# Algorithmique TD3

Magistère Informatique 1ère Année

2 Octobre 2008

**Exercice 1 (Anciennement Exercice 7 du TD2)** Une *matrice de Toeplitz* est une matrice $n \times n(a_{i,j})$ telle que $a_{i,j} = a_{i-1,j-1}$ pour $2 \leq i, j \leq n$.

1. Que peut-on dire de la somme et du produit de deux *matrices de Toeplitz* ?
2. Donner un algorithme qui additionne les *matrices de Toeplitz* en $O(n)$.
3. Comment calculer le produit d'une *matrice de Toeplitz* $n \times n$ par un vecteur de longueur $n$ ? Donner la complexité de l'algorithme.

**Exercice 2 (Anciennement Exercice 8 du TD2)** On dispose d'un tableau d'entiers $T$ de taille $n$, on cherche à déterminer le $k^{\text{ième}}$ élément de $T$ c'est à dire l'élément qui se trouverait en position $k-1$ si on triait $T$. Un algorithme naïf est de commencer par trier le tableau puis de sélectionner l'élement voulu. La complexité d'un tel algorithme est en $O(nlog(n))$, le but de l'exercice est de montrer qu'il existe un algorithme en temps linéaire s'inspirant du tri rapide. On dispose d'une fonction $PIVOT(T, i, j)$ qui étant donné une table $T$ et deux positions $i$ et $j$, réorganise les éléments entre $i$ et $j$ autour d'un pivot $k$ qu'elle retourne. On ne donne pour l'instant aucune précision sur la façon dont le pivot est choisi.

1. Utiliser cette fonction $PIVOT$ pour écrire un algorithme $SELECT$ traitant le problème.
2. Montrer que sans hypothèses sur la façon dont la fonction $PIVOT$ choisit l'élément, l'algorithme $SELECT$ peut effectuer jusqu'à $O(n^2)$ comparaisons.
3. Montrer que si le choix du pivot garantit que la taille des deux sous-tableaux construits autour du pivot ne dépasse pas $\alpha n$ où $\alpha < 1$, le nombre de comparaisons effectuées par l'algorithme $SELECT$ est $O(n)$.
4. On se concentre maintenant sur le choix du pivot. Considérer l'algorithme de choix du pivot suivant :
   (a) Découper le tableau en $\lfloor \frac{n}{5} \rfloor$ blocs $\{B_1, ..., B_{\lfloor \frac{n}{5} \rfloor}\}$, de cinq éléments. Les éléments restants (au plus 4) sont ignorés par la suite.
   (b) Déterminer les éléments médians des $B_k$, $\{m_1, ..., m_{\lfloor \frac{n}{5} \rfloor}\}$.
   (c) Utiliser la fonction $SELECT$ pour déterminer l'élément d'ordre $\lfloor \frac{n+5}{10} \rfloor$ de la liste $\{m_1, ..., m_{\lfloor \frac{n}{5} \rfloor}\}$ des médians.

1. Montrer que le pivot choisi est strictement supérieur à au moins $3 \lfloor \frac{n-5}{10} \rfloor$ éléments de $T$ et inférieur ou égal à au moins $3 \lfloor \frac{n-5}{10} \rfloor$. En déduire que pour $n \geq 75$ la taille des sous-tableaux construits autour du pivot est au plus égale à $\frac{3n}{4}$.
2. Ecrire un algorithme $SELECT2$ qui résout le problème en $O(n)$ comparaisons.

**Exercice 3** *Files*

Une file est une liste linéaire où les insertions se font toutes d'un même côté et les suppressions toutes de l'autre côté (à l'inverse des piles dans lesquelles insertions et suppressions sont faites du même côté).

1. Définir le type abstrait file
2. Implémenter une file à l'aide de deux piles et calculer le coût amorti d'une opération.

**Exercice 4** *Expressions arithmétiques*

1. La grammaire des expressions arithmétiques en notation préfixe est :

   ```
   EXP := REEL | OP_BIN EXP EXP | OP_UN EXP | (EXP)
   ```

   – Ecrire une fonction récursive $EVAL$ pour évaluer une expression en notation préfixe.
   – Donner une version itérative de la fonction $EVAL$.
2. Considérons maintenant les expressions en notation infixe données par la grammaire :

   ```
   EXP := REEL | EXP OP_BIN EXP | OP_UN EXP | EXP
   ```

   Ecrire une fonction qui construit l'arbre représentant une expression infixe en respectant les règles de priorité et d'associativité.

**Exercice 5** *Arbres Binaires de Recherche*

1. Ecrire une fonction $SUPPRIMER(ABR)$ qui prend en argument une clé $k$ et un arbre binaire de recherche $t$, et qui supprime $k$ de $t$.
2. Ecrire une fonction $SCINDER(t, x)$ qui prend en argument un arbre binaire de recherche $t$ et une clé $k$ et qui retourne une décomposition $(t_1, t_2)$ de $t$ telle que :

$$\begin{aligned} c(t) &= c(t_1) \cup c(t_2) \\ c(t_1) &= \{c \in c(t) | c \leq k\} \\ c(t_2) &= \{c \in c(t) | c > k\} \end{aligned}$$

3. Ecrire une fonction

$$CONCATENER(t_1, t_2)$$

   qui prend en argument deux arbres binaires de recherche $t_1$ et $t_2$, tels que :

$$max(c(t_1)) < min(c(t_2))$$

   et qui retourne un arbre binaire de recherche $t$ tel que :

$$c(t) = c(t1) \cup c(t2)$$

**Exercice 6** *Arbres Binaires de Recherche Balisés*

Dans un ABR balisé, les clés ne se trouvent qu'aux feuilles et les noeuds internes ont tous exactement 2 fils et contiennent des balises qui permettent d'orienter la recherche. Si $x$ est un noeud interne, alors :

$$\{c(x.g)\} < x.balise \leq \{c(x.d)\}$$

1. Montrer que $n(a) = 2f(a) - 1$ si $a$ est un ABR balisé.
2. Ecrire les procédures d'insertion et de suppression dans un ABR balisé.
3. Ecrire une procédure pour transformer en temps linéaire un ABR balisé en un ABR classique ayant la même structure.
4. Ecrire la procédure inverse.

**Exercice 7** *Arbres binomiaux*

On définit une suite d'arbres $(B_n)_{n>0}$ par réccurrence sur $n$ comme suit : $B_0$ est l'arbre à un seul sommet, $B_{n+1}$ est formé de la réunion de deux copies disjointes de $B_n$, $B_n^g$ et $B_n^d$. La racine de $B_{n+1}$ est celle de $B_n^d$, la racine de $B_n^g$ étant le fils le plus à gauche de celle-ci. L'arbre $B_n$ est l'arbre binomial d'ordre $n$.

1. Démontrer que dans $B_n$, il y a exactement $C_n^k$ sommets de profondeur $k$.
2. Démontrer qu'un seul sommet de $B_n$ a $n$ fils, et que pour $0 \leq k < n$, $2^{n-k-1}$ sommets ont $k$ fils.
3. On numérote les sommets de $B_n$ de 0 à $2n-1$ de la gauche vers la droite en ordre postfixe (fils avant le père). Montrer qu'un sommet est de profondeur $n-k$ si et seulement si son numéro a $k$ "1" en écriture binaire.
4. Montrer que le nombre de fils d'un sommet est égal au nombre de "1" qui suivent le dernier "0" dans l'écriture binaire de son numéro.

**Exercice 8** *Hauteur moyenne d'un ABR*

1. Calculer la hauteur moyenne d'un ABR construit aléatoirement.
   Univers : Permutations de taille $n$ avec distribution uniforme.
   Hauteur : Pour une permutation $\sigma$, hauteur de l'arbre obtenu par insertions successives de $\sigma(1), ..., \sigma(n)$.
2. Calculer le coût moyen d'une recherche dans un ABR construit aléatoirement.

**Exercice 9** Etant donné un alphabet $\Sigma$ ordonné, on définit l'ordre lexicographique sur $\Sigma^*$ comme suit : Soit $w = a_0a_1...a_p$ et $w' = b_0b_1...b_q$, $w < w'$ si et seulement si $w$ est un prefixe de $w'$ ou $\exists j$ tel que $\forall i < j$ $a_i = b_i$ et $a_j < b_j$.

La structure d'arbre de base peut contenir des mots écrits sur un alphabet à deux lettres. Par exemple l'arbre ci-dessous contient les mots *bab*, *ba*, *baa*, *bbb* et *a*. Soit $\mathcal{L}$ un langage fini sur un alphabet à deux lettres dont la somme des longueurs des mots vaut $n$. Montrer qu'on peut utiliser la structure d'arbre de base pour trier le langage lexicographiquement en temps $O(n)$.