Conservatoire National des Arts et Métiers
Département Informatique – EPN 5

Année 2017–2018

UE NFP 136 (Responsable : Cédric Bentz)
Calculatrices graphiques et appareils électroniques interdits — Durée : 3h
Documents autorisés : documents papier uniquement

Examen de session 1 de l'UE NFP 136 (VARI 2)

25 juin 2018

Exercice 1 : degrés des sommets d'un graphe et tri par tas

Dans cet exercice, on s'intéresse aux degrés des sommets d'un graphe orienté. Le degré **sortant** d'un sommet d'un tel graphe est son nombre de successeurs. On commencera par écrire une méthode qui, étant donné un graphe à n sommets et un numéro de sommet (entre 1 et n), renvoie le degré sortant de ce sommet (c'est-à-dire son nombre de successeurs).

On rappelle que, si n désigne le nombre de sommets d'un graphe orienté, la matrice d'adjacence de ce graphe est une matrice d'entiers contenant n lignes et n colonnes telle que, pour tout couple de sommets i et j, l'élément sur la ligne $i-1$ et la colonne $j-1$ vaut 1 s'il existe un arc du sommet i vers le sommet j (autrement dit si j est un **successeur** de i), et 0 sinon.

Par ailleurs, le tableau de listes d'adjacence d'un tel graphe est un tableau de taille n, dont chaque élément (l'élément d'indice $i-1$ étant associé au sommet i) est une liste chaînée d'entiers (éventuellement vide), contenant, pour chaque i, les numéros des **successeurs** du sommet i.

Question 1

(a) Implémenter cette méthode en JAVA, dans le cas où le graphe est donné sous la forme d'une matrice d'adjacence. Son en-tête sera : `public static int degreSortant(int[][] matAdj, int numDuSommet)`, où `matAdj` est la matrice d'adjacence du graphe considéré, `numDuSommet` le numéro du sommet (entre 1 et le nombre de sommets) dont on veut calculer le degré sortant, et la valeur retournée est ce degré.

(b) Implémenter cette méthode en JAVA, dans le cas où le graphe est donné sous la forme d'un tableau de listes chaînées. Son en-tête sera : `public static int degreSortant(Liste[] tabListAdj, int nSommet)`, où `tabListAdj` est le tableau de listes d'adjacence du graphe considéré, `nSommet` le numéro du sommet (entre 1 et le nombre de sommets) dont on veut calculer le degré sortant, et la valeur retournée est ce degré.

Dans un second temps, on souhaite écrire une méthode permettant de calculer le nombre de sommets de degré sortant i dans graphe donné à n sommets, pour toutes les valeurs possibles de i (entre 0 et $n-1$).

Question 2

(a) Implémenter cette méthode en JAVA, dans le cas où le graphe est donné sous la forme d'un tableau de listes chaînées, en utilisant la méthode écrite à la Question 1(b). Son en-tête sera : `public static int[] histogrammeDegres(Liste[] tabListAdj)`, où, ici aussi, `tabListAdj` est le tableau de listes d'adjacence du graphe considéré, et la valeur de retour est un tableau dont le $i$ème élément est égal au nombre de sommets de degré sortant i dans ce graphe, pour tous les i possibles.

(b) Que faudrait-il modifier à la Question 2(a) si le graphe considéré était donné sous la forme d'une matrice d'adjacence ? Justifier.

Enfin, on souhaite écrire une méthode qui, étant donné un graphe orienté, retourne un tableau d'entiers dont le $i$ème élément est l'indice du sommet ayant le $i$ème plus petit degré sortant dans ce graphe, pour tout i. Ainsi, si on considère le graphe donné par le tableau de listes d'adjacence suivant (qui indique que le degré sortant du sommet 1 –à l'indice 0– est 2, celui du sommet 2 est 4, celui des sommets 3 et 5 est 1, celui du sommet 4 est 5, celui du sommet 6 est 3, celui du sommet 7 est 6, et celui du sommet 8 est 0)

```
[0] --> 5 --> 6
[1] --> 1 --> 3 --> 4 --> 6
[2] --> 5
[3] --> 1 --> 3 --> 5 --> 7 --> 8
[4] --> 2
[5] --> 1 --> 5 --> 8
[6] --> 1 --> 2 --> 4 --> 5 --> 6 --> 8
[7] null
```

alors on veut retourner le tableau suivant (dans cet ordre) :

```
| 8 | 3 | 5 | 1 | 6 | 2 | 4 | 7 |
```

On peut remarquer que, si on note n le nombre de sommets du graphe et $deg(i)$ le degré sortant du sommet i pour tout i entre 1 et n, alors trier les quantités $(n+1)\cdot deg(i)+i$ par ordre croissant produit le même ordre que trier les degrés sortants $deg(i)$ par ordre croissant. À partir de chaque quantité $(n+1)\cdot deg(i)+i$, on peut alors récupérer la valeur de i en faisant une division euclidienne par $n+1$ (i sera égal au reste de cette division euclidienne, et $deg(i)$ à son quotient). On triera ces quantités à l'aide d'un tri par tas, en utilisant, sans les réécrire, toutes les méthodes de la classe JAVA Tas (vue en cours) qui paraîtront nécessaires (`minimum`, `supprimerMin` ou `inserer`).

Question 3

1. Implémenter cette méthode en JAVA, dans le cas où le graphe est donné sous la forme d'un tableau de listes chaînées. Son en-tête sera : `public static int[] triSommetsParDegViaTasMin(Liste[] tabListAdj)`, où `tabListAdj` est le tableau de listes d'adjacence du graphe considéré, et la valeur de retour est un tableau dont le $i$ème élément est l'indice du sommet ayant le $i$ème plus petit degré sortant dans ce graphe, pour tout i. Comment faut-il adapter cette méthode si le graphe est donné sous la forme d'une matrice d'adjacence ? Justifier.

2. Donner la complexité en temps (au pire cas) et en espace de la méthode `triSommetsParDegViaTasMin`, en fonction du nombre de sommets n et du nombre d'arcs m, selon la forme sous laquelle le graphe est donné.

3. Illustrer en détails le déroulement de `triSommetsParDegViaTasMin`, étape par étape, sur le graphe à 8 sommets donné ci-dessus en exemple.

Exercice 2 : ajout d'un élément dans une liste triée

Dans cet exercice, on vous demande d'implémenter une méthode JAVA permettant d'insérer, dans une liste chaînée d'entiers triés, que l'on notera `l`, un élément (entier) `x` donné, de façon à ce que la liste obtenue reste triée. Ainsi, si `x` = 17 et si la liste considérée est comme ceci

```
-> 1 -> 3 -> 4 -> 7 -> 10 -> 11 -> 15 -> 23 -> 26 -> 37
```

alors la liste que l'on souhaite obtenir à la fin est la suivante :

```
-> 1 -> 3 -> 4 -> 7 -> 10 -> 11 -> 15 -> 17 -> 23 -> 26 -> 37
```

Travail demandé. Implémenter en JAVA cette méthode de la classe Liste, dont l'en-tête est : `public static Liste insererDansListeTriee(Liste l, int x)`, où `x` et `l` sont définis comme ci-dessus, et la valeur de retour est la liste obtenue après insertion de `x`. On détaillera aussi sa complexité en temps (au pire cas) et en espace, en fonction de la taille n de la liste `l`.

Exercice 3 : test de la perfection d'un nombre

Un nombre parfait est un nombre qui est égal à la somme de ses diviseurs propres. Par exemple, $6 = 1 + 2 + 3$ et $28 = 1 + 2 + 4 + 7 + 14$ sont parfaits.

Travail demandé. Implémenter en JAVA une méthode qui permet de tester si un entier `n` passé en paramètre est parfait ou non et renvoie ainsi *true* ou *false*, et dont l'en-tête est : `public static boolean estParfait(int n)`.

Exercice 4 : chronogramme d'exécution

On considère un système monoprocesseur et les 3 processus P1, P2 et P3, qui y effectuent du calcul et des opérations d'entrée/sortie (sur un disque), selon les temps et l'ordre donnés ci-dessous :

Processus P1	Processus P2	Processus P3
Calcul : 3 unités de temps	Calcul : 1 unités de temps	Calcul : 2 unités de temps
E/S : 5 unités de temps	E/S : 4 unités de temps	E/S : 3 unités de temps
Calcul : 2 unités de temps	Calcul : 3 unités de temps	Calcul : 6 unités de temps
E/S : 1 unité de temps	E/S : 2 unités de temps	
Calcul : 1 unité de temps	Calcul : 1 unité de temps	

On considère que l'ordonnancement sur le processeur est à priorité avec préemption : le processus élu à un instant t est le processus prêt de plus forte priorité. On a ici : priorité (P2) > priorité (P1) > priorité (P3).

On considère que l'ordre de service des requêtes d'E/S pour le disque suit toujours une politique FIFO. Une seule requête d'E/S peut être traitée à la fois, et une requête d'E/S ne peut pas être interrompue une fois lancée.

Sur le graphique suivant, donner le chronogramme d'exécution des 3 processus P1, P2, et P3. On distinguera les états des processus :

- Élu (En exécution sur le processeur)
- Prêt (En attente du processeur)
- AttD (En attente du disque)
- E/S (Entrée/Sortie)