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

3 Liste infixe, préfixe et suffixe

Le parcours d'un arbre binaire peut donner lieu à 3 listes de valeurs différentes selon l'ordre dans lequel on considère la racine, l'arbre gauche et l'arbre droit.

1. À partir de votre cours, et sans regarder les implémentations Java proposées, écrire les méthodes `infixe()`, `prefixe(A)` et `postfixe(A)` qui s'appliquent sur tout arbre binaire et retournent une liste chaînée représentée par une structure de type `Node<T>`.

Solution :

```
1   /** new version with iterable list */
2   public ISList<T> prefix(){
3     if (isLeaf()) return new ISList<>(getLabel(), null);
4     else {
5       // Computing nodes for left and right sub trees
6       ISList<T> left = (hasLeft())? getLeft().prefix() : new
          ISList<>();
7       ISList<T> right = (hasRight())? getRight().prefix() :
          new ISList<>();
8       // merging left and right nodes
9       Iterator<T> it = right.iterator();
10      while (it.hasNext()){
11        left.add(it.next());
12      }
13      return new ISList<T>(this.getLabel(), left);
14    }
```

```java
  }

  public ISList<T> infix(){
    if (isLeaf()) return new ISList<T>(getLabel(), null);
    else {
      ISList<T> left = (hasLeft())? getLeft().infix() : new
         ISList<>();
      ISList<T> right = (hasRight())? getRight().infix() :
         new ISList<>();

      left.add(getLabel());

      Iterator<T> it = right.iterator();
      while (it.hasNext()){
        left.add(it.next());
      }

      return left;
    }
  }

  public ISList<T> postfix(){
    if (isLeaf()) return new ISList<T>(getLabel(), null);
    else {
      // Computing nodes for left and right sub trees
      ISList<T> left = (hasLeft())? getLeft().postfix(): new
         ISList<>();
      ISList<T> right = (hasRight())? getRight().postfix():
         new ISList<>();

      // merging left and right nodes

      Iterator<T> it = right.iterator();
      while (it.hasNext()){
        left.add(it.next());
      }

      left.add(getLabel());
      return left;
    }
  }
```

java/parcours.java

4 Dénombrement à un certain niveau

1. On s'intéresse tout d'abord à l'écriture d'une méthode pour déterminer le nombre de nœuds à un niveau k (entier positif) dans un arbre binaire B. On considérera que la racine d'un arbre binaire non vide est au niveau 1 et que de manière général, le niveau d'un nœud est égal à celui de son père augmenté de 1.

2. Écrire maintenant une méthode qui permet de déterminer le nombre de feuilles à un niveau donné k (entier positif) en suivant les mêmes contraintes que précédemment.

Solution :

```
1  package trees;
2
3  import list.ISList;
4
5  import java.util.Iterator;
6  import java.util.Random;
7
8  public class Denombrement extends ABInt{
9
10     public Denombrement(Integer _label) {
11         super(_label);
12     }
13
14     public Denombrement(Integer _label, Denombrement _left,
          Denombrement _right){
15         super(_label, _left, _right);
16     }
17     @Override
18     public Denombrement getLeft(){
19         return (Denombrement) left;
20     }
21     @Override
22     public Denombrement getRight(){
23         return (Denombrement) right;
24     }
25     @Override
26     public void addRand (int val) {
27         if (Math.random() < 0.5) {
28             if (hasLeft()) {
29                 getLeft().addRand(val);
30             }else {
31                 setLeft(new Denombrement(val));
```

```
32              }
33          } else {
34              if (hasRight()) {
35                  getRight().addRand(val);
36              }else {
37                  setRight(new Denombrement(val));
38              }
39          }
40      }
41      @Override
42      public void addAlea(int size, int valMax) {
43          Random r = new Random();
44          for (int i = 0; i < size; i++) {
45              this.addRand(r.nextInt(valMax));
46          }
47      }
48
49
50      public int nbNodesLevel(int k){
51          if (k == 1) return 1;
52          else {
53              int nbLeft = (hasLeft())? getLeft().nbNodesLevel(
                    k-1):0;
54              int nbRight = (hasRight())? getRight().
                    nbNodesLevel(k-1):0;
55              return nbLeft + nbRight;
56          }
57      }
58
59      public ISList<Integer> listNodesLevel(int k){
60          if (k == 1) return new ISList<>(label);
61          else {
62              ISList<Integer> Left = (hasLeft())? getLeft().
                    listNodesLevel(k-1):new ISList<>();
63              ISList<Integer> Right = (hasRight())? getRight().
                    listNodesLevel(k-1):new ISList<>();
64              Iterator<Integer> it = Right.iterator();
65              while(it.hasNext()) Left.add(it.next());
66              return Left;
67          }
68      }
69
70      public static void main(String[] args) {
71          Denombrement tree = new Denombrement(42);
```

```
72          tree.addAlea(19,100);
73          System.out.println(tree);
74          System.out.println("Nb. nodes at depth 3: " + tree.
               nbNodesLevel(3));
75          System.out.println("List of nodes at depth 3: " +
               tree.listNodesLevel(3));
76      }
77  }
```

java/Denombrement.java

5 Égalité entre deux arbres binaires

On s'intéresse maintenant à la définition d'une méthode pour déterminer si deux arbres binaires sont égaux. On considère ici que 2 arbres binaires sont égaux s'ils possèdent les mêmes valeurs d'étiquettes et la même structure : si l'un possède (resp. ne possède pas) un fil (gauche ou droit) l'autre doit (resp. ne doit pas) en avoir un. On pose que deux feuilles sont égales si elles possèdent la même étiquette.

Solution :

```
1  package trees;
2
3  import java.util.Objects;
4
5  public class Egalite<T> extends AB<T> {
6      public Egalite(T value){
7          super(value);
8      }
9
10     public Egalite(T value, Egalite _left, Egalite _right){
11         super(value, _left, _right);
12     }
13
14     @Override
15     public Egalite getLeft(){return (Egalite) left;}
16
17     @Override
18     public Egalite getRight(){return (Egalite) right;}
19
20     @Override
21     public boolean equals(Object obj) {
22         if (obj instanceof Egalite){
```

```
23              return equal((Egalite) obj);
24          } else return false;
25      }
26
27      public boolean equal(Egalite<T> ab){
28          if (ab == null) return false;
29          else if (isLeaf() && ab.isLeaf()) return Objects.
               equals(label, ab.label);
30          else {
31              boolean e =  Objects.equals(label, ab.label);
32              boolean l = (hasLeft() && getLeft().equal(ab.
                   getLeft())) || (!hasLeft() && !ab.hasLeft());
33              boolean r = (hasRight() && getRight().equal(ab.
                   getRight())) || (!hasRight() && !ab.hasRight()
                   );
34              return e && l && r;
35          }
36      }
37
38      public static void main(String[] args) {
39          Egalite<Integer> a1 = new Egalite<>(1, null, new
               Egalite(2));
40          Egalite<Integer> a2 = new Egalite<>(1, null, new
               Egalite(2));
41          Egalite<Integer> a3 = new Egalite<>(2);
42
43          System.out.println("a1 == a2? " + a1.equals(a2));
44          System.out.println("a1 == a3? " + a1.equals(a3));
45          System.out.println("a2 == a3? " + a2.equals(a3));
46      }
47  }
```

java/Egalite.java

6 Miroir d'un arbre binaire

L'arbre miroir d'un arbre binaire B est défini récursivement de la manière suivante :

- il possède la même racine que l'arbre binaire B,
- son fil gauche est l'arbre miroir du fil droit de B,
- son fil droit est l'arbre miroir du fil gauche de B.

Le miroir d'une feuille est elle-même et on ne gère pas le cas des arbres vides.

1. Écrire une méthode qui prend en argument un arbre binaire et retourne son arbre miroir. Vous proposerez différents tests pour vérifier votre méthode.
2. Écrire une méthode prenant en argument 2 arbres binaires quelconques A et B et retourne `true` si et seulement si B est l'arbre miroir de A et `false` sinon.
3. Que vaut l'arbre miroir de l'arbre miroir d'un arbre binaire ?

Solution :

```
1  package trees;
2
3  import java.util.Objects;
4
5  public class Miroir<T> extends Egalite<T> {
6      public Miroir(T value){
7          super(value);
8      }
9
10     public Miroir(T value, Miroir _left, Miroir _right){
11         super(value, _left, _right);
12     }
13
14     @Override
15     public Miroir getLeft(){return (Miroir) left;}
16
17     @Override
18     public Miroir getRight(){return (Miroir) right;}
19
20     public Miroir miroir(){
21         if (hasLeft()) {
22             if (hasRight()) return new Miroir(label, getRight
                   ().miroir(), getLeft().miroir());
23             else return new Miroir(label, null, getLeft().
                   miroir());
24         } else { // left == null
25             if (hasRight()) return new Miroir(label, getRight
                   ().miroir(), null);
26             else return new Miroir(label, null, null);
27         }
28     }
29
30     public void addRand (int val) {
31         if (Math.random() < 0.5) {
```

```
32              if (hasLeft()) {
33                  getLeft().addRand(val);
34              }else {
35                  setLeft(new Miroir(val));
36              }
37          } else {
38              if (hasRight()) {
39                  getRight().addRand(val);
40              }else {
41                  setRight(new Miroir(val));
42              }
43          }
44      }
45
46      public static void main(String[] args) {
47          int valMax = 100;
48          int size = 4;
49          Miroir<Integer> tree = new Miroir<Integer>((int) (
                Math.random()* valMax));
50          for (int i = 0; i < size; i++){
51              tree.addRand((int) (Math.random()* valMax));
52          }
53
54          System.out.println(tree);
55          System.out.println(tree.miroir());
56          System.out.println(tree.miroir().miroir());
57      }
58  }
```

java/Miroir.java