Devoir à la maison

à rendre la semaine du 14 novembre

Ce devoir est à rendre en TD la semaine du 14 novembre. Il est composé de 4 parties indépendantes (numérotées de 1 à 4). **Attention : vous devez rendre les solutions de ces 4 parties sur des copies séparées !**

Il est autorisé de réfléchir en groupe aux solutions. En revanche, la **rédaction** doit être **personnelle**.

1 Arbres AVL

Un arbre AVL (structure proposée par G. Adelson-Velskii et E. M. Landis) est un arbre binaire de recherche équilibré en hauteur : pour chaque noeud x les hauteurs des sous-arbres gauche et droite de x ne diffèrent que de 1 au plus. $A(x)$, $G(x)$ et $D(x)$ désignent respectivement l'arbre correspondant à x, le sous-arbre gauche et le sous-arbre droit. $p(x)$ est le père de x et $val(x)$ est le nombre stocké dans le nœud x. Dans l'implémentation d'un AVL, pour chaque noeud x il y a un champ supplémentaire : $h(x)$ la hauteur du sous-arbre enraciné en x.

Exercice 1 Soit h une hauteur, décrivez les AVL de hauteur h ayant le plus de noeuds et le moins de noeuds. Combien y a-t-il de noeuds dans ces deux cas ?

Exercice 2 En déduire un encadrement de la hauteur h d'un AVL en fonction de n le nombre de noeuds.

L'insertion et la suppression d'un noeud se font d'abord comme dans un arbre binaire de recherche classique ; mais il se peut que la propriété d'équilibre ne soit plus respectée et il faudra alors réorganiser l'arbre, dans un deuxième temps.

Exercice 3 Quelle propriété sur les hauteurs a-t-on si $A(x)$ était un arbre AVL et ne l'est plus suite à l'insertion d'un nouveau noeud ?

Exercice 4 En déduire que si $D(x)$ et $G(x)$ sont équilibrés et de même hauteur, alors après insertion d'un nouveau noeud y, $A(x)$ est toujours équilibré.

Pour rétablir l'équilibre lorsque nécessaire on fait ce qu'on appelle des rotations.

Exercice 5 Soit x un noeud tel que $A(x)$ est équilibré, gx son fils gauche et dx son fils droit. Après insertion d'un nouveau noeud y on suppose ici que le sous-arbre enraciné en x est déséquilibré mais que ceux de gx et dx sont toujours équilibré, de plus on supposera pour simplifier que c'est $A(gx)$ qui est devenu trop grand, le cas $D(x)$ est symétrique.

- Supposons que y est dans $G(gx)$. À l'aide d'un dessin, expliquez comment on peut réorganiser les noeuds x et gx et les sous arbres $G(gx)$, $D(gx)$ et $D(x)$, de sorte que l'arbre obtenu soit un AVL.
- Supposons que y est dans $D(gx)$. De même expliquez la réorganisation entre les noeuds x, gx et dgx (le fils droit de gx) et des sous arbres $G(gx)$, $G(dgx)$, $D(dgx)$ et $D(x)$ tel que le nouvel arbre soit un AVL. Notez que y est allé soit dans $G(dgx)$ soit dans $D(dgx)$.

Justifiez que vos transformations donnent bien un arbre AVL.
Indication : L'arbre AVL obtenu doit toujours être un ABR.

Exercice 6 Donner le pseudo-code la fonction `Equilibre` correspondante qui prend en entrée un sous-arbre vérifiant la propriété de l'exercice 3 et dont les deux sous arbres, gauche et droit, sont eux équilibrés.

Exercice 7 Donnez les arbres correspondant à l'insertion de 14,3,1,5,4,8,9,12 à partir d'un arbre vide ainsi que les étapes intermédiaires annotées pour indiquer qu'elles sont les rotations effectuées. **Indication :** Ne détaillez pas toute l'insertion d'un nouveau noeud dans un ABR.

Exercice 8 Montrez qu'une seule rotation est suffisante lors de l'insertion d'un nouveau noeud.

Exercice 9 À l'aide la fonction `Equilibre`, décrivez puis donner le pseudo-code de l'opération `Insertion` qui insère nouveau noeud dans un arbre AVL.
Indication : Pensez à mettre à jour la hauteur quand nécessaire.

Exercice 10 Quelle est la complexité de cette opération ? Indiquez ce que vous considérez comme opération de base pour le calcul de la complexité.

2 La bâtière

Une **bâtière** désigne un tableau à deux dimensions de nombre réels tel que
– chaque ligne est triée en ordre croissant
– chaque colonne est aussi triée en ordre croissant
On note n le nombre de lignes et m le nombre de colonnes.

Certaines cases contiennent la valeur $+\infty$ et sont appelées les **cases vides**. Il n'y a pas d'autres valeurs spéciales : les cases contenant des réels ordinaires sont dites pleines.

Exemple : le tableau ci-dessous est une bâtière à 4 lignes et 5 colonnes. Il y a 4 cases vides.

2	14	25	30	69
3	15	28	30	81
7	15	32	43	$+\infty$
20	28	$+\infty$	$+\infty$	$+\infty$

Le but de ces exercices est d'utiliser une bâtière avec des méthodes ressemblant à celles des tas, afin d'écrire un algorithme `tri-bâtière` et de le comparer au `tri-par-tas`. On compare aussi l'efficacité de la recherche d'un élément à celle dans un tableau trié à une seule dimension.

Notations

Dans cette section 2, B désigne toujours une bâtière à n lignes et m colonnes$n \times m$. Les numéros de ligne vont de 1 à n et ceux de colonne de 1 à m. On accède à une case via `B[i][j]`, et aux dimensions par `B.n` et `B.m` (pour réduire le nombre de paramètre des méthodes). Enfin `B.p` donne le nombre de cases pleines de la matrice.

On appelle **adja-échange** l'opération consistant à échanger deux éléments contigus, c'est-à-dire :
– soit situés en (x, y) et en $(x, y + 1)$ (horizontalement)
– soit situés en (x, y) et en $(x + 1, y)$ (verticalement)

Exercice 11 Donnez une fonction `LireMinimum(B : bâtière)` retournant le minimum de la bâtière. Analysez sa complexité. Attention, il peut y avoir égalité entre éléments.

Exercice 12 Supposons que dans une bâtière **un** élément soit incorrect, situé ligne i et colonne j. C'est-à-dire que toutes les lignes et colonnes sont triées, sauf éventuellement la ligne i et la colonne j. Proposez un algorithme qui rétablisse la correction uniquement en faisant des adja-échanges. Vous l'appellerez `Rétablir(B,i,j)` où i et j sont la position de la case erronée et `B` la bâtière modifiée par effet de bord.

Exercice 13 Montrez que seulement deux directions, parmi les quatre possibles (haut, bas, gauche, droite) sont utilisés par `Rétablir(B,i,j)` jusqu'à correction de l'erreur (pour une instance donnée). En déduire la complexité de cet algorithme.

Exercice 14 Donnez une méthode `ExtraireMinimum(B : bâtière)` qui non seulement renvoie le minimum de la bâtière, mais le supprime, en le remplaçant par une case vide (une valeur $+\infty$). Il faut maintenir les propriétés de la bâtière. Les fonctions des exercices précédents peuvent servir !

Exercice 15 Supposons que `B` ne soit pas complètement pleine (`B.p`$<$ `B.n`$*$ `B.m`). Proposez une méthode `Inserer(B : bâtière, x : réel)` qui insère l'élément `x` dans la bâtière. En respectant l'ordre des éléments bien sûr ! Là aussi vous pouvez utiliser `Rétablir( )`.

Exercice 16 Avec le même argument que l'exercice 13 donnez la complexité de votre méthode

On donne un algorithme `tri-bâtière` qui ressemble au tri par tas. La donnée est un tableau `T` de n^2 réels et qui est modifié à la fin.

B : bâtière $n \times n$ initialisée avec des $+\infty$;
pour *i de 1 à* n^2 **faire**
| `Inserer(B, T[i]);`
pour *i de 1 à* n^2 **faire**
| `T[i]=ExtraireMin(B);`

Exercice 17 Prouvez que cet algorithme trie T en ordre croissant.

Exercice 18 Donnez sa complexité. Comparez avec celle de quelques algorithmes de tri connus comme QuickSort, Tri-Bulles, Tri-par-tas

Exercice 19 Donnez une méthode `Rechercher(B : bâtière, x : réel)` qui renvoie les coordonnées d'une case de valeur `x` dans la bâtière s'il y en a une, ou $(0, 0)$ sinon. Essayez de faire une complexité la plus petite possible.

Exercice 20 La recherche dichotomique donne un élément en $O(\log n)$ dans un tableau trié à n éléments. Pourquoi dans la bâtière tout algorithme de recherche est en $\Omega(\min(n, m))$ (c'est à dire qu'il faut un temps au moins $(\min(n, m))$?

3 Arbres binomiaux

On définit une suite $(B_n)_{n\geq 0}$ d'arbre ordonnés par récurrence 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 $B_n^{(g)}$ et $B_n^{(d)}$ de l'arbre B_n ; sa racine est la racine de $B_n^{(d)}$, et il y a un arc de la racine de $B_n^{(d)}$ vers la racine de $B_n^{(g)}$. Cet arc est le plus à gauche des arcs issus de la racine. L'arbre B_n est *l'arbre binomial d'ordre n*.

Exercice 21 Démontrer que dans B_n, il y a exactement $\binom{n}{k}$ sommets de profondeur k.

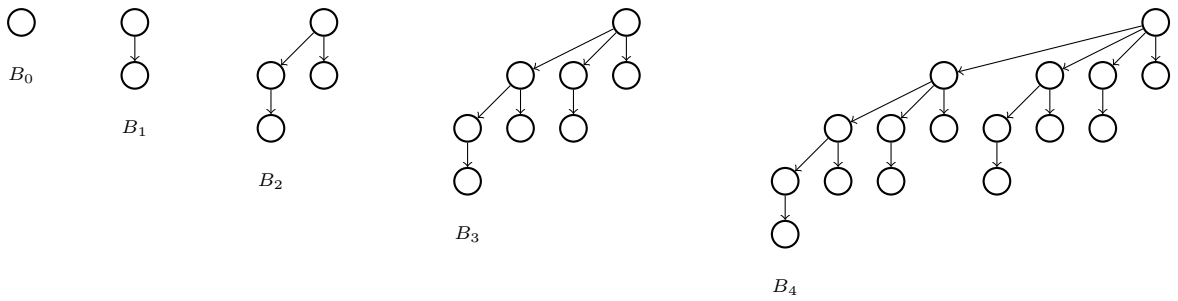

FIGURE 1 – Arbres binomiaux.

Exercice 22 Démontrer qu'un seul sommet de B_n a n fils, et que 2^{n-k-1} sommets ont k fils pour $0 \leq k < n$.

On numérote les sommets de B_n de 0 à $2^n - 1$ de la gauche vers la droite en ordre postfixe.

Exercice 23 Montrer qu'un sommet est de profondeur $n-k$ si et seulement si son numéro a k de chiffres "1" en écriture binaire.

Exercice 24 Montrer que le nombre de fils d'un sommet est égal au nombre de chiffres "1" qui suivent le dernier "0" dans l'écriture binaire de son numéro.

Etant donné un entier n, soit

$$n = \sum_{i \geq 0} b_i \cdot 2^i, \quad b_i \in \{0, 1\}$$

sa decomposition en base 2, soit $I_n = \{i \mid b_i = 1\}$ (l'ensemble d'indices i tel que $b_i = 1$), et soit $\nu(n)$ le nombre d'elements de l'ensemble I_n.

La forêt binomiale d'ordre n, dénotée F_n, est l'ensemble de tous les arbres B_i tel que $i \in I_n$. La composante d'indice i de F_n est B_i, si $i \in I_n$, et $\emptyset$ sinon.

Exercice 25 Montrer que F_n a $n - \nu(n)$ arêtes.

Une file binomiale est une forêt binomiale dont chaque sommet x est muni d'une clé entière $c(x)$ vérifiant : si y est fils de x, alors $c(y) > c(x)$.

Exercice 26 Montrer que la recherche du sommet de clé minimale dans une file binomiale F_n peut se faire en $\nu(n) - 1$ comparaisons.

4 Tournois

Soit $n \geq 2$ un entier et V l'ensemble $\{1, 2...n\}$. Un **tournoi** à n éléments est un cas particulier de graphe orienté où il y a toujours un (et un seul) arc orienté entre deux éléments différents de V. Formellement c'est une relation binaire R

- antireflexive : $\forall x \; x \not R y$
- antisymmétrique : $\forall x \neq y$
  - soit xRy et $y \not R x$
  - soit yRx et $x \not R y$

Un **cycle** est une suite $x_1...x_k$ de $k \geq 3$ éléments tous différents tels que $x_k R x_1$ et pour tout $i \in [1, k-1]$, $x_i R x_{i+1}$. Un tournoi est un **ordre total** si on peut numéroter tous ses éléments de x_1 à x_n tel que $x_i R x_j$ si et seulement si $i < j$.

Exercice 27 Montrez que tout tournoi
– Soit est un ordre total
– Soit contient un cycle

Indication : On peut regarder le degré (nombre d'arcs sortants) pour chaque élément

Exercice 28 Proposez un algorithme qui, étant donné un tournoi sur n éléments,
– Soit exhibe un cycle de k éléments. Par exemple il affiche :
`Ce tournoi contient le cycle 3,8,12,2`
– Soit donne l'ordre total si ce tournoi en est un. Par exemple il affiche :
`Ce tournoi est l'ordre total 3,5,9,11,8,4,10,1,12,5,7,2`
Vous préciserez comment la relation R doit être stockée en mémoire. Prouvez que votre algorithme donne le résultat attendu. Et pour terminer, donnez sa complexité !

Indication : un algorithme de parcours de graphe vous donne la réponse. En un seul parcours vous pouvez savoir si vous êtes dans le cas «cycle» ou «ordre total», et afficher ce qu'il faut. Pensez au théorème de classification des arcs de parcours. Mais il y a une autre façon de procéder rien qu'en regardant les degrés de chaque sommet.