Analyse d'algorithmes et programmation - Examen

Enseignants: Christina Boura, Gaëtan Leurent

3 mai 2017

Durée du contrôle : 2h.

Les documents sont autorisés.

1 Questions

Répondre par **vrai** ou **faux** aux questions suivantes en justifiant votre réponse. Des réponses sans justification ne seront pas prises en compte.

1. Soit $f(n) = an^3$, où a une constante et soit $g(n) = n^3$. Alors, $f(n) = \mathcal{O}(g(n))$.
2. On suppose que $f(n) = \mathcal{O}(n)$ et que $g(n) = \mathcal{O}(n \log n)$. Alors $f(n) = \mathcal{O}(g(n))$.
3. Le tri par fusion sur un tableau de taille n déjà trié, se fait en temps $\mathcal{O}(n)$.
4. Si chaque nœud d'un arbre binaire de recherche contenant n éléments a soit 0 soit exactement 2 fils, alors l'hauteur de l'arbre est $\Theta(\log n)$.
5. L'insertion d'un élément dans un arbre binaire de recherche ayant n nœuds se fait toujours en $\mathcal{O}(\log n)$.
6. Pour une table de hachage de taille n employant la méthode de chaînage pour résoudre les collisions et contenant n éléments, on peut toujours trouver le plus petit élément en temps $\mathcal{O}(1)$.
7. L'implantation d'un ensemble dynamique avec une table de hachage garantit toujours un temps de recherche en $\mathcal{O}(1)$.
8. Lorsqu'on double la taille d'une table de hachage, on peut continuer à utiliser la même fonction de hachage.

2 Tri

Étant donnés k tableaux déjà triés, de taille n chacun, décrire un algorithme en temps $\mathcal{O}(nk \log k)$ qui fusionne ces k tableaux en un seul tableau trié.

3 Structures de données

On suppose qu'on a une liste d'entiers non triés. On veut trouver et imprimer tous les entiers qui sont présents plus d'une fois dans la liste. Par exemple, étant donnée la liste $\{1, 5, 3, 5\}$, on doit imprimer 5.

Donner trois algorithms différents pour effectuer cette tâche. Le premier doit être en temps $\mathcal{O}(n^2)$, le deuxième en temps $\mathcal{O}(n \log n)$ et le troisième en temps $\mathcal{O}(n)$. Vous pouvez écrire du pseudo-code (sans pour autant entrer trop dans les détails) ou simplement décrire dans le language courant vos algorithmes.

Remarque : Les trois algorithmes peuvent être réalisés avec les algorithmes et les structures de données vus en cours.

4 Une nouvelle structure de données

On considère une nouvelle structure de données qui peut être utilisée comme une alternative aux arbres binaires de recherche. Cette structure de données stockant n éléments peut être représentée comme une matrice carrée de taille $\sqrt{n} \times \sqrt{n}$. On suppose que $\sqrt{n}$ est un nombre entier. Chaque colonne et ligne de la matrice sont triés par ordre croissant (voir la figure ci-dessous pour un petit exemple). Le plus petit élément se trouve toujours en haut et à gauche de la matrice et l'élément le plus grand se trouve toujours en bas et à droite de la matrice.

M	1	2	3	4	5
1	2	4	5	7	10
2	3	6	8	15	17
3	14	19	25	32	39
4	18	26	34	41	47
5	21	28	36	45	52

Ci-dessous est donné le pseudo-code d'un algorithme `Recherche` qui permet de chercher un élément de clé k dans la structure de données M. Pour simplifier le raisonnement, on suppose qu'on ne stocke que les clés, sans données associées. Chaque élément consiste donc en un entier. On note $l = \sqrt{n}$ le nombre de lignes et des colonnes de la matrice. Par $M[r, c]$ on désigne l'élément à l'intersection de la ligne r et de la colonne c de la matrice M. L'algorithme renvoie `Vrai` si l'élément k se trouve dans la matrice et renvoie `Faux` sinon.

```
Recherche(M, l, k)
   r <- l
   pour c de 1 à l
      tant que M[r,c] > k ET r > 1
         r <- r - 1
      si M[r,c] = k
         renvoyer "VRAI"
   renvoyer "FAUX"
```

1. Montrer en détail les étapes que cet algorithme effectue pour la recherche de la clé 15 dans la matrice ci-dessus.
2. Expliquer comment cet algorithme marche de façon générale.
3. Quelle est la complexité de cet algorithme pour une structure de données contenant n éléments ? Donner votre réponse en notation $\mathcal{O}$. Justifier.
4. Comparer cette complexité avec la recherche d'un élément dans un arbre binaire de recherche équilibré de taille n.

5 Graphes

Dans cette partie, nous allons étudier deux problèmes d'optimisation sur un graphe non orienté (non pondéré) $G = (S, A)$.

Un ensemble de sommet $T \subset S$ est appelé une *couverture* si T contient au moins une extremité de chaque arrête du graphe. Formellement,

$$T \text{ est une couverture de } G \iff \forall (u, v) \in A, u \in T \text{ ou } v \in T$$

Le problème de *couverture minimum* consiste à trouver une couverture de cardinalité minimale.

Un ensemble de sommet $T \subset S$ est appelé une *clique* si il y a une arrête dans A entre chaque paire de sommets de T. Formellement,

$$T \text{ est une clique de } G \iff \forall u \neq v \in T, (u, v) \in A$$

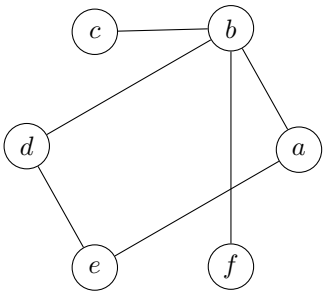

FIGURE 1 – Graphe G_1.

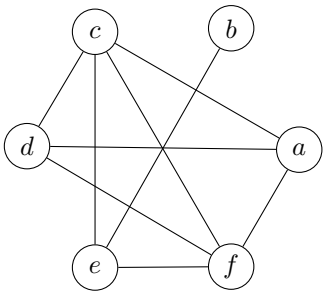

FIGURE 2 – Graphe G_2.

Le problème de *clique maximum* consiste à trouver une clique de cardinalité maximale.
Exercice 1 (1 point)
Trouver une couverture minimum pour le graphe G_1, et une clique maximum pour le graphe G_2.

Exercice 2 (1 point)
Definir le problème de décision COUVERTURE *correspondant au problème de couverture minimum, et le problème de décision* CLIQUE *correspondant au problème de clique maximum.*
Expliquer pourquoi ces problèmes sont dans la classe NP.

On défini le complémentaire $\overline{G} = (S, \overline{A})$ d'un graphe $G = (S, A)$, comme le graphe dont l'ensemble des arrêtes est le complémentaire de A :

$$\forall u \neq v \in S,\ (u, v) \in \overline{A} \Leftrightarrow (u, v) \notin A$$

Par exemple, G_1 est le complémentaire de G_2 (et réciproquement).
Exercice 3 (2 points)
Montrer que G possède une couverture de taille t si et seulement si $\overline{G}$ possède une clique de taille $|S| - t$.

Exercice 4 (2 points)
Montrer que COUVERTURE *est un problème NP-Complet, en utilisant le fait que* CLIQUE *est un problème NP-Complet.*

On propose un algorithme glouton pour trouver une couverture minimum :

1: **fonction** COUVERTUREGLOUTON($G = (S, A)$)
2: $T \leftarrow \varnothing$
3: **tant que** $A \neq \varnothing$ **faire**
4: Choisir une arrête (u, v)
5: Ajouter u et v à T
6: Retirer de A toutes les arrêtes ayant u ou v comme extremité
7: **retourner** T

Exercice 5 (2 points)
Quelle est la complexité de COUVERTUREGLOUTON *?*
Pensez-vous que cette algorithme trouve une couverture minimum ? Expliquer pourquoi, et donner une preuve ou un contre-exemple.

Exercice 6 (2 points)
Montrer que la couverture renvoyée par COUVERTUREGLOUTON *contient au pire deux fois plus de sommets qu'une couverture minimum.*

Exercice 7 (3 points)
Donner un algorithme polynomial pour construire une couverture minimum, dans le cas particulier où le graphe est acyclique.
Indice : on pourra monter que dans un graphe acyclique avec un ensemble d'arrêtes non vide, il existe au moins un sommet qui est incident à une seule arrête.