Fiche TD7 - Partiel d'algorithmique 2003

Remarque : sauf indication contraire, le terme *complexité* dans les exercices suivants désignera la *complexité dans le pire des cas.*

Exercice 1 - Le grand saut

Le problème est de déterminer à partir de quel étage d'un immeuble, sauter par une fenêtre est fatal. Vous êtes dans un immeuble à n étages (numérotés de 1 à n) et vous disposez de k étudiants. Il n'y a qu'une opération possible pour tester si la hauteur d'un étage est fatale : faire sauter un étudiant par la fenêtre. S'il survit, vous pouvez le réutiliser ensuite, sinon vous ne pouvez plus.

Vous devez proposer un algorithme pour trouver la hauteur à partir de laquelle un saut est fatal (renvoyer $n+1$ si on survit encore en sautant du n-ème étage) en faisant le minimum de sauts.

1 - Si $k \geq \lceil \log_2(n) \rceil$, proposer un algorithme en $\mathcal{O}(\log_2(n))$ sauts.

2 - Si $k < \lceil \log_2(n) \rceil$, proposer un algorithme en $\mathcal{O}(k + \frac{n}{2^{k-1}})$ sauts.

3 - Si $k = 2$, proposer un algorithme en $\mathcal{O}(\sqrt{n})$ sauts.

Exercice 2 - Plus grand carré de 1

Donner un algorithme de programmation dynamique pour résoudre le problème suivant :

Entrée : une matrice A de taille $n \times m$ où les coefficients valent 0 ou 1.
Sortie : la largeur maximum K d'un carré de 1 dans A, ainsi que les coordonnées (I, J) du coin en haut à gauche d'un tel carré (autrement dit pour tout i, j, $I \leq i \leq I + K - 1$, $J \leq j \leq J + K - 1$, $A[i, j] = 1$).

Quelle est sa complexité ?

Indication : considérer $t[i, j]$ la largeur du plus grand carré de 1 ayant (i, j) pour coin en haut à gauche.

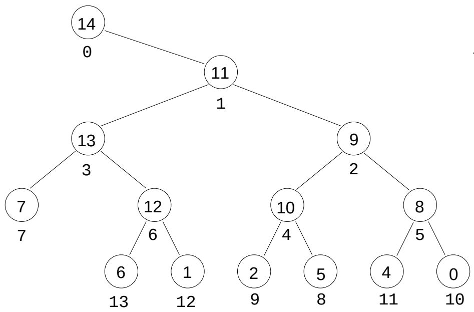
A	1	2	3	4	5	6	7	8	9
1	1	0	0	0	0	0	0	1	0
2	1	0	1	1	1	1	1	1	0
3	1	0	1	1	1	1	0	0	0
4	0	1	1	1	1	1	0	1	0
5	0	0	1	1	1	1	0	0	1
6	0	0	1	1	1	0	0	1	1
7	0	1	0	0	0	0	0	1	1

Figure 1: Exemple où la largeur maximum d'un carré de 1 vaut 4 (avec $(2, 3)$ comme coin en haut à gauche).

Exercice 3 - Tas faibles & tri

Une technique classique de tri par comparaisons consiste à utiliser la structure de données appelée *tas binaire*. Le tri correspondant a une complexité totale en $2n\log_2(n) + O(n)$ comparaisons pour n éléments. L'objectif de cet exercice est d'étudier une variante de cette structure de données, les *tas faibles*, pour améliorer la complexité et se rapprocher de la borne inférieure théorique.

Un *tas faible* pour n éléments est un arbre binaire où chaque nœud est étiqueté par un élément, et qui vérifie les conditions suivantes :
(i) Les éléments du sous-arbre droit de chaque nœud sont inférieurs ou égaux à l'élément associé au nœud.
(ii) La racine de l'arbre n'a pas de fils gauche.
(iii) Les feuilles se trouvent uniquement sur les deux derniers niveaux de l'arbre.



t	14	11	9	13	10	8	12	7	5	2	0	4	1	6
	0	1	2	3	4	5	6	7	8	9	10	11	12	13

r	0	1	0	1	1	1	1	1	1	1	1	1	1	1
	0	1	2	3	4	5	6	7	8	9	10	11	12	13

Figure 2: Exemple de tas faible et un codage de cet arbre.

Pour coder cette structure de données, nous utiliserons deux tableaux $t[0..n{-}1]$ et $r[0..n{-}1]$. Le tableau t contiendra les n éléments et le tableau r est un tableau de bits 0 ou 1 (qui permettra de distinguer fils gauche et fils droit d'un nœud, et ensuite de les permuter à faible coût). Plus précisément, avec le tableau t, le fils gauche de l'élément d'indice i a pour indice $2i + r[i]$ et son fils droit a pour indice $2i + 1 - r[i]$ (on fixe juste $r[0] = 0$ pour que la racine d'indice 0 ait un unique fils qui soit à droite). Changer le bit $r[i]$ revient alors à échanger fils droit et fils gauche de l'élément d'indice i et donc permuter le sous-arbre droit et le sous-arbre gauche de cet élément. La figure 2 présente un exemple de tas faible et de codage de cet arbre.

Les successeurs d'un nœud sur la *branche gauche* du *sous-arbre droit* sont appelés ses *petits-fils*. Par exemple sur la figure 2, les petits-fils de la racine sont les éléments 11, 13 et 7. On note `Pere(`i`)` l'indice du père du nœud d'indice i, et la fonction `Gpere(`i`)` (pour *Grand-père*) définie par `Gpere(Pere(`i`))` si i est le fils gauche de `Pere(`i`)` et par `Pere(`i`)` si i est le fils droit de `Pere(`i`)`. Par exemple avec le codage de la figure 2 le nœud d'indice 9 (de valeur $t[9] = 2$) a pour grand-père `Gpere(9)`$= 1$ (indice du nœud avec la valeur $t[1] = 11$).

1 - Pour le codage de l'arbre avec les tableaux t et r, écrire précisément la fonction `Gpere(`i`)` qui calcule l'indice du grand-père de l'élément d'indice i dans t.

Soit `echange(`i`,`j`)` une fonction qui échange les valeurs de $t[i]$ et $t[j]$. Soit `fusion(`i`,`j`)` la fonction définie par : `si (`$t[i] > t[j]$`) alors echange(`i`,`j`) et` $r[i] \leftarrow 1 - r[i]$.

2 - Etant donné un tableau de n valeurs $t[0..n–1]$, on lui associe le tableau $r = [0, 0, \ldots, 0]$ où tous les bits sont à 0, et on effectue la fonction `ConstruitTas(`t`,`r`)` définie par :

`pour` i `de` $n - 1$ `à 1, faire fusion(`i`,Gpere(`i`))`

Montrer que pendant le déroulement de la boucle, après avoir effectué `fusion(`i`,Gpere(`i`))`, alors pour tout $j \geq i$, les valeurs du sous-arbre de racine le nœud d'indice j sont toutes inférieures ou égales à la valeur du nœud d'indice `Gpere(`j`)`. En déduire qu'en fin de boucle, on obtient un tas faible. Combien de comparaisons a-t-on effectué ?

3 - Etant donné un tas faible représenté par les tableaux t et r, supposons que l'on remplace la valeur $t[0]$ de la racine par une valeur quelconque, comment reconstruire rapidement un tas faible avec cette valeur et à quel coût en nombre de comparaisons ?

Indication : mettre à jour en parcourant les petits fils de la racine.

4 - En déduire un algorithme de tri d'un tableau t en $n \log_2(n) + \mathcal{O}(n)$ comparaisons, qui commence par construire un tas faible puis exploite cette structure de données.

Exercice 4 - Cherchez la star (à ne faire que si vous avez presque fini le reste)

Dans un groupe de n personnes (numérotées de 1 à n pour les distinguer), une *star* est quelqu'un qui ne connait personne mais que tous les autres connaissent. Pour démasquer une star, s'il en existe une, vous avez juste le droit de poser des questions, à n'importe quel individu i du groupe, du type "est-ce que vous connaissez j ?" (noté "$i \to j$?"), on suppose que les individus répondent la vérité. On veut un algorithme qui trouve une star, s'il en existe, ou sinon qui garantit qu'il n'y a pas de star dans le groupe, en posant le moins de questions possibles.

1 - Combien peut-il y avoir de stars dans le groupe ?

2 - Ecrire le meilleur algorithme que vous pouvez et donner sa complexité en nombre de questions (on peut y arriver en $\mathcal{O}(n)$ questions).

3 - Donner une borne inférieure sur la complexité (en nombre de questions) de tout algorithme résolvant le problème. ((*Difficile*) prouver que la meilleure borne inférieure pour ce problème est $3n - \lfloor \log_2(n) \rfloor - 3$).