Partiel d'algorithmique 2004 - 2 heures

Remarque : sauf indication contraire, la notion de *complexité* dans les exercices suivants désignera la *complexité dans le pire des cas* et en *nombre d'opérations élémentaires* (tests, opérations arithmétiques, affectations ...). Les questions avec une étoile * sont plus difficiles.

Exercice 1 - Hors d'œuvre

On considère un ensemble S de $n \geq 2$ entiers distincts stockés dans un tableau (S n'est pas supposé trié). Résoudre les questions suivantes :

1 - Proposer un algorithme en $\mathcal{O}(n)$ pour trouver deux éléments x et y de S tels que $|x - y| \geq |u - v|$ pour tout $u, v \in S$.

2 - Proposer un algorithme en $\mathcal{O}(n \log n)$ pour trouver deux éléments x et y de S tels que $x \neq y$ et $|x - y| \leq |u - v|$ pour tout $u, v \in S$, $u \neq v$.

3 - Soit m un entier arbitraire (pas nécessairement dans S), proposer un algorithme en $\mathcal{O}(n \log n)$ pour déterminer s'il existe deux éléments x et y de S tels que $x + y = m$.

* 4 - Proposer un algorithme en $\mathcal{O}(n)$ pour trouver deux éléments x et y de S tels que $|x - y| \leq \frac{1}{n-1}(\max(S) - \min(S))$.

Exercice 2 - Arbres binaires de recherche optimaux

Un *arbre binaire de recherche* est une structure de données permettant de stocker un ensemble de clés ordonnées $x_1 < x_2 < \ldots < x_n$, pour ensuite effectuer des opérations du type *rechercher*, *insérer* ou *supprimer* une clé. Il est défini par les propriétés suivantes:
(i) C'est un arbre où chaque nœud a 0, 1 ou 2 fils, et où chaque nœud stocke une des clés.
(ii) Etant donné un nœud avec sa clé x, alors les clés de son sous-arbre gauche sont $< x$ et celles de son sous-arbre droit sont $> x$.

La figure 1 représente un arbre binaire de recherche pour les clés $x_1 < x_2 < x_3 < x_4 < x_5 < x_6 < x_7 < x_8 < x_9$.

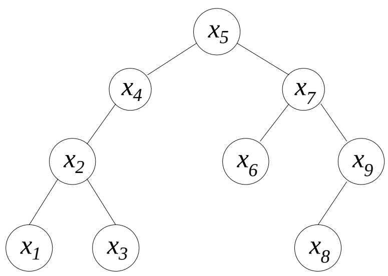

Figure 1: Exemple d'arbre binaire de recherche.

Les requètes auxquelles on s'intéresse ici sont les *recherches* de clés. Le coût de la recherche d'une clé x correspond au nombre de comparaisons effectuées pour la retrouver dans l'arbre binaire de recherche, soit exactement la profondeur de x dans l'arbre, plus 1 (en disant que la racine est de profondeur 0).

Pour une séquence fixée de recherches, on peut se demander quel est l'arbre binaire de recherche qui minimise la somme des coûts de ces recherches. Un tel arbre est appelé *arbre binaire de recherche optimal* pour cette séquence.

1 - Pour un arbre binaire de recherche fixé, le coût de la séquence ne dépend clairement que du nombre de recherches pour chaque clé, et pas de leur ordre. Pour $n = 4$ et $x_1 < x_2 < x_3 < x_4$, supposons que l'on veuille accèder une fois à x_1, 9 fois à x_2, 5 fois à x_3 et 6 fois à x_4. Trouver un arbre binaire de recherche optimal pour cette ensemble de requètes.

2 - Donner un algorithme (METTRE "DE PROG DYN" ???) en temps $\mathcal{O}(n^3)$ pour construire un arbre binaire de recherche optimal pour une séquence dont les nombres d'accès aux clés sont $c_1, c_2, \ldots, c_n$ (c_i est le nombre de fois que x_i est recherché). Justifier sa correction et sa complexité.

Indication : pour $i \leq j$, considérer $t[i,j]$ le coût d'un arbre de recherche optimal pour les clés $x_i < ... < x_j$ accédées respectivement $c_i, \ldots, c_j$ fois.

* 3 - Pour tout $i \leq j$, soit $r[i,j]$ l'*indice* de la racine d'un arbre de recherche optimal pour les clés $x_i < ... < x_j$ accédées respectivement $c_i, \ldots, c_j$ fois (s'il existe plusieurs arbres optimaux, $r[i,j]$ est arbitrairement choisi parmi eux). Montrer que, pour tout $1 \leq i < j \leq n$, on a : $r[i,j-1] \leq r[i,j] \leq r[i+1,j]$.

* 4 - En déduire un algorithme en temps $\mathcal{O}(n^2)$ pour construire un arbre binaire de recherche optimal.

Exercice 3 - Bricolage

Dans une boite à outils, vous disposez de n écrous de diamètres tous différents et des n boulons correspondants. Mais tout est mélangé et vous voulez réappareiller chaque écrou avec le boulon qui lui correspond. Les seules opérations autorisées consistent à comparer un écrou avec un boulon, ce qui peut amener trois réponses possibles : soit l'écrou est strictement plus large que le boulon, soit il est strictement moins large, soit ils ont exactement le même diamètre. Les différences de diamètre entre les écrous sont tellement minimes qu'il n'est pas possible de déterminer à l'œil nu si un écrou est plus grand qu'un autre. Il en va de même avec les boulons.

1 - Ecrire un algorithme simple en $\mathcal{O}(n^2)$ comparaisons qui réappareille chaque écrou avec son boulon.

2 - Supposons qu'au lieu de vouloir réappareiller tous les boulons et écrous, vous voulez juste trouver le plus petit écrou et le boulon correspondant. Montrer que vous pouvez résoudre ce problème en moins de $2n - 2$ comparaisons.

3 - Prouver que tout algorithme qui réappareille tous les écrous avec tous les boulons doit effectuer $\Omega(n \log n)$ comparaisons dans le pire des cas.

Problème ouvert : proposer un algorithme en $o(n^2)$ comparaisons pour résoudre ce problème.