I202B, Les arbres + la récursivité: exercices

J. Vander Meulen C. Damas

Mars 2017

1 La récursivité

Nous vous demandons de construire à la fois les méthodes récursives suivantes de la classe `Trees` et les tests de ces méthodes :

1. `static int nbrNodes(Tree t)` qui renvoie le nombre de noeuds de l'arbre `t`.
2. `static int min(Tree t)` qui renvoie la plus petite valeur associée à un noeud de l'arbre `t`.
3. `static int sum(Tree t)` qui renvoie la somme des valeurs associées aux noeuds de l'arbre `t`.
4. `static boolean equals(Tree t1, Tree t2)` qui renvoie `true` si `t1` et `t2` représente le même arbre et qui renvoie `false` sinon.
5. `static int depth(Tree n)` qui renvoie la profondeur du noeud `n`, c'est-à-dire la longueur du chemin entre la racine et le noeud `n`, ou autremment dit le nombre d'arcs qu'il faut traverser pour aller de la racine au noeud `n`.
6. `static boolean sameOne(Tree n1, Tree n2)` qui renvoie `true` si `n1` et `n2` appartiennent tous les deux à un même arbre et qui renvoie `false` sinon.
7. `static void dfsPrint(Tree t)` qui simule un parcours de l'arbre de type « depth-first search ». En pratique, cette méthode affiche la valeur des noeuds visités, en respectant l'ordre du parcours DFS, sur la sortie standard.
8. `static void bfsPrint(Tree t)` qui simule un parcours de l'arbre de type « breadth-first search ». En pratique, cette méthode affiche la valeur des noeuds visités, en respectant l'ordre du parcours BFS, sur la sortie standard. Pensez à écrire un sous-problème plus général.

2 Les chemins

Nous vous demandons de construire à la fois les méthodes suivantes de la classe `Trees` et les tests de ces méthodes :

1. `static void printPathV1(Tree n)` : Cette méthode récursive prend en paramètre un noeud `n` d'un arbre et affiche le chemin qui va de la racine à `n` sur la sortie standard.
2. `static void printPathV2(Tree node)` : Cette méthode fait la même chose que la méthode `printPathV1` mais n'est pas récursive.
3. `static void printPathV3(Tree t, int v)` : Cette méthode prend en paramètre un arbre `t` et une valeur `v`. Vous pouvez supposer qu'il existe au moins un noeud de `t` dont la valeur vaut `v`. Elle affiche le chemin qui va de la racine vers un noeud dont la valeur vaut `v` sur la sortie standard.

3 Les réprésentations

Nous vous demandons de construire à la fois les méthodes suivantes de la classe `Trees` et les tests de ces méthodes :

1. `static int[][] toArray(Tree t)` : Cette méthode récursive prend en paramètre un arbre binaire `t` qui est représenté à l'aide de la classe `Tree` et elle renvoie l'arbre `t` représenté cette fois-ci sous forme d'un tableau à deux dimension (cf. transparents 23 relatifs à la partie théorique).
2. `static Tree toTree(int[][] t)` : Cette méthode récursive prend en paramètre un arbre binaire `t` qui est représenté à l'aide d'un tableau à deux dimensions et elle renvoie l'arbre `t` représenté cette fois-ci à l'aide de la classe `Tree`.

4 Le plus petit ancêtre commun

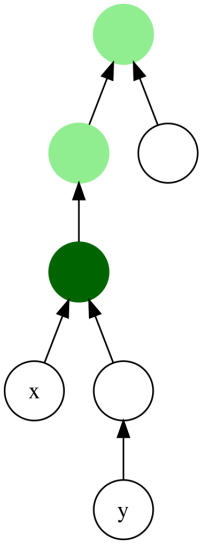

FIGURE 1 – Dans cet arbre, le plus petit ancêtre commun de x et y est le sommet vert foncé. Les autres ancêtres sont les sommets verts clairs.

Étant donné un arbre, les ancêtres communs de deux nœuds, sont les sommets qui sont ancêtres de ces deux nœuds ; autrement dit ce sont les nœuds qui ont ces deux nœuds dans leurs descendances. Le plus petit ancêtre commun (LAC) à ces deux nœuds est l'ancêtre commun le plus profond, c'est-à-dire le plus éloigné de la racine (voir Figure 1) [1].

Nous vous demandons de construire la méthode, et les tests de la méthode, « `static Tree lca(Tree n1, Tree n2)` » qui prend en paramètre deux noeuds `n1` et `n2` et qui renvoie le plus petit ancêtre commun à `n1` et `n2`. Vous pouvez supposez que `n1` et `n2` appartiennent au même arbre. La complexité de votre méthode doit être équivalente à $\mathcal{O}(n)$ où n est le nombre de noeuds de l'arbre.

1. `https://fr.wikipedia.org/wiki/Plus_petit_ancêtre_commun`