# Devoir Maison Algorithmique
# Recherche dichotomique

Guillaume Bury

A rendre lors du TD du 8 novembre, rédigé à la main sur papier.

Les deux exercices sont indépendants et peuvent être traités séparément.

**Algorithmes** Tout algorithme doit être présenté en pseudo-code (comme vu en cours), et accompagné d'une démonstration de sa correction, terminaison, et complexité, ce qui est parfois explicitement demandé dans des questions séparées.

## 1 Arbres équilibrés

On considère dans cet exercice des arbre binaires de recherche. En notant $\text{size}(x)$ le nombre de nœuds d'un arbre enraciné en $x$, $\text{left}(x)$ son fils gauche et $\text{right}(x)$ son fils droit, on dit qu'un noeud est *$\alpha$-équilibré* si :

$$\text{size}(\text{left}(x)) \leq \lceil \alpha(\text{size}(x) - 1) \rceil$$

et

$$\text{size}(\text{right}(x)) \leq \lceil \alpha(\text{size}(x) - 1) \rceil$$

Un arbre binaire est dit $\alpha$-équilibré ssi tout nœud de l'arbre est $\alpha$-équilibré.

1. Étant donné un noeud $x$ d'un arbre, donner un algorithme pour rééquilibrer le sous-arbre enraciné en $x$ afin d'obtenir un sous-arbre $\frac{1}{2}$-équilibré. L'algorithme devra avoir une complexité en temps et en espace linéaire en le nombre d'éléments du sous-arbre enraciné en $x$
2. Donner un algorithme de recherche dans un arbre binaire $\alpha$-équilibré. Montrer que cet algorithme a une complexité en temps logarithmique en le nombre d'éléments de l'arbre.

On considère dans le reste du problème un $\alpha > \frac{1}{2}$. Lors de l'ajout et de la suppression d'éléments, on procède comme dans un arbre binaire de recherche habituel, sauf dans le cas ou l'on se retrouverait avec un sous-arbre qui n'est plus équilibré, auquel cas, on rééquilibre le sous-arbre maximal non équilibré avec la procédure de la question 1.

3. Implémenter l'algorithme d'insertion et de suppression d'un élément, et montrer qu'il conserve le $\alpha$ équilibrage d'un arbre.

On définit maintenant :

$$\Delta(x) = |\text{size}(\text{left}(x)) - \text{size}(\text{right}(x))|$$

et le potentiel d'un arbre par :

$$\Phi(t) = c \sum_{x \in T : \Delta(x) \geq 2} \Delta(x)$$

où $c$ est une constante qu'il faudra déterminer.

4. Montrer qu'un arbre $\frac{1}{2}$-équilibré possède un potentiel nul.

5. Supposons que le rééquilibrage d'un arbre à $m$ éléments un coût en temps exactement égal à $m$. Démontrer et trouver une valeur de $c$ telle que le coût amorti du rééquilibrage d'un arbre non équilibré soit en $O(1)$.
6. Montrer que la l'ajout et la suppression d'un élément dans un arbre $\alpha$-équilibré a une complexité amortie en $O(\log(n))$ ou $n$ est le nombre d'éléments de l'arbre.

## 2 Tableaux et recherche dichotomique

Cet exercice a pour but d'étendre la structure de tableau trié, usuellement utilisée pour faire de la recherche dichotomique, afin de supporter des opérations d'insertion et de suppression efficaces.

**Tableaux et notations.** Dans le reste de l'exercice, on utilisera les conventions suivantes. Étant donné un tableau $T$, on note sa taille $|T|$, et ses éléments, indexés de 0 à $|T|-1$, et notés $T(0), \ldots, T(|T|-1)$, sont accessibles en temps constant. On considérera le coût d'allocation des tableaux comme constant quelle que soit la taille du tableau alloué. On pourra aussi utiliser la syntaxe d'OCaml pour décrire des tableaux de taille constante, par exemple $[|0;1;2;3|]$.

**Éléments d'un tableau.** On considérera les éléments contenus dans les tableaux comme abstraits, i.e. ces éléments peuvent seulement être déplacés, copiés, et comparés d'après un ordre total fourni.

Dans un premier temps, on s'intéresse au problème suivant. Étant donné un élément $a$, et $k$ tableaux $T_0, \ldots, T_{k-1}$, chacun trié et tels que $|T_i| = 2^i$. On veut créer le tableau trié $T$, tel que $|T| = 2^k$, résultat de la fusion des tableaux $[|a|], T_0, \ldots, T_{k-1}$, en temps linéaire en le nombre total d'éléments, soit $O(2^k)$.

1. Donner un algorithme qui effectue cette fusion en temps linéaire (et prouver sa correction).
2. Démontrer la complexité de l'algorithme.

Afin de supporter l'insertion d'éléments, on va considérer la structure de données suivante. Pour stocker $n$ éléments, considérons $n = \sum_{i=0}^{i<=\lfloor \log(n) \rfloor} b_i 2^i$, sa décomposition en binaire, alors on utilise au plus $\lfloor \log(n) \rfloor$ tableaux $T_0, \ldots, T_{\lfloor \log(n) \rfloor}$ avec $|T_i| = b_i 2^i$ (certains tableaux ont donc une taille nulle), chaque tableau étant trié.

3. Implémenter un algorithme de recherche dichotomique sur cette structure de données. Calculer sa complexité.
4. Implémenter l'insertion d'un élément dans la structure. Quelle est sa complexité dans le pire des cas ?
5. Étant donné un état $T_0, \ldots, T_k$ de notre structure, on notera $c(T_0, \ldots, T_i) = \sum_{j=0}^{i} |T_i|$. On définit maintenant un potentiel comme suit : $p(T_0, \ldots, T_k) = c_0 * \sum_{i=0}^{k} c(T_0, \ldots, T_i)$. Trouver une valeur de $c_0$ pour que l'insertion d'un élément soit en coût amorti logarithmique.
6. En gardant la même structure et représentation des données, proposer (à haut niveau) un algorithme pour supprimer un élément. Quelle est sa complexité dans le pire des cas ?
7. Montrer qu'avec cet algorithme pour la suppression d'élément, il est impossible d'avoir un coût amorti logarithmique pour l'ajout et la suppression d'éléments.

On veut maintenant supporter efficacement la suppression d'éléments. A cet effet, on va changer notre représentation pour autoriser les tableaux à contenir des valeurs fantômes. Une valeur fantôme représente une valeur qui a été éliminée de la structure, mais la valeur de la clé est conservé afin d'aider la recherche dichotomique. On maintiendra aussi le fait que dans un tableau, lorsqu'il y a plusieurs fois la même clé $x$, toutes les instances fantômes de $x$ sont apparaissent avant les instances non fantômes (i.e. les indices des instances fantômes sont plus petits que ceux des instances non fantômes). On utilise donc maintenant des tableaux ou chaque élément est une paire d'une clé, et d'un booléen, qui indique si la valeur est fantôme. (en termes OCaml, on utilise des "('a * bool) array"). On note maintenant $||T||$ le nombre d'éléments effectivement stocké dans

un tableau, qui est égal à la taille du tableau moins le nombre de valeurs fantômes présentes. On défini aussi $s(T_0, \ldots, T_i) = \sum_{j=0}^{i} ||T_j||$. Afin d'éviter de se retrouver avec des tableaux tous presque vides, on rajoute l'invariant suivant : pour tout $i$, on impose que $s(T_0, \ldots, T_i) > 2^{i-1}$. Afin d'aider à maintenir cet invariant, on se souviendra, pour chaque tableau, du nombre effectif de ses éléments (i.e $||T||$ est calculable en temps constant).

Notons que maintenant, il existe plusieurs manières différentes de représenter un ensemble de $n$ éléments. Par exemple pour $n = 7$, on pourrait avoir : $T_1 = [|3_{\text{ghost}}; 6|], T_3 = [|1; 2; 3_{\text{ghost}}; 3; 4; 5; 7_{\text{ghost}}; 8|]$ aussi bien que $T_3 = [|1; 2; 3; 4; 5; 6; 7_{\text{ghost}}; 8|]$.

6. Implémenter le nouvel algorithme de recherche dichotomique (qui doit prendre en compte la présence des valeurs fantômes), et prouver sa complexité.
7. Implémenter l'ajout d'un élément, et calculer sa complexité dans le pire des cas.
8. Implémenter la suppression d'un élément, et calculer sa complexité dans le pire des cas. Attention à bien maintenir les invariants de la structure.
9. Prouver que l'insertion d'un élément est en coût amorti logarithmique, et la suppression en coût amorti $O(\log^2(n))$.