PRG – TP9 à TP12 – Arbres Binaires

Il est fortement conseillé de lire la totalité du sujet avant de se lancer dans la réalisation.

L'objectif de ce TP est d'utiliser les arbres binaires pour le stockage d'images binaires (ou en noir et blanc). Il vise à fournir des méthodes efficaces pour la transformation des images binaires à travers la manipulation de la structure des arbres binaires.

1 Introduction

On considère des images binaires de taille 256×256 pixels où chaque pixel peut avoir soit la valeur 0 (couleur noire) soit la valeur 1 (couleur blanche). On suppose que l'image est divisée en **régions** qui sont classées en trois categories :

— 0 si tous les pixels de la région sont noirs (ou éteints),

— 1 si tous les pixels de la région sont blancs (ou allumés),

— 2 sinon.

Chaque région est définie par le 5-tuple suivant : $\{c, x_{hg}, y_{hg}, x_{bd}, y_{bd}\}$ avec ;

c : la catégorie de la région,

(x_{hg}, y_{hg}) : les coordonnées du coin haut-gauche de la région,

(x_{bd}, y_{bd}) : les coordonnées du coin bas-droit de la région.

On souhaite associer l'image à un arbre binaire où chaque noeud représente une région particulière. Chaque noeud de l'arbre binaire peut avoir la valeur 0, 1, ou 2. Les noeuds de l'arbre sont de type `Node` qui conserve des objets de type entier. **N.B.** Les coordonnées d'une région ne sont pas explicitement représentées dans le noeud correspondant (mais implicitement par la "place" de ce noeud dans l'arbre).

Pour la création des régions, on procède par des découpages successifs, horizontalement puis verticalement, de l'image en deux "moitiés", jusqu'à obtention de régions (carrées ou rectangulaires) de catégorie 0 ou 1 (donc totalement éteintes ou totalement allumées).

L'arbre binaire qui découle de ce processus a les propriétés suivantes :

- Les feuilles représentent des régions de l'image totalement éteintes ou bien totalement allumées, et ont resp. soit la valeur 0, soit la valeur 1.
- Chaque autre nœud a deux fils (nœud double) et la valeur 2 ; ces fils correspondent au découpage en deux "moitiés" de la région représentée par le noeud double.
- Les nœuds de niveau *pair* correspondent à des régions carrées obtenues après un découpage vertical de la région représentée par leur noeud père (sauf pour la racine, de niveau 0, qui représente l'image complète).
- Les nœuds de niveau *impair* correspondent à des régions rectangulaires après un découpage horizontal de la région représentée par leur noeud père.

Exemple de représentation d'une image par un tel arbre binaire
On considère l'image suivante :

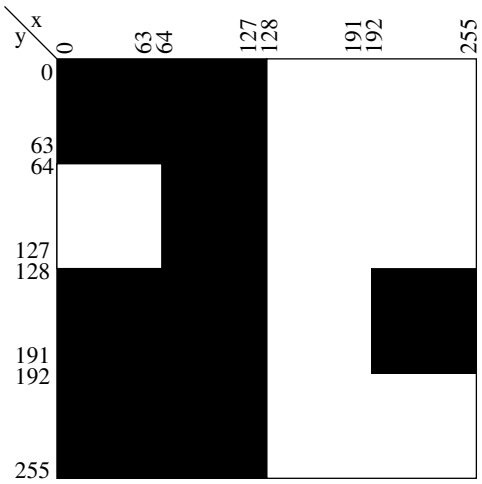


FIGURE 1 – Image binaire.

L'arbre binaire associé à cette image est le suivant :

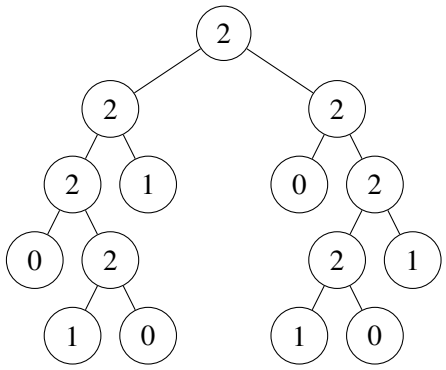


FIGURE 2 – Arbre binaire associé à l'image 1

L'application du processus décrit auparavant sur l'image 1 conduit à la représentation par les feuilles de valeur 1 des régions blanches suivantes : $\{1, 0, 64, 63, 127\}$, $\{1, 128, 0, 255, 127\}$, $\{1, 128, 128, 191, 191\}$ et $\{1, 128, 192, 255, 255\}$, par ex.

2 Manipulation d'images via des arbres

On souhaite appliquer diverses manipulations sur les images binaires décrites précédemment à travers des opérations faites sur les arbres binaires associés.

Pour cela, la classe `AbstractImage` hérite de la classe `BinaryTree` et définit les méthodes abstraites des manipulations d'image souhaitées (par ex. affectation d'une image à une autre, rotation à 180° d'une image, intersection de deux images, etc.).

`AbstractImage` fournit aussi certaines méthodes concrètes telles que `constructTreeFromFile`, `saveImage`, `height`, `numberOfNodes`, `plotTree` qui sont nécessaires/utiles pour l'implémentation des méthodes abstraites.

Notez bien que la classe `AbstractImage` vous est fournie avec l'implémentation des méthodes concrètes `constructTreeFromFile`, `saveImage`, etc.

Ci-après, le squelette de la class `AbstractImage` vous est donné :

```
1
2  public abstract class AbstractImage extends BinaryTree<Node>
3
4  /**
5   * Crée this à partir d'un fichier texte (cf a1.arb, ...) et l'affiche
6   * dans une fenêtre. Chaque ligne du fichier est de la forme
7   * (e x1 y1 x2 y2) et indique si on souhaite  éteindre (e=0) ou
8   * allumer (e=1) la région rectangulaire de coordonnées x1, y1, x2, y2.
9   * Le fichier se termine par un e de valeur -1.
10  */
11 public void constructTreeFromFile() { ... }
12
13
14 /**
15  * Sauvegarde, dans un fichier texte, les feuilles de this selon un
16  * format conforme aux fichiers manipulés par la commande
17  * constructTreeFromFile.
18  *
19  * @pre           !this.isEmpty()
20  */
21 public void saveImage() { ... }
22
23
24 /**
25  * @pre           !this.isEmpty()
26  * @return        hauteur de this
27  */
28 public int height() { ... }
29
```

```
30
31 /**
32  * @pre           !this.isEmpty()
33  * @return        nombre de nœuds de this
34  */
35 public int numberOfNodes() { ... }
36
37
38 /**
39  * @param x       abscisse du point
40  * @param y       ordonnée du point
41  * @pre           !this.isEmpty()
42  * @return        true, si le point (x, y) est allumé dans this,
43  *                false sinon
44  */
45 public abstract boolean isPixelOn(int x, int y);
46
47
48 /**
49  * this devient identique à image.
50  *
51  * @pre           !image.isEmpty()
52  * @pre           this != image
53  */
54 public abstract void affect(AbstractImage image);
55
56
57 /**
58  * this devient inverse vidéo de this, pixel par pixel.
59  *
60  * @pre           !this.isEmpty()
61  */
62 public abstract void videoInverse();
63
64
65 /**
66  * this devient rotation de image à 180 degrés
67  *
68  * @param image   image pour rotation
69  * @pre           !image.isEmpty()
70  * @pre           this != image
71  */
72 public abstract void rotate180(AbstractImage image);
73
74
```

```
75  /**
76   * this devient image miroir vertical de image.
77   *
78   * @param image  image à agrandir
79   * @pre          !image.isEmpty()
80   * @pre          this != image
81   */
82  public abstract void mirrorV(AbstractImage image);
83
84
85  /**
86   * this devient image miroir horizontal de image.
87   *
88   * @param image  image à agrandir
89   * @pre          !image.isEmpty()
90   * @pre          this != image
91   */
92  public abstract void mirrorH(AbstractImage image);
93
94
95  /**
96   * this devient quart supérieur gauche de image.
97   *
98   * @param image  image à agrandir
99   * @pre          !image.isEmpty()
100  * @pre          this != image
101  */
102 public abstract void zoomIn(AbstractImage image);
103
104
105 /**
106  * Le quart supérieur gauche de this devient image (réduite),
107  * le reste de this devient éteint.
108  *
109  * @param image  image à réduire
110  * @pre          !image.isEmpty()
111  * @pre          this != image
112  */
113 public abstract void zoomOut(AbstractImage image) ;
114
115
116 /**
117  * this devient l'intersection de image1 et image2 au sens des pixels
118  * allumés.
119  *
```

```
120  * @param image1 première image
121  * @param image1 deuxième image
122  * @pre          !image1.isEmpty() && !image2.isEmpty()
123  * @pre          this != image1
124  * @pre          this != image2
125  */
126 public abstract void intersection(AbstractImage image1,
127                                   AbstractImage image2);
128
129
130 /**
131  * this devient l'union image1 et image2 au sens des pixels allumés.
132  *
133  * @param image1 première image
134  * @param image1 deuxième image
135  * @pre          !image1.isEmpty() && !image2.isEmpty()
136  * @pre          this != image1
137  * @pre          this != image2
138  */
139 public abstract void union(AbstractImage image1,
140                            AbstractImage image2);
141
142
143 /**
144  * Cette fonction ne doit pas utiliser la fonction isPixelOn.
145  *
146  * @pre          !this.isEmpty()
147  * @return       true, si tous les points de la forme (x, x)
148  *               (avec 0 <= x <= 255) sont, ou non, allumés dans this,
149  *               false sinon
150  */
151 public abstract boolean testDiagonal();
152
153
154 /**
155  * @param x1    abscisse du premier point
156  * @param y1    ordonnée du premier point
157  * @param x2    abscisse du deuxième point
158  * @param y2    ordonnée du deuxième point
159  * @pre         !this.isEmpty()
160  * @return      true si les deux points (x1, y1) et (x2, y2) sont
161  *              représentés par la même feuille de this, false sinon
162  */
163 public abstract boolean sameLeaf(int x1, int y1, int x2, int y2);
164
```

```
165
166 /**
167  * @param image  seconde image
168  * @pre          !this.isEmpty() && !image.isEmpty()
169  * @pre          this != image
170  * @return       true si this est inclus dans image au sens
171  *               des pixels allumés, false sinon
172  */
173 public abstract boolean isIncludedIn(AbstractImage image);
174
175
176 /**
177  * Affiche this sous forme d'arbre dans une fenêtre externe.
178  *
179  * @pre         !this.isEmpty()
180  */
181 public void plotTree() { ... }
182
183 }
```

Travail à faire

Vous devez coder les méthodes abstraites souhaitées dans la classe `Image` qui hérite de la classe `AbstractImage` et implémente les méthodes de manipulation des images à partir des arbres binaires les représentant.

`public class` `Image` **`extends`** `AbstractImage`

Les méthodes d'instance suivantes sont obligatoirement à implémenter :
`affect`, `rotate180`, `videoInverse`, `mirrorV`, `mirrorH`, `intersection`, `inIncludedIn`, `testDiagonal`, `isPixelOn` et `zoomOut`.

Les méthodes suivantes peuvent être implémentées mais sont en bonus pour le rendu de ce TP : `zoomIn`, `union` et `sameLeaf`.

L'archive `tp-arbres.tar`, disponible sur `Moodle`, comporte le squelette de la classe `Image.java` à compléter, la classe `TpArbre.java` qui fournit un `menu` permettant d'appeler les différentes méthodes de manipulation des images, et un ensemble de fichiers décrivant des images.

L'archive `tp-arbres.jar` contient l'ensemble des fichiers `.class` nécessaires au bon fonctionnement de la classe `Image`, dont `AbstractImage.class`.

Dans un premier temps, vous devez utiliser la mise en œuvre des arbres binaires par les enseignants, `BinaryTree.class`, fournie dans `tp-arbres.jar`.

Le fichier `conseils-TpArbres-PRG.pdf` résume la démarche à suivre pour générer votre projet java et pour réaliser votre TP.

3 Test de l'implementation de la classe `Image`

Pour tester votre implémentation de la classe `Image`, vous disposez d'un jeu de tests `JUnit` dans le fichier `testImage.java`. Ce programme de test utilise des fichiers *.arb* décrivant des images (par stockage des coordonnées des régions blanches/allumées des images à traiter).

Pour le test en mode interactif, vous disposez d'une classe `TpArbre.java` qui permet de créer une fenêtre graphique comportant cinq images de taille 256×256. La classe `TpArbre` fournit un `menu` permettant d'appeler les différentes méthodes d'instance de la classe `Image`. La transformation appliquée aux images peut être affichée pour vérifier visuellement le résultat du traitement. Notez que les arbres représentant les images peuvent également être visualisés.

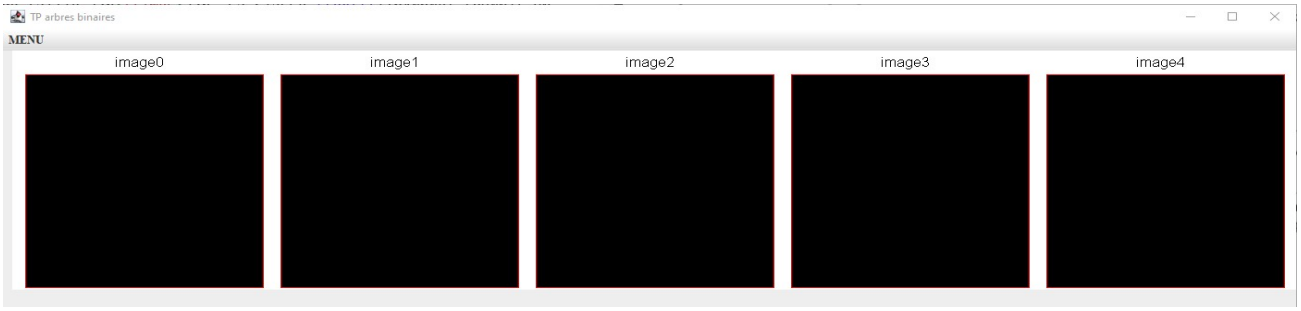


FIGURE 3 – Interface de test de la classe `Image`

4 Écriture d'une mise en œuvre

Dans un second temps, vous compléterez la classe `BinaryTree.java` qui se trouvera sur `Moodle`. Elle correspond à une mise en œuvre des arbres binaires par une représentation chaînée par références avec pile des pères. Tester le bon fonctionnement des méthodes implémentées de manipulation d'images avec votre mise en œuvre des arbres binaires.

5 Travail à rendre

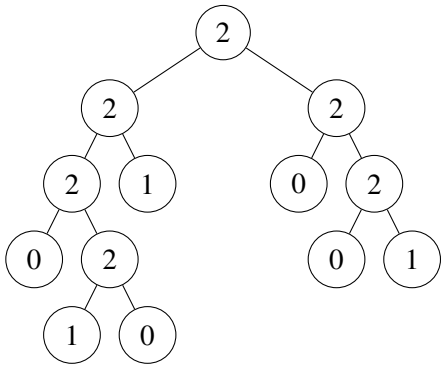
Les fichiers complétés `Image.java` et `BinaryTree.java` sont **à rendre sur Moodle au plus tard le mardi 7 novembre 2023 à 20h**.

Vous veillerez à ne pas modifier les noms de classes et les packages des squelettes de code fournis.
Le code source livré devra être entièrement compilable et l'en-tête de vos fichiers sera un commentaire (idéalement de type JavaDoc) indiquant les noms et prénoms des auteurs (les deux membres du binôme).

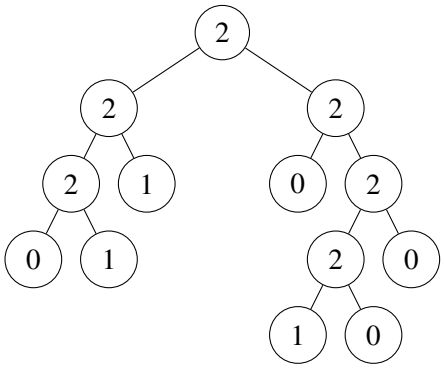
Attention, si vous programmez sur votre ordinateur personnel, vos classes doivent pouvoir être utilisées dans un environnement du type des salles de TP Istic.

Travail préliminaire à rendre à votre enseignant.e lors de la 1ere séance de TP

Soit les deux arbres suivants :



(a) A1 : Arbre binaire 1



(b) A2 : Arbre binaire 2

Donnez l'arbre binaire résultant de l'opération suivante A1 ∩ A2

Donnez l'implémentation (code Java) de la méthode `intersection` de la classe `Image`.