Université de Technologie de Compiègne — Documents non autorisés
Examen médian NF16, Printemps 2018 — Durée 2 heures

Dans un souci d'équité, il ne sera pas répondu à des questions pendant la durée de l'épreuve (sauf aux étudiants étrangers ayant des difficultés pour comprendre le français aidés pour la compréhension du sujet). Si l'énoncé semble comporter une imprécision, faites un choix pour la lever que vous indiquerez clairement dans votre copie. Assurez-vous par une relecture attentive de l'énoncé que la réponse à votre question n'y figure pas. **Chaque partie est à rédiger sur une copie séparée, un point vous sera retranché sinon**.

Exercice I (copie numéro I : 4 points). Soit une fonction f : IN $\rightarrow$ IN.

1. O(f(n)) : montrer que la relation « est de l'ordre de » est réflexive, transitive, mais pas symétrique.

2. Trouver l'erreur

a) O(n) = O(n3 + (n – n3)) = O(max{n3, n – n3}) = O(n3)

b) O(n2) = O(n + n + … +n) = O(max{n , …, n}) = O(n)

Exercice II (copie numéro I, 6 points). Soit une fonction f telle que

f(n) = 0 si n = 0
f(n) = 1 si n = 1
f(n) = f(n-1) + f(n-2) si n >= 2

Question 1 : Que vaut f(2), f(3) et f(4) ? Reconnaissez-vous cette fonction ?

Question 2 :

a) Ecrire une fonction naïve en récursif qui calcule f(n). Il s'agit de la transcription intuitive vue en cours de la définition ci-dessus de cette fonction f.
b) Cette version de la fonction n'est pas efficace. Expliquer pourquoi et donner un exemple.
c) Proposer une version récursive plus efficace.

Question 3 :

a) Ecrire une fonction itérative qui calcule f(n).
b) Prouver que cette fonction calcule f(n).
c) Quelle est sa complexité ? Justifier.

Exercice III (copie numéro II, 10 points). Une partition d'un ensemble X est une décomposition de X en sous-ensembles **non-vides** deux à deux **disjoints** et qui **recouvrent** X. Par exemple, l'ensemble {0,1,2} possède les partitions {{0},{1},{2}}, {{0},{1,2}}, {{0,1},{2}}, {{0,2},{1}}, {{0,1,2}}. On souhaite représenter **une partition des valeurs entières de 0 à K** et pouvoir réaliser les opérations suivantes :
- Find : Indique à quel sous-ensemble appartient une valeur
- Union : Permet de faire l'union de deux sous-ensembles

Une première solution pour représenter une telle partition est d'utiliser un tableau. On stocke aux indices i et j la même valeur si les valeurs i et j font partie du même ensemble.

0	1	2	3	4	5	6
1	**1**	**2**	**3**	**1**	**3**	**2**

Dans l'exemple ci-dessus, la première ligne indique l'indice du tableau et la seconde ligne la valeur qui s'y trouve. La partition représentée est {{0,1,4},{2,6},{3,5}}.

1) Écrire une fonction qui renvoie 1 si deux valeurs *v1* et *v2* appartiennent au même ensemble et 0 sinon. Son prototype sera : « **int same_set_tab(int v1, int v2, int tab[], int n)** » avec *tab* le tableau représentant la partition et *n* la taille du tableau. Indiquer (sans démontrer) sa complexité.

2) Écrire une fonction qui réalise l'union entre les ensembles d'appartenance de deux valeurs *v1* et *v2*. Son prototype sera : « **void union_tab(int v1, int v2, int tab[], int n)** » avec *tab* le tableau représentant la partition et *n* la taille du tableau. Les éléments du sous-ensemble de *v2* seront ajoutés dans le sous-ensemble de *v1*. Indiquer (sans démontrer) sa complexité.

On propose une autre structure de représentation des sous-ensembles d'une partition basée sur les listes chaînées. Chaque sous-ensemble est représenté de la manière suivante :
- Une liste chaînée permet de représenter les valeurs qui appartiennent au même sous-ensemble. Chaque élément de la liste chaînée contient une valeur, un pointeur vers l'élément suivant ainsi qu'un pointeur vers un élément particulier appelé « Nœud »
- Un nœud contenant un pointeur vers la tête d'une liste chaînée et un pointeur vers la queue de la liste chaînée

```
struct Node {
        Elem * head ; // Pointeur vers le premier élément de la liste chaînée
        Elem * tail ; // Pointeur vers le dernier élément de la liste chaînée
} ;
struct Elem {
        int value ; // Valeur appartenant au sous-ensemble
        struct Elem * next ; // Pointeur vers le prochain élément de la liste chaînée
        struct Node * nd ; // Pointeur vers le noeud
} ;
```

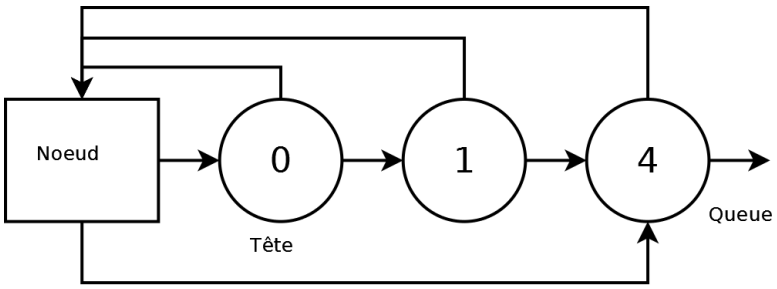

Illustration 1: Sous-ensemble {0,1,4}

Une partition est alors représentée sous la forme d'un tableau de pointeurs vers *struct Elem*. L'indice *i* du tableau contient un pointeur permettant d'accéder à l'élément de type *struct Elem* utilisé pour représenter la valeur *i*.

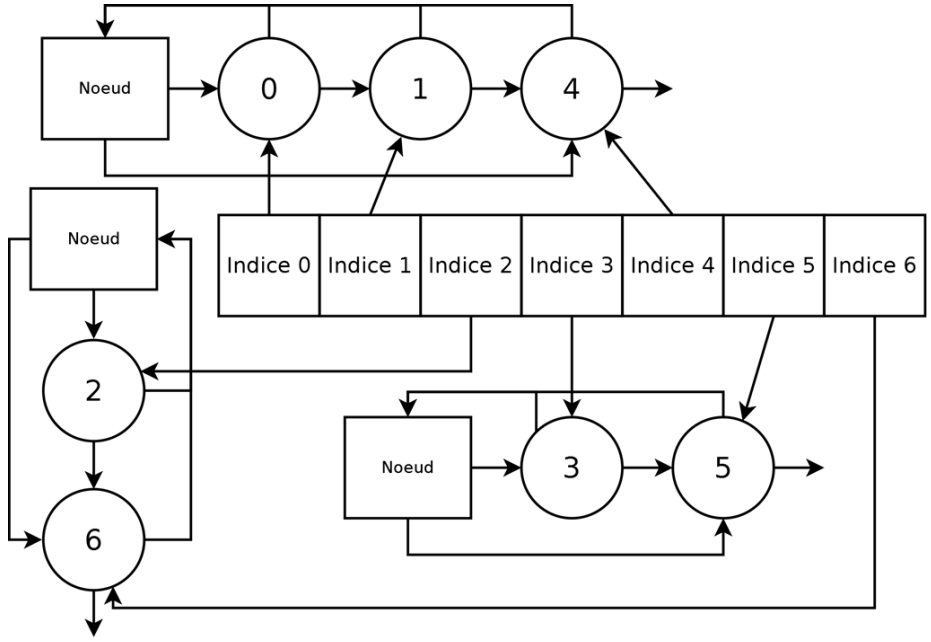

Illustration 2: Partition {{0,1,4},{2,6},{3,5}}

3) Écrire une fonction permettant de créer un nouveau sous-ensemble composé d'un unique élément de valeur *val*. Son prototype sera : « **struct Node * createSubset(int val)** »

4) Écrire une fonction renvoyant la valeur de l'élément de tête de la liste chaînée d'appartenance d'un élément *e*. Son prototype sera : « **int find(struct Elem * e)** ». Indiquer (sans démontrer) sa complexité.

5) Écrire une fonction renvoyant 1 si deux éléments *e1* et *e2* appartiennent au même sous-ensemble et 0 sinon. Son prototype sera : « **int same_set(struct Elem * e1, struct Elem * e2)** ». Indiquer (sans démontrer) sa complexité.

6) Écrire une fonction réalisant l'union entre le sous-ensemble auquel appartient *e1* et le sous-ensemble auquel appartient *e2*. Son prototype sera : « **void union(struct Elem * e1, struct Elem * e2)** ». Les éléments du sous-ensemble de v2 seront ajoutés dans le sous-ensemble de v1. On veillera à réaliser les opérations de libération mémoire. Indiquer (sans démontrer) sa complexité.

7) Écrire une fonction permettant de retirer un élément *e* de son sous-ensemble d'appartenance. Un nouveau sous-ensemble contenant uniquement *e* sera créé et renvoyé par la fonction. Son prototype sera : « **struct Node * extract(struct Elem * e)** ». On prendra en compte le cas où le sous-ensemble ne contenait déjà que *e*.

8) Écrire une fonction permettant d'initialiser une partition constituée des valeurs entières entre 0 et *k*. La partition sera composée de sous-ensembles contenant chacun une unique valeur. Le prototype de la fonction sera : « **struct Elem ** init(int k)** ». La fonction renverra un tableau contenant à l'indice *i* un pointeur permettant d'accéder à l'élément de type *struct Elem* et de valeur *i*.

9) On propose de modifier la structure *Node* afin d'ajouter un attribut *size*, indiquant le nombre d'éléments contenus dans la liste chaînée rattachée. En utilisant et en mettant à jour cet attribut, écrire une version plus efficace de la fonction union.
Indiquer (sans démontrer) sa complexité.