Algorithmique et Programmation

TD n° 4 : Arbres

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

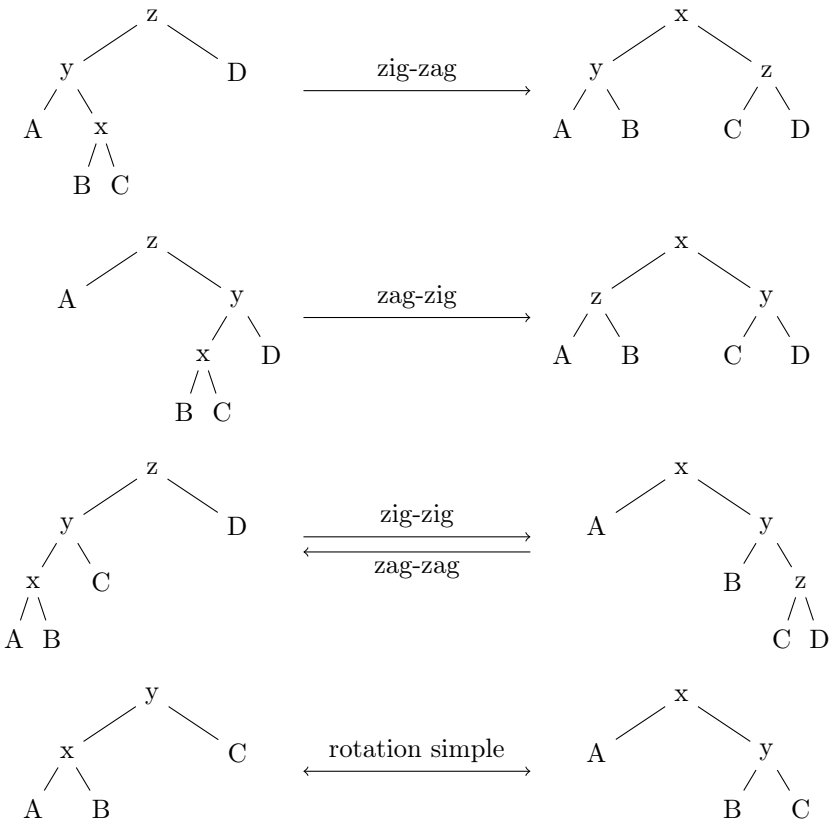
2012-2013

Exercice 1. 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, y) 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.

Les Splay Trees ont été inventés en 1985 par Sleator et Tarjan. Un splay tree est un arbre binaire de recherche qui supporte une nouvelle opération, splay (littéralement, "évaser", "écarter"). Dans un arbre binaire de recherche, une opération 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 :



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. 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 splay tree 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.

L'idée est qu'un arbre binaire de recherche qui subit fréquemment l'opération Splay reste à peu près équilibré, *alors qu'aucune donnée supplémentaire n'est stockée*, contrairement aux arbres rouges-noirs. En plus, les nœuds auxquels on accède fréquemment sont près de la racine.

Pour estimer la complexité des opérations sur les splay trees, on observe que la descente dans l'arbre est toujours moins coûteuse que l'opération splay, et donc il est suffisant d'avoir une bonne borne sur le coût de cette opération.

On va faire une analyse amortie en utilisant la méthode du potentiel. Dans une telle analyse, on définit une fonction potentiel Φ pour la structure de donnée qui initialement vaut $\Phi_0 = 0$ et reste toujours positive. Le coût amorti d'une opération est son coût réel plus la différence de potentiel : $a_i = c_i + \Phi_i - \Phi_{i-1}$. Il est facile de voir que le coût total de toute suite de m opérations est inférieur à son coût total amorti :

$$\sum_{i=1}^{m} a_i = \sum_{i=1}^{m} (c_i + \Phi_i - \Phi_{i-1}) = \Phi_m + \sum_{i=1}^{m} c_i \geq \sum_{i=1}^{m} c_i$$

Tout le problème est de définir une bonne fonction de potentiel. On définit le rang d'un nœud v par $\text{RANG}(v) = \log_2 \text{TAILLE}(v)$, où $\text{TAILLE}(v)$ désigne le nombre de sommets du sous-arbre enraciné en v (y compris v). De la sorte, le rang est toujours positif. On définit aussi le potentiel de l'arbre par $\Phi = \sum_v \text{RANG}(v)$, et par définition, le potentiel est toujours positif. On notera $\text{RANG}(v)$ et $\text{RANG}'(v)$ les rang avant et après une rotation (simple ou double), respectivement.

4. Donner l'ordre de grandeur asymptotique du potentiel pour un arbre parfaitement équilibré et pour un peigne.
5. Montrer que le coût amorti d'une rotation simple qui fait monter un noeud x (comme sur le dessin) est au plus :
$$1 + \text{RANG}'(x) - \text{RANG}(x).$$
6. Montrer que le coût amorti d'un Zig-Zag qui fait monter le noeud x (comme sur le dessin) est au plus :
$$2 \cdot \text{RANG}'(x) - 2 \cdot \text{RANG}(x).$$
7. Montrer que le coût amorti d'un Zig-Zig qui fait monter le noeud x (comme sur le dessin) est au plus :
$$3 \cdot \text{RANG}'(x) - 3 \cdot \text{RANG}(x).$$
8. En déduire le coût amorti d'une opération splay.
9. En déduire le coût amorti total d'une suite de n opérations RECHERCHER, INSÉRER, SUPPRIMER à partir d'un arbre vide.

Exercice 2.

1. Soit P une structure de tas qui effectue les opérations d'insertion, de suppression du minimum et de fusion en $O(\log n)$ opérations et qui crée un tas en $O(n)$ opérations (où n est la taille de la file de priorité).
   Montrer que P peut être modifié pour effectuer l'insertion en complexité amortie $O(1)$ opérations tout en conservant une suppression de minimum et de fusion en complexité amortie $O(\log n)$.
2. En utilisant la technique précédente, montrer que même un tas binaire peut être modifié pour effectuer l'insertion en complexité amortie $O(1)$ tout en conservant une suppression de minimum en complexité amortie $O(\log n)$.