Algorithmique et Programmation

TD n° 3 : Arbres

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

2013-2014

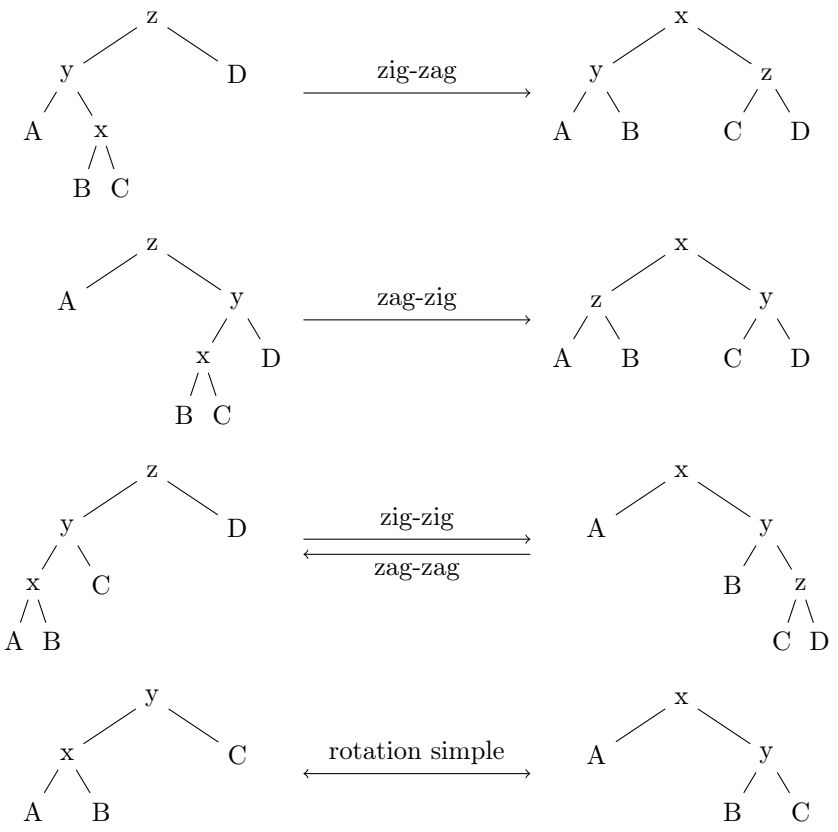
Exercice 1. Arbres splay

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, 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.

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. Un *peigne* est une 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

Soit une table de hachage de taille fixe avec insertion et effaçage en $O(1)$. Nous voulons utiliser cette table avec un nombre variable d'éléments. Pour ce faire, on applique les opérations de reconstruction suivantes.

– Si plus de $3/4$ de la table est remplie après une insertion, on crée une table vide de taille le double de la table actuelle et on y insert tout les éléments de la table actuelle (et ensuite, on détruit l'ancienne table).
– Si moins de $1/4$ de la table est remplie après une effaçage, on crée une table vide de taille la moitié de la table actuelle et on y insert tous les éléments du table actuelle (et ensuite, on détruit l'ancienne table).

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 veux garder plus d'information à chaque noeud pour pouvoir répondre à certaines requêtes plus rapidement. Ici, on garde en plus à chaque noeud, le 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 opérationn, on peut effectuer toutes les opérations d'un arbre splay en mettre 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.

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) $2($Rang$'(x) -$ Rang$(x)) - 1$ lors d'un zig-zig remontant x,
   (c) $3($Rang$'(x) -$ Rang$(x)) - 1$ lors d'un zig-zag 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.

Exercice 5. Structure de donnée augmentée

Nous cherchons une structure de donnée qui nous permet les opérations suivantes. Cette structure garde en mémoire un ensemble d'intervalles qui est au départ vide.

– Insérer d'une intervalle $[g, d]$.
– Effacer une intervalle $[g, d]$.
– Trouver une intervalle contenant un entier p.

Cette dernière opération retourne une érreur si p est dans aucune intervalle. N'importe quel intervalle peut être retournée si p est contenu dans plusieur intervalles.

Nous cherchons une structure de donnée qui permet ces trois opérations en $O(\log(n))$ où n est le nombre d'intervalles présents au moment de l'opération.

(Supposer que les opération de comparaison et arithmétique sur entiers délimitant les intervalles se font en temps constant. Ou en d'autre terme, on permet un facteur supplémentaire de log du plus grand entier reçu à l'entrée.)

1. Trouver un arbre splay augmenté (sur les intervalles) pour ce problème et montrer que nous pouvons trouver une intervalle contenant p en temps $O(\log n)$.
   Indice : utiliser le point gauche des intervalles comme clés de l'arbre.
2. Montrer que les opérations de l'arbre splay peuvent toujours être complétés dans le même ordre de temps si on met à jours les champs supplémentaires.