Algorithmique avancée - TD6

—o000o——o000o—

Introduction aux arbres binaires

1 Génération d'un arbre aléatoire

On s'intéresse tout d'abord à la création aléatoire d'un arbre binaire de type `ABInt` contenant des valeurs entières qui étend `AB<Integer>`.

1. Écrire une méthode `addRand(int val)` qui permet d'ajouter la valeur `val` passée en argument en suivant aléatoirement les branches de l'arbre binaire. Par construction du modèle objet, l'arbre binaire ne peut être vide. En partant du nœud initial, un tirage alétoire doit être réalisé pour savoir si la valeur doit être ajoutée au sous-arbre gauche ou au sous-arbre droit. L'algorithme doit vérifier si les sous-arbres existent avant de les modifier en y ajoutant récursivement la valeur.

2. À partir de la méthode précédente, proposer une méthode `addAlea(int size, int valMax)` qui ajoute aléatoirement `size` valeurs générées aléatoirement entre 0 et $valMax - 1$ à l'arbre binaire.

3. Écrire enfin une méthode `addOrder(int val)` qui cette fois n'ajoute pas la valeur en choisissant aléatoirement entre l'arbre gauche et l'arbre droit, mais en suivant les règles suivantes :

   - si la valeur est égale à l'étiquette de l'arbre, on ne l'insère pas et on laisse l'arbre tel qu'il est
   - si la valeur est strictement inférieure à l'étiquette de l'arbre, on l'insère dans le sous-arbre gauche
   - sinon, on l'insère dans le sous-arbre droit

2 Plus court chemin dans un arbre binaire

On souhaite compléter la classe précédente en calculant le chemin le plus court entre la racine de l'arbre et une feuille.

1. Écrire la méthode `shortestPath()` qui retourne la longueur du chemin le plus court entre la racine de l'arbre et une feuille.

2. Écrire la méthode `shortestPathNodes()` qui retourne la liste chaînée des valeurs contenues dans le chemin le plus court entre la racine de l'arbre et une feuille. La liste chaînées prendra la forme d'un nœud de liste de type `Node<Integer>`. Si le chemin gauche et le chemin droit font la même longueur, on préférera le chemin gauche.

Solution :

```java
package trees;

import list.ISList;

import java.util.Iterator;
import java.util.Random;

public class ABInt extends AB<Integer> {

    public ABInt(int value){
        super(value);
    }

    public ABInt(int value, ABInt left, ABInt right){
        super(value, left, right);
    }

    @Override
    public ABInt getLeft(){
        return (ABInt) left;
    }

    public void setLeft(ABInt tree){
        left = tree;
    }

    @Override
    public ABInt getRight(){
        return (ABInt) right;
    }

    public void setRight(ABInt tree){
        right = tree;
    }

    // Question 1 - add a value randomly to a tree
    public void addRand (int val) {
        if (Math.random() < 0.5) {
            if (hasLeft()) {
                getLeft().addRand(val);
            }else {
```

```
42                  setLeft(new ABInt(val));
43              }
44          } else {
45              if (hasRight()) {
46                  getRight().addRand(val);
47              }else {
48                  setRight(new ABInt(val));
49              }
50          }
51      }
52
53      // Question 2 - random generation of a complete binary
           tree
54      public void addAlea(int size, int valMax) {
55          Random r = new Random();
56          for (int i = 0; i < size; i++) {
57              this.addRand(r.nextInt(valMax));
58          }
59      }
60
61      // Question 3 - add a value following the order of values
            inserted in a tree
62      public void addOrder (int val) {
63          if (val < this.getLabel()) {
64              if (hasLeft()) {
65                  getLeft().addOrder(val);
66              }else {
67                  setLeft(new ABInt(val));
68              }
69          } else {
70              if (hasRight()) {
71                  getRight().addOrder(val);
72              }else {
73                  setRight(new ABInt(val));
74              }
75          }
76      }
77
78      // Exercice 2
79      public int shortestPath(){
80          int pathLeft = (hasLeft())? getLeft().shortestPath()
               :0;
81          int pathRight = (hasRight())? getRight().shortestPath
               ():0;
```

```
82          if (pathLeft > 0 && pathRight > 0) return Math.min(
               pathLeft,pathRight) + label;
83          else if (pathLeft > 0) return label + pathLeft;
84          else return label + pathRight;
85      }
86
87      public ISList<Integer> shortestPathNode(){
88          ISList<Integer> res = new ISList<>(label);
89          ISList<Integer> pathLeft = (hasLeft())? getLeft().
               shortestPathNode():null;
90          ISList<Integer> pathRight = (hasRight())? getRight().
               shortestPathNode():null;
91          if (pathLeft != null && pathRight != null) {
92              if (pathLeft.size() < pathRight.size()){
93                  Iterator<Integer> it = pathLeft.iterator();
94                  while (it.hasNext()) res.add(it.next());
95                  return res;
96              } else {
97                  Iterator<Integer> it = pathRight.iterator();
98                  while (it.hasNext()) res.add(it.next());
99                  return res;
100             }
101         } else if (pathLeft != null) {
102             Iterator<Integer> it = pathLeft.iterator();
103             while (it.hasNext()) res.add(it.next());
104             return res;
105         }
106         else if (pathRight != null){
107             Iterator<Integer> it = pathRight.iterator();
108             while (it.hasNext()) res.add(it.next());
109             return res;
110         } else return res; // pathLeft == null && pathRight
               == null
111     }
112
113     public static void main(String[] args) {
114         ABInt tree = new ABInt(42);
115         tree.addAlea(4, 100);
116         System.out.println(tree);
117
118         System.out.println("Somme des poids " + tree.
               shortestPath());
119         System.out.println(tree.shortestPathNode());
120
```

```
121         /*
122         ABInt tree2 = new ABInt(50);
123         Random r = new Random();
124         int valMax = 100;
125         int size = 9;
126         for (int i = 0; i < size; i++) {
127             tree2.addOrder(r.nextInt(valMax));
128         }
129         System.out.println(tree2);
130          */
131     }
132
133 }
```

java/ABInt.java