et pour chacun des deux types d'implémentations A et B (il y a donc au total 4 petites fonctions à écrire).

Q 3.3 On rappelle que le produit matriciel entre $A \in \mathcal{M}_N$ et $B \in \mathcal{M}_N$ s'écrit :

$$(A \times B)[i, j] = \sum_{k=0}^{N-1} A[i, k] \cdot B[k, j]$$

Écrire la fonction `mult(Matrice A,Matrice B,Matrice C)` qui calcule $C = A \times B$. Quelle est sa complexité (en fonction de N) ?

Q 3.4 On a souvent besoin de représenter des matrices dites 'creuses', c'est-à-dire des matrices dont les valeurs non nulles sont très rares. Ce qui implique une représentation classique en mémoire remplie d'un très grand nombre de zéros. Afin d'éviter une telle consommation de mémoire, on représente plutôt les matrices creuses sous la forme de dictionnaire de valeurs. Ainsi une matrice telle que :

$$\begin{vmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 3 \end{vmatrix}$$

serait représentée plus efficacement par la liste :

$$\{(0, 4) \Rightarrow 1; (2, 0) \Rightarrow 1; (2, 2) \Rightarrow 1; (3, 3) \Rightarrow 2; (4, 4) \Rightarrow 3\}$$

- Quelle structure de données proposeriez-vous pour représenter une telle matrice creuse (le temps d'accès à un élément est TRÈS important) ? Quelle est la complexité en mémoire de cette représentation ?

- Pour ce qui est de programmer la multiplication, une information importante (encore plus que le temps d'accès à un élément) serait de pouvoir connaître les colonnes ou les lignes complètement vides. Comment pourriez-vous modifier votre structure de données pour prendre en compte cette demande ?

Exercice 4 (8 points) – Détection de circuit

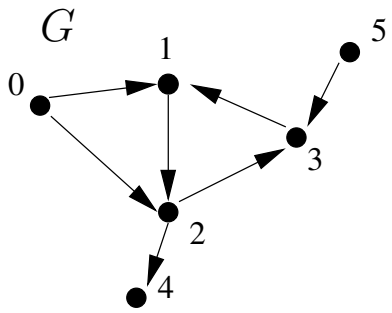
Le but de cet exercice est d'implanter un algorithme de détection de circuits dans un graphe orienté différent de celui vu en TD.

Soit $G = (S, A)$ un graphe orienté où V est un ensemble de n sommets et A un ensemble de m arcs. S'il existe un arc (i, j) dans A, on dit que i est *prédécesseur* de j et que j est *successeur* de i. On représente en mémoire un tel graphe par un tableau de listes d'adjacence qui est implémenté par les déclarations suivantes :

```
1  typedef struct arc {
2    int i ;     // sommet i de l'arc (i,j)
3    int j ;     // sommet j de l'arc (i,j)
4  } Arc ;
5
6  typedef struct elementListeA {
7      Arc a ;
8      struct elementListeA * suiv ;
9  } ElementListeA ;
10
11 typedef struct sommet {
12     int num;
13   ElementListeA *LA; // liste des aretes incidentes a ce sommet
14 } Sommet ;
15
16 typedef struct graphe {
17     int n ; // nb de sommets
18     Sommet* tabS ; // tableau de sommets
19 } Graphe ;
```

On considère qu'un graphe a déjà été alloué et affecté à des valeurs. Dans ce graphe les n sommets sont numérotés de 0 à $n-1$ et sont stockés respectivement dans les cases de 0 à $n-1$ du tableau `tabS`.

Q 4.1 Considérons le graphe G suivant :



Donner unr représentation de la structure de données correspondant à ce graphe.

Q 4.2 On veut connaître le nombre de prédécesseurs de chaque sommet. Pour cela, on définit un tableau de n entiers `t_nbpred` initialisé avec les valeurs 0. Ecrire une fonction `void compte_nbpred(Graphe G, int* t_nbpred) ;` qui affecte à chaque case i le nombre de prédécesseurs du sommet i.

Q 4.3 Soit $T \subset S$ un sous-ensemble de sommets (cela peut-être, par exemple, tout S ou un seul sommet). Un *sous-graphe induit* par T est le graphe obtenu à partir de G en ne gardant que les sommets de T et uniquement les arcs entre deux sommets de T. Ainsi, pour un sommet i de T, on compte

uniquement les prédécesseurs de i qui sont dans T.
On code T par un tableau de n sommets tel que $T[i]$ vaut 1 si le sommet est dans T et vaut 0 si le sommet n'est pas contenu dans T.
On suppose dans cette question que, pour tout sommet i de T, $t\_nbpred[i]$ contient le nombre des prédécesseurs de i qui sont dans T et, pour tous les autres sommets, $t\_nbpred[i]$ contient une valeur quelconque. Donner une fonction `int rech_sanspred(Graphe G, int* T, int* t_nbpred) ;` qui retourne le numéro d'un sommet de T qui n'a pas de prédecesseur dans T, s'il en existe un, et -1 s'il n'existe pas un tel prédécesseur.

Q 4.4 Soit $T \subset S$ et le tableau `t_nbpred` du nombre de prédécesseurs des sommets de T. On veut supprimer un sommet u de T, c'est-à-dire effectuer l'instruction $T[u] = 0$. Donner la fonction `void maj_nbpred(Graphe G, int* T, int* t_nbpred, int u) ;` qui effectue la mise à jour du tableau `t_nbpred` pour l'ensemble $T \setminus \{u\}$.

Q 4.5 L'idée de l'algorithme de détection d'un circuit dans un graphe repose sur les deux propriétés suivantes :
Proposition 1 : Si un graphe est sans circuit, alors il contient un sommet sans prédécesseur.
Proposition 2 : Soit $G = (S, A)$ un graphe (avec ou sans circuit) et u un sommet sans prédécesseur de G. Alors G contient un circuit si et seulement si le sous-graphe induit par $S\{u\}$ contient un circuit.
En utilisant ces deux propositions et les 3 fonctions vues dans les questions précédentes, proposer une fonction `int detect_circuit(Graphe G) ;` qui retourne vrai si et seulement si G contient un circuit.

Q 4.6 (Bonus) Donner la complexité de la fonction `compte_nbpred` en fonction du nombre d'arcs dans le graphe.
Au total des appels de la fonction `maj_nbpred`, combien de fois (en fonction du nombre d'arcs) sont appelées les lignes à l'intérieur de la boucle contenue dans la fonction `maj_nbpred` ?
Quelle est la complexité de la fonction `rech_sanspred` et combien de fois est-elle utilisée ?
En déduire la complexité de la fonction `detect_circuit`.
Proposer une structure de données pour améliorer la complexité de `detect_circuit` ? Quelle complexité obtiendrait-on alors ?