Complexité
TD 5 - Correction

Master 1 Informatique

Serge Miguet, Guillaume Metzler, Tess Masclef
Institut de Communication (ICOM)
Université de Lyon, Université Lumière Lyon 2
serge.miguet@univ-lyon2.fr guillaume.metzler@univ-lyon2.fr
tess.masclef@univ-lyon2.fr

File de Priorité

Une file de priorités doit permettre de connaître rapidement quelle est la tâche la plus prioritaire parmi un ensemble de tâches.

Montrez qu'un simple tableau trié peut répondre rapidement à cette attente. En classant les éléments par ordre décroissant de priorité, il est très simple de pouvoir accéder aux éléments prioritaires dans une liste. Cela nécessite par contre de ranger les tâches par ordre de priorité (on va faire du tri).

On peut également associer un pointeur sur la tâche la plus prioritaire (affectation à un coût constant). En revanche, l'identification de la tâche la plus prioritaire nécessite de parcourir l'ensemble de la liste des tâches. Elle a donc un coût linéaire.

Quelle est la complexité de l'ajout d'une nouvelle tâche, associée à un niveau de priorité ? En reprenant ce qui précède, l'ajout d'une nouvelle tâche avec un niveau de priorité nécessite, dans le pire des cas, de parcourir l'ensemble d'un tableau déjà trié pour la placer au bon endroit.

Quelle est la complexité de la suppression de la tâche la plus prioritaire du tableau ? Le coût de suppression de la tâche prioritaire est constant et ne dépend pas de la longueur de la liste. En revanche, ce coût pourrait être linéaire s'il faut ensuite déplacer les élements dans notre liste (question de point de vue).

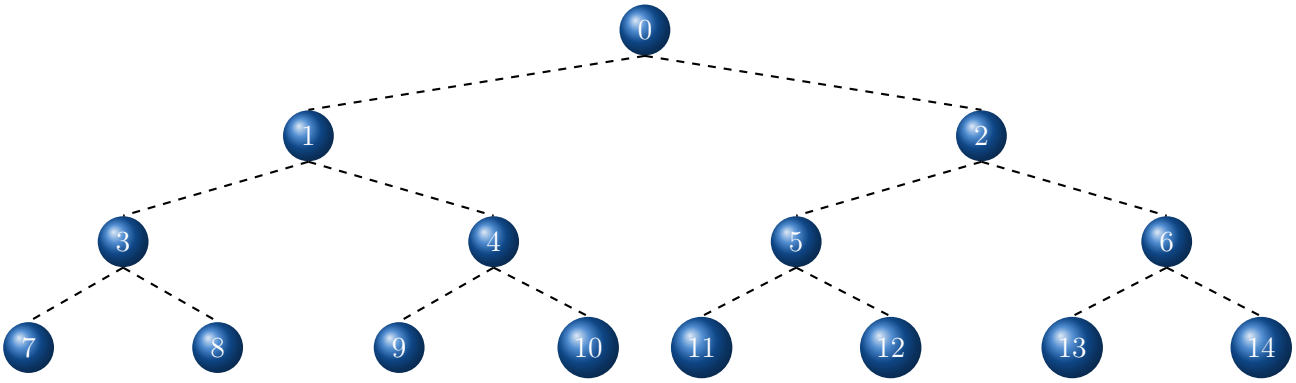

Figure 1: Représentation d'un arbre binaire pour des niveaux allant de 0 à 4.

Tas Binaire

Un tas binaire est une structure de données informatique utilisée pour gérer de manière efficace les files de priorité. Il est également à la base de l'un des algorithmes de tri les plus efficaces de la littérature : le *heapsort*.

Il s'agit d'un arbre binaire. Chaque nœud (sauf la racine) possède un nœud père, et chaque nœud (sauf les feuilles) possède au plus deux fils. Il est rempli avec des clefs, niveau par niveau, et de gauche à droite dans le dernier niveau. Seul le dernier niveau peut être incomplet.

1) Combien de nœuds peux porter le niveau k ?

   Le noeud de niveau k peut porter au plus 2^k éléments.

2) Combien de clefs peuvent être mémorisées dans un arbre ayant k niveaux complets ?

Nous avons vu que chaque niveau comporte au plus 2^l, où l est le niveau considéré. Ainsi un arbre à k niveau complets pourra mémoriser un nombre de clefs au plus égal à :

$$\sum_{l=0}^{k} 2^l = \frac{2^{k+1}-1}{2-1} = 2^{k+1} - 1.$$

3) Quelle est la hauteur h de l'arbre complet (c'est à dire la longueur d'un plus long chemin allant d'une feuille à la racine) correspondant à un tas de n clefs ?

La question précédente a montré qu'un arbre complet à k niveaux peut stocker au plus $2^{k+1} - 1$ clefs, où $k + 1 = h$ est la longueur du chemin dans l'arbre. Il nous suffit de résoudre l'équation $n = 2^h + 1$, on a donc

$$h = \log_2(n + 1) = \frac{\ln(n + 1)}{\ln(2)}.$$

4) Quel sont les indices des deux nœuds fils du nœud i, prenant en compte que le premier indice est zéro ?

Pour se faire une idée de la relation qui existe entre le numéro d'un noeud père et celui des noeuds fils, on peut regarder la Figure 1.

- Fils de gauche : $2 \times i + 1$
- Fils de droite : $2 \times i + 2$

5) Quel est l'indice du nœud père du nœud j ?

Il suffit cette fois-ci de faire le processus inverse à celui de la question précédente. Il faudra simplement prêter attention à attribuer le noeud fils au bon noeud père.

D'après la question précédente, s'il existe un entier $k \in \mathbb{N}$ tel que $j = 2k + 1$ ou $j = 2k + 2$ alors le noeud père du noeud j est le noeud k.

Pour trouver cet entier k, on procèdera à la division euclidienne de j par 2. Alors le noeud père du noeud j est le noeud k si et seulement si :

$$j \equiv 0\ [k + 1] \quad \text{ou} \quad j \equiv 1\ [k].$$

Vérifions sur deux exemples :

- Le noeud père du noeud 7 est 3, et on a bien $7 \equiv 1\ [3]$.
- De la même façon, nous avons également 4 qui est le neoud père du noeud 10. En effet : $10 \equiv 0\ [4 + 1]$.

6) Montrez que la valeur maximale de clef se trouve à la racine de l'arbre.

Le raisonnement utilise la propriété du tas-max qui dit que *la clef d'un nœud doit toujours être de valeur inférieure ou égale à celle de son père.*

Supposons que la valeur maximale d'une clef se trouve à un noeud d'un niveau quelconque $N > 0$ de l'arbre. Dans ce cas, d'après la propriété du tas-max, le noeud père a aussi une valeur de clef supérieure, le père du noeud en question également. On peut ainsi par récurrence le niveau de localisation do noeud montrer que le noeud père du noeud,situé à tout niveau $0 < K \leq N$ a une valeur de clef supérieure à celle de son fils. Le noeud du niveau 0 n'ayant pas de noeud père, il a donc la plus grande valeur de clef dans l'arbre. Le noeud situé au niveau 0 correspond à la racine de l'arbre.

7) Quelle est la complexité de l'opération permettant d'afficher la valeur de la priorité la plus haute ?

La clef la plus propriétaire se trouvant à la racine de l'arbre, l'opération a donc une complexité constante, *i.e.* en $\mathcal{O}(1)$.

8.a) Étant donné un tas-max, proposez un algorithme permettant d'ajouter une clef au tas, en gardant la propriété du tas-max.

Ajoutons cette nouvelle clef à un noeud de l'arbre se trouvant aux niveaux des feuilles de notre arbre (si nécessaire, on peut créer un nouveau niveau afin de pouvoir attribuer cette clef).
Ensuite, on se contente de vérifier que la propriété du tas-max est vérifiée en procédant comme suit : tant que la valeur de la clef nouvellement ajoutée est supérieure à celle de son noeud père ou que l'on n'a pas atteint la racine, échanger les clefs des noeuds fils et père. Cet algorithme s'arrête donc une fois que la propriété du tas-max est vérifiée, donc, au plus tard, lorsque la nouvelle de clef est remontée à la racine de l'arbre.

8.b) Étant donné un tas-max, quelle est la complexité de cet algorithme, en fonction de la hauteur de l'arbre h ? en fonction du nombre de clefs dans le tas n ?

Regardons la complexité de ces deux algorithmes séparément

- en reprenant la procédure précédente, nous devrons donc effectuer au plus h tests de comparaison et au plus h inversions. La complexité est donc globalement linéaire en la hauteur h de l'arbre.
- il suffit de savoir à quelle hauteur d'arbre correspond un ensemble de n noeuds. S'agissant d'un arbre binaire, cela correspond à arbre de hauteur $h = \log_2(n)$. On a globalement une complexité en $\mathcal{O}(\log_2(n))$.

9) Trouvez un algorithme permettant de supprimer du tas la clef ayant la priorité la plus haute, tout en conservant la propriété de tas-max. Quelle est la complexité de cet algorithme, en fonction de la hauteur de l'arbre h ? en fonction du nombre de clefs dans le tas n ?

La clef avec la priorité la plus haute se trouvait à la racine de l'arbre. Il faut maintenant remplacer cette valeur là par une valeur existante dans l'arbre. Nous allons prendre celle qui se trouve dans la racine la plus à droite de notre arbre et la placer à la racine. On effectuera ensuite les échanges nécessaires afin d'obtenir notre fameuse propriété.

Si la nouvelle valeur de clef à la racine vérifie la propriété du tas-max, nous n'avons rien à faire. Sinon, on compare la valeur de la racine (notre noeud père) avec celles de ces deux noeuds fils, nous devons alors on échanger la valeur du noeud père avec celle du noeud fils ayant la plus grande valeur de clef. Le noeud fils sélectionné devient alors notre nouveau noeud père.
On va donc se concentrer sur des arbres binaires sur deux niveaux à chaque fois en répétant le processus précédent jusqu'à ce que la propriété du tas-max soit respectée, *i.e.* jusqu'à ce que tous les noeuds pères aient bien une valeur supérieure aux noeuds fils.

Cet algorithme a à nouveau une complexité linéaire en la hauteur h de l'arbre ou en $\log_2(n)$ où n est le nombre de noeuds dans l'arbre.

10) Proposez un algorithme de tri basé sur un tas binaire. Quelle est sa complexité en temps ? en espace ? pouvez-vous trouver une solution pour trier le tableau «sur place» , *i.e.* sans utilisation d'un tableau annexe ?

On prend les éléments du tableau non trié un par un, et on les ajoute à un tas initialement vide (stockée sur un deuxième tableau). Cette opération a une complexité : $\mathcal{O}(nlogn)$, car nous faisons n opérations dont chacune est en $\mathcal{O}(logn)$. On obtient un tas binaire.

On extrait ensuite tous les éléments, du plus prioritaire au moins prioritaire, et on remplit le tableau de la dernière à la première case. Chacune des suppressions prend $\mathcal{O}(logn)$. La construction du tableau trié à partir du tas prend donc $\mathcal{O}(nlogn)$. En espace, on a besoin du tas annexe : $\mathcal{O}(n)$ en mémoire.

On peut tout faire sur place (construction du tas, puis utilisation du tas pour remplir le tableau de la dernière à la première case). O(1) en mémoire.