Conservatoire National des Arts et Métiers
École SITI - Département Informatique

Année 2015–2016

UE NFP 136 (Responsable : Cédric Bentz)
Tous documents papier autorisés

Examen de l'UE NFP 136 (VARI 2)

28 juin 2016

Exercice 1 : tri en deux moitiés d'un tableau d'entiers

Dans cet exercice, on souhaite trier un tableau donné contenant $n = 2p$ entiers (on supposera n pair dans tout l'exercice) de la façon suivante :

- Les p premiers entiers du tableau obtenu seront les p plus petits entiers du tableau initial, triés par ordre croissant.
- Les p derniers entiers du tableau obtenu seront les p plus grands entiers du tableau initial, triés par ordre décroissant.

Par exemple, si le tableau à "trier" contient les 10 entiers suivants :

$$4 - 2 - 5 - 10 - 6 - 3 - 7 - 2 - 9 - 6$$

Alors, on souhaite obtenir à la fin le tableau suivant :

$$2 - 2 - 3 - 4 - 5 - 10 - 9 - 7 - 6 - 6$$

<u>Travail demandé</u>

1. En utilisant (sans le réécrire) l'algorithme de tri de son choix, implémenter en JAVA une méthode triant un tableau d'entiers en 2 moitiés telles que décrites ci-dessus, et dont l'en-tête est : `public static void triEn2(int[] tabATrier)`, où `tabATrier` est le tableau d'entiers à "trier", et qui contiendra à la fin de la méthode les deux moitiés triées par ordre croissant et décroissant (tri "sur place").
2. Détailler la complexité en temps (au pire cas) et en espace de cette méthode, en fonction du nombre d'entiers à trier n.
3. Illustrer en détails le déroulement de cette méthode (y compris le tri) sur le tableau de taille 10 donné ci-dessus en exemple.

Exercice 2 : transformer un graphe orienté en non orienté

Dans cet exercice, on vous demande d'implémenter une méthode JAVA permettant de transformer un graphe orienté (donné sous la forme d'une matrice d'adjacence) en graphe non orienté, en "oubliant" l'orientation de ses arcs. L'en-tête d'une telle méthode est le suivant :

```
public static void grapheOrienteEnNonOriente(int[][] matAdj) {
    /* corps de la méthode */
}
```

Le paramètre `matAdj` représente la matrice d'adjacence du graphe orienté considéré. Si n désigne le nombre de sommets du graphe, `matAdj` est donc une matrice d'entiers contenant n lignes et n colonnes telle que, pour tout couple de sommets i et j, l'élément `matAdj[`$i-1$`][`$j-1$`]` vaut 1 s'il existe un arc entre les sommets i et j, et 0 sinon. Ainsi, si on considère le graphe orienté donné par la matrice d'adjacence suivante :

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

Ce graphe est composé de 3 sommets $1, 2, 3$, et de 2 arcs $(1, 2)$ et $(2, 3)$. Le graphe non orienté associé est donc composé des 3 mêmes sommets, et de 2 arêtes : entre les sommets 1 et 2, et entre les sommets 2 et 3. L'élément de la matrice d'adjacence d'un graphe non orienté situé sur la $i$ème ligne et $j$ème colonne vaut 1 s'il existe une arête entre les sommets i et j, et 0 sinon. La matrice d'adjacence associée à ce graphe non orienté est donc :

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

Ainsi, si l'élément sur la ligne i et la colonne j vaut 1, alors l'élément sur la ligne j et la colonne i doit également valoir 1.

<u>**Travail demandé**</u>

Implémenter une telle méthode en JAVA, et déterminer sa complexité en temps au pire cas (on notera n le nombre de sommets du graphe).

Exercice 3 : insertion dans une liste après la $i$ème occurrence d'un élément donné

Dans cet exercice, on vous demande d'implémenter une méthode JAVA permettant d'insérer, dans une liste chaînée d'entiers donnée, un élément (entier) x donné, juste après la $i$ème occurrence d'un autre élément (entier) y donné,

pour un certain $i \geq 1$. Par exemple, si $x = 3$, $y = 4$, et $i = 2$, et si la liste considérée est comme ceci

```
-> 1 -> 4 -> 2 -> 9 -> 5 -> 4 -> 7 -> 1 -> 4 -> 3
```

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

```
-> 1 -> 4 -> 2 -> 9 -> 5 -> 4 -> 3 -> 7 -> 1 -> 4 -> 3
```

Travail demandé

1. Implémenter en JAVA cette méthode de la classe Liste, dont l'en-tête est : `public static void insererApres(Liste l, int i, int x, int y)`, où i, x et y sont définis comme ci-dessus, et l est la liste dans laquelle insérer l'élément x. À noter : si l'élément y n'apparaît pas au moins i fois dans la liste ou si $i < 1$, alors cette méthode ne fera rien.

2. Détailler la complexité en temps (au pire cas) et en espace de cette méthode, en fonction de la taille de la liste l.

Exercice 4 : insertions d'éléments dans un tas

Travail demandé

On vous demande de répondre au questions suivantes, en indiquant à chaque fois la plus faible complexité possible (à un facteur constant près) :

1. Quelle est la complexité en temps nécessaire pour ajouter un élément dans un tas min contenant n éléments, si l'élément à ajouter est plus petit que tous les éléments du tas, sauf un ?

2. Quelle est la complexité en temps nécessaire pour ajouter un élément dans un tas min contenant n éléments, si l'élément à ajouter est plus grand que tous les éléments du tas, sauf un ?

3. Quelle est la complexité en temps nécessaire pour ajouter dans un tas min initialement vide les n premiers entiers, dans l'ordre suivant : $1, n, n-1, \ldots, 3, 2$?

4. Quelle est la complexité en temps nécessaire pour ajouter dans un tas min initialement vide les n premiers entiers, dans l'ordre suivant : $n, 1, 2, \ldots, n-2, n-1$?

5. Insérer dans un tas min initialement vide les 10 premiers entiers, dans l'ordre suivant : $1, 5, 4, 3, 2, 10, 6, 7, 8, 9$. **Détailler chacune de ces insertions sur le tas min représenté comme un arbre binaire**.

6. Supprimer ensuite le minimum du tas obtenu à l'issue de la question précédente, en détaillant bien toutes les étapes de cette suppression.

Exercice 5 : processus et adresses

On considère 3 processus A, B et C :

- Le processus A dispose de 4 pages P1, P2, P3 et P4 dans son espace d'adressage ;
- Le processus B dispose de 3 pages P1, P2 et P3 dans son espace d'adressage ;
- Le processus C dispose de 2 pages P1, P2 dans son espace d'adressage.

Les cases mémoires numérotées de 1 à 15 sont au nombre de 15. Une case mémoire a une capacité de 256 octets. A l'instant t, l'état d'allocation en mémoire centrale des pages des 3 processus est le suivant :

- Les pages P2 et P3 du processus A sont chargées respectivement dans les cases 10 et 7 de la mémoire centrale ;
- Les pages P1 et P3 du processus B sont chargées respectivement dans les cases 15 et 5 de la mémoire centrale ;
- La page P1 du processus C est chargée dans la case 3 de la mémoire centrale.

<u>Travail demandé</u>

1. Représenter sur un dessin les structures allouées pour ce type d'allocation mémoire et la mémoire centrale correspondant à l'allocation décrite.
2. Dans chacun des deux cas suivants, calculer l'adresse paginée, puis l'adresse physique, correspondant à l'adresse linéaire indiquée. En cas de défaut de page, on supposera que la page manquante est chargée en mémoire centrale dans la première case libre de plus petit numéro :
   (a) Adresse linéaire 624 dans l'espace d'adressage du processus A ;
   (b) Adresse linéaire 506 dans l'espace d'adressage du processus C.