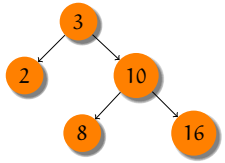
Examen - 1ère session

Durée : 2H00

Seul document autorisé : une feuille A4 manuscrite
– Barème indicatif –

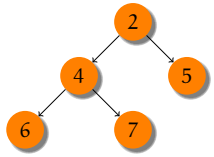
Exercice 1 (4 points) – Questions de cours : tas, ABR et AVL

Q 1.1 Voici un arbre :



Est-ce un tas ? un ABR ? un AVL ? Justifiez votre réponse.

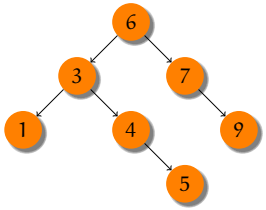
Q 1.2 Voici un tas représenté sous la forme d'un arbre :



Donnez sa représentation sous forme de tableau. Donnez les formules permettant de retrouver les positions du père, du fils gauche et du fils droit d'un noeud du tas.

Q 1.3 Après suppression de l'élément 2, quel tas obtient-on ? Décrivez les étapes.

Q 1.4 Voici un AVL :



Après suppression de l'élément 9, quel AVL obtient-on ? Décrivez les étapes.

Exercice 2 (4 points) – Modélisation : gestion de commandes

On souhaite mettre en place une structure de données efficace pour la gestion de commandes. Les opérations à réaliser sont les suivantes :

- Insérer une nouvelle commande.
- Retrouver une commande à partir du nom du client et de la date de la commande.

- Afficher les données d'un client (nom, adresse, téléphone).
- Afficher toutes les commandes d'un client, triées par ordre chronologique.

On désire que ces opérations soient **les plus rapides possibles** en sachant que :

- il n'y a pas d'homonymes (c-à-d il n'y a pas deux clients différents avec le même nom).
- les commandes ne sont pas insérées dans la base par ordre chronologique, car elles arrivent par plusieurs sources (courrier, mail ou téléphone).

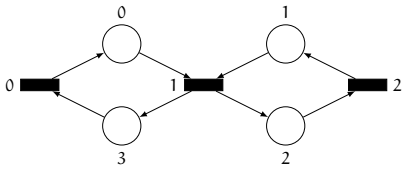
Q 2.1 Décrivez, sans donner le code C, une structure de données pour cette gestion de commandes. Justifiez votre réponse en donnant la complexité des quatre opérations.

Indication : il faut utiliser au moins 2 structures de données classiques liées entre elles.

Exercice 3 (12 points) – Problème : réseau de Pétri et graphe de marquage

Réseau de Pétri

Un réseau de Pétri est un graphe orienté $R = (S, A)$ particulier permettant de représenter le comportement d'un système possédant plusieurs états. C'est le cas par exemple d'une usine où la chaîne de production change d'état en fonction des pièces qui circulent d'une machine-outil à une autre. Le dessin suivant représente un réseau de Pétri.



L'ensemble des sommets S d'un réseau de Pétri se divise en deux types de sommets :

- n_1 *sommets-place* qui correspondent à des variables décrivant les états,
- n_2 *sommets-transition* correspondant aux étapes permettant de passer d'un état à un autre.

Sur le dessin, il y a 4 sommets-place représentés par des cercles (numérotés de 0 à 3) et 3 sommets-transition représentés par des rectangles noires (numérotés de 0 à 2). Il n'existe que deux types d'arcs dans l'ensemble A :

- des arcs allant d'un sommet-place à un sommet-transition, appelés *arcs transition*,
- des arcs allant d'un sommet-transition à un sommet-place, appelés *arcs non-transition*.

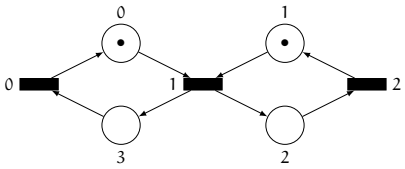
Comme les réseaux de Pétri ne contiennent qu'un nombre très petit de sommets et d'arcs, on décide ici d'implémenter un réseau de Pétri selon le principe de matrice d'adjacence. Comme il existe deux types d'arcs, on va utiliser deux matrices : une matrice $n_1 \times n_2$ correspondant aux arcs transition et une matrice $n_2 \times n_1$ correspondant aux arcs non-transition. Plus précisément, on va travailler avec la structure suivante :

```
1 typedef struct{
2   int n1;  /* Nb de sommets-place */
3   int n2;  /* Nb de sommets-transition */
4   int** Arc_trans; /* Matrice d'adjacence des arcs transition */
5   int** Arc_nontrans;  /*Matrice d'adjacence des arcs non transition*/
6 } Rpet;
```

Q 3.1 Donnez la fonction `void alloue_init_Rpet(Rpet** R, int n1, int n2);` permettant d'allouer et d'initialiser un réseau de Pétri ayant n_1 sommets-place et n_2 sommets-transition et ne contenant pas d'arcs.

Q 3.2 Donnez la fonction `void ajoute_arc(Rpet *R, int trans, int u, int v);` qui ajoute un arc (u,v) à la structure `Rpet` : si `trans` vaut 1, c'est un arc transition et sinon, c'est un arc non-transition.

Pour définir un réseau de Pétri, il faut ajouter un *marquage*. On appelle *marquage* un tableau M de n_1 entiers tel que `M[i]` correspond au nombre de jetons contenus dans le sommet-place numéro `i`. Sur le dessin suivant, on montre le marquage $M_0 = [1,1,0,0]$ où les jetons sont représentés par des petits ronds noirs.



Le marquage correspond à un état possible du réseau de Pétri. Le réseau change de marquage en fonction du franchissement de ses sommets-transition. On dit qu'un sommet-transition est *franchissable* si tous les sommets-place qui le précèdent contiennent au moins un jeton. Dans l'exemple de la figure, le sommet-transition 1 est franchissable alors que les deux autres sommets-transition ne le sont pas. Un sommet-transition franchissable u peut alors être *franchi* : cette opération consiste à changer le marquage en ôtant un jeton à chacun des sommets-place qui précèdent u et en ajoutant un jeton à chacun des sommets-place qui succèdent à u. On peut remarquer qu'à partir de M_0, le réseau de Pétri de la figure peut conduire uniquement à 3 autres marquages, obtenus en franchissant chacune des trois transitions : $M_1 = [0,0,1,1]$, $M_2 = [1,0,1,0]$ et $M_3 = [0,1,0,1]$. On dit que M_1, M_2 et M_3 sont *atteignables* à partir de M_0.

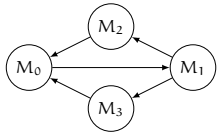
Q 3.3 Donnez la fonction `int franchissable(Rpet *R, int u, int* M);` qui teste si le sommet-transition `u` du réseau de Pétri `R` est franchissable pour le marquage `M`. Donnez la fonction `void franchir(Rpet *R, int u, int* M);` qui effectue le franchissement, sous l'hypothèse que `franchissable` retourne vrai pour `R`, `u` et `M`.

Graphe de marquage

On appelle *graphe de marquage* $G_M = (S_M, A_M)$ un graphe construit à partir d'un réseau de Pétri R et d'un marquage initial M de la façon suivante :

- S_M est l'ensemble des *sommets-marquage* : on a un sommet $s \in S_M$ par marquage atteignable depuis M.
- A_M est l'ensemble des *arcs-marquage* : on a un arc pour chaque paire (s_1, s_2) telle que le sommet-marquage s_2 est obtenu par franchissement d'une (unique) transition à partir du sommet-marquage s_1.

Par exemple, le marquage initial $M_0 = [1,1,0,0]$ donne le graphe de marquage suivant :



On souhaite construire efficacement le graphe de marquage associé à un réseau de Pétri R et à un marquage initial M. Malheureusement, on ne connaît pas toujours la liste des

marquages atteignables. On va donc devoir construire le graphe de marquage tout en découvrant les marquages atteignables. Un algorithme possible est de réaliser un parcours de la manière suivante :

ALGORITHME :
- Créer un graphe de marquage $G_M = (S_M, A_M)$.
- Ajouter dans G_M un sommet-marquage correspondant au marquage initial M.
- Pour chaque sommet-marquage $s \in S_M$ non encore traité :
  - Pour chaque transition u franchissable à partir de s :
    - Créer le marquage M' obtenu après franchissement de u.
    - Si M' n'appartient pas déjà au graphe :
      - Ajouter dans G_M un sommet-marquage s' pour M'.
    - Fin Si
    - Ajouter dans G_M l'arc (s, s').
  - Fin Pour
- Fin Pour

FIN ALGORITHME

On peut remarquer que l'on a besoin d'un graphe totalement dynamique pour ajouter facilement des sommets et des arcs. On va donc utiliser la structure suivante :

```
1  typedef struct sommet_marquage SommetMarquage;
2
3  typedef struct elem_list{
4    SommetMarquage* s;
5    struct elem_list* suiv;
6  } ElemListe;
7
8  struct sommet_marquage{
9    int *M;
10   ElemListe* Lsucc; /* Liste des sommets_marquage successeurs */
11 };
12
13 typedef struct{
14   int nbSommets;
15   ElemListe* Lsommets;  /* Liste des sommets-marquage du graphe */
16 } GrapheMarquage;
```

Q 3.4 Écrivez les fonctions suivantes :
- `GrapheMarquage* cree_graphe_marquage()` qui crée et initialise un graphe de marquage.
- `SommetMarquage* cree_sommet_marquage(int* M)` qui crée et initialise un sommet-marquage associé à `M`.
- `void ajoute_elem_list(SommetMarquage* s, ElemListe** L)` qui ajoute un sommet-marquage à une liste ; cette liste peut être une liste de successeurs (`Lsucc`) ou la liste des sommets du graphe (`Lsommets`).

Q 3.5 Écrivez une fonction `SommetMarquage* recherche_marquage(GrapheMarquage* GM, int* M, int n1)` qui recherche si le marquage `M` est présent dans le graphe `GM`.

Q 3.6 Donnez la fonction `GrapheMarquage* cree_graphe_marquage(Rpet* R, int* M);` qui crée le graphe de marquage correspondant au réseau de Pétri `R` et au marquage initial `M` en suivant l'algorithme décrit précédemment.

Q 3.7 Donnez une fonction de suppression d'un graphe de marquage.