

Algorithmique et Programmation

TD n° 1 : Arbres

École normale supérieure – Département d’informatique
 algoL3@di.ens.fr

2016-2017

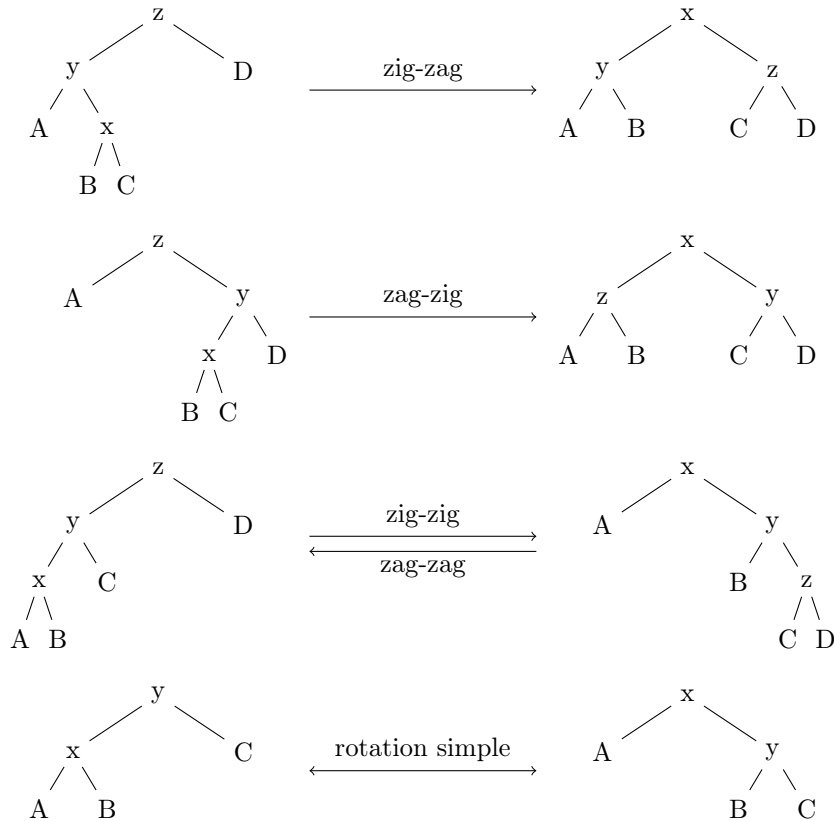
Exercice 1. ARBRES SPLAY

Les arbres binaires de recherche supportent typiquement les opérations suivantes :

1. RECHERCHER(T, x) renvoie vrai ou faux selon que la clé x est présente dans T .
2. INSÉRER(T, x) ajoute la clé x à T (en supposant qu’elle n’y est pas déjà).
3. SUPPRIMER(T, x) retire la clé x de T

Nous supposons dans la suite que toutes les clés sont distinctes.

Dans un arbre binaire de recherche, une opération $\text{SPLAY}(T, x)$ remonte un noeud arbitraire x dans l’arbre jusqu’à la racine par une séquence de doubles rotations (qui se termine éventuellement par une simple rotation à la fin si la profondeur de x était impaire). Les opérations qui peuvent avoir lieu sont illustrées ici (les lettres minuscules sont des noeuds et majuscule des sous arbres, possiblement vide) :



Après *chaque* opération RECHERCHER(T, x), INSÉRER(T, x), SUPPRIMER(T, x), ..., on remonte x vers la racine par un *splay*. L’arbre peut donc être modifié par RECHERCHER, ce qui est inhabituel. On impose que si RECHERCHER(T, x) renvoie “Faux”, alors l’opération *splay* est appliquée au dernier noeud visité.

1. Un *peigne* est un arbre obtenu à partir d’un chemin en ajoutant un fils à tous les noeuds du chemin. Dans un arbre *splay*, quel est l’effet d’appliquer l’opération *splay* au noeud du bas d’un peigne de hauteur 3, 4 puis n pour n un entier arbitraire? Quel est l’effet d’appliquer uniquement des rotations simples sur un peigne?
2. Donner une suite de n opérations INSÉRER qui produit un arbre *splay* de profondeur n .
3. En supposant que l’opération *splay* est déjà programmée, décrire les manières les plus simples possibles d’implanter les opérations suivantes :

- SÉPARER(T, x) renvoie les deux arbres qui contiennent les clés plus petites et plus grandes que x , respectivement.
- FUSIONNER(T_1, T_2) combine les deux arbres T_1 et T_2 , et renvoie l'arbre résultant. Cette opération suppose que les clefs présentes dans T_1 sont toutes plus petites que celles présentes dans T_2 .

Exercice 2. COMPLEXITÉ AMORTIE

Nous voulons utiliser une nouvelle structure de donnée : un tableau de taille variable où il est possible

- d'ajouter un élément à la fin en $O(1)$
- de supprimer le dernier élément en $O(1)$
- et d'avoir accès à n'importe quel élément par indice en $O(1)$

De plus, nous voulons que la mémoire utilisée à tout moment est linéaire dans le nombre d'éléments dans le tableau.

Pour ce faire, nous utiliserons des tableaux de taille fixe, avec des recopies au bon moment. On applique les opérations de reconstruction suivantes.

- Si plus de $3/4$ de la tableau est remplie après une insertion, on crée un nouveau tableau vide de taille le double du tableau actuelle et on y insère tout les éléments du tableau actuel (et ensuite, on détruit l'ancien tableau).
- Si moins de $1/4$ du tableau est rempli après une suppression, on crée un nouveau tableau vide de taille la moitié de la table actuelle et on y insère tous les éléments du tableau actuelle (et ensuite, on détruit l'ancien tableau).

Montrer que pour n'importe quelle suite d'insertion et d'effaçage, on arrive à un temps de calcul $O(1)$ par opération.

Exercice 3. STRUCTURE DE DONNÉE AUGMENTÉE

Dans un arbre binaire de recherche T de taille n et hauteur $h(T)$, on veut garder plus d'information à chaque noeud pour pouvoir répondre à certaines requêtes plus rapidement. Ici, on garde en plus à chaque noeud, la taille du sous-arbre à ce noeud (noté $|T(v)|$ pour un noeud v).

1. Montrer qu'on peut calculer $|T(v)|$ pour tout noeud v en temps $O(n)$.
2. Trouver l'élément de rang i (le i ème plus petit élément) dans T en temps $O(h(T))$ à partir d'un tableau $|T(v)|$ pour tout v .
3. Trouver le rang d'un élément dans T en temps $O(h(T))$ à partir d'un tableau $|T(v)|$ pour tout v .
4. Montrer qu'en doublant le temps requis par opération, on peut effectuer toutes les opérations d'un arbre splay en mettant à jours la tailles des arbre $|T(v)|$.

Exercice 4. ANALYSE DES ARBRES SPLAY

Soit $\lfloor \log(|T(v)|) \rfloor$ le rang RANG(v) d'un sommet v dans un arbre. Soit RANG'(v) le rang de v après une rotation ou double rotation.

(Ici on suppose que tous les log sont en base 2, mais il est possible de continuer en 4.2 si on peut compléter 4.1 en n'importe quel base supérieur à 1.)

1. Montrer que la somme des rangs d'un arbre augmente d'au plus
 - (a) RANG'(x) - RANG(x) lors d'une rotation remontant x ,
 - (b) $2(\text{RANG}'(x) - \text{RANG}(x)) - 1$ lors d'un zig-zag remontant x ,
 - (c) $3(\text{RANG}'(x) - \text{RANG}(x)) - 1$ lors d'un zig-zig remontant x .
2. Montrer que pour n'importe quelle suite d'insertion, d'effaçage et de recherche à partir d'un arbre splay vide, on arrive à un temps de calcul $O(\log(n))$ par opération.