PRG1 – TP 6 – Arbres binaires

On désire conserver l'état d'une fenêtre graphique (256×256) sous la forme d'un arbre binaire d'objets *Node* entiers. Chaque pixel de la fenêtre est soit éteint (state=0), soit allumé (state=1). On associe à une région quelconque de la fenêtre un des trois états suivants :

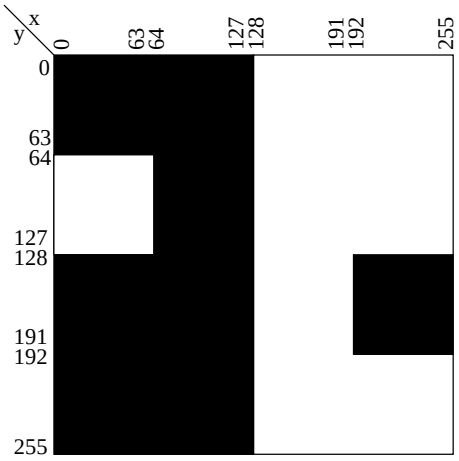
0 si tous les pixels de la région sont éteints,

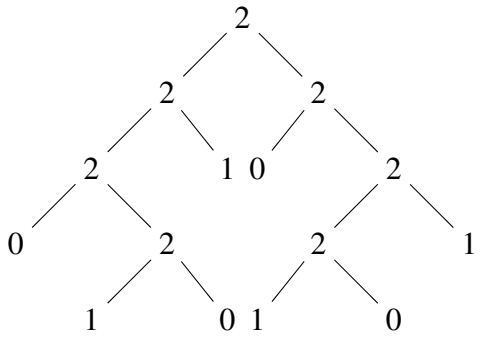
1 si tous les pixels de la région sont allumés,

2 sinon.

Pour représenter la fenêtre dans un arbre binaire, on procède par découpages successifs de la fenêtre en deux régions égales, d'abord horizontalement, puis verticalement, puis horizontalement,... jusqu'à obtention de régions (carrées ou rectangulaires) dont l'état est 0 ou 1. Dans l'arbre binaire associé, les nœuds de niveau pair correspondent à une région carrée et ceux de niveau impair à une région rectangulaire.

Exemple (les pixels éteints sont en noir, les pixels allumés sont en blanc) :





fournit l'arbre associé à un dessin composé par allumage des rectangles ayant pour coordonnées $[0, 64, 63, 127]$, $[128, 0, 255, 127]$, $[128, 128, 191, 191]$ et $[128, 192, 255, 255]$.

Manipulation d'images à l'aide des accès arbre

La classe `TpArbre` permet d'obtenir une configuration écran comportant cinq fenêtres graphiques de taille 256 × 256, correspondant à cinq images que le programme peut traiter.

La classe `TpArbre` gère le menu et traite les commandes suivantes à l'aide des méthodes d'instance de la classe `Image`.

Les images sont représentées par la classe

```
public class Image extends AbstractTree
```

```
1  public abstract class AbstractImage extends BinaryTree<Node>
2
3  /**
4   * Crée this à partir d'un fichier texte (cf a1.arb, ...) et l'affiche
5   * dans une fenêtre. Chaque ligne du fichier est de la forme
6   * (e x1 y1 x2 y2) et indique si on souhaite  éteindre (e=0) ou
7   * allumer (e=1) la région rectangulaire de coordonnées x1, y1, x2, y2.
8   * Le fichier se termine par un e de valeur -1.
9   */
10 public void constructTreeFromFile() { ... }
11
12
13 /**
14  * Sauvegarde, dans un fichier texte, les feuilles de this selon un
15  * format conforme aux fichiers manipulés par la commande
16  * constructTreeFromFile.
17  *
18  * @pre           !this.isEmpty()
19  */
20 public void saveImage() { ... }
21
22
23 /**
24  * @pre           !this.isEmpty()
25  * @return        hauteur de this
26  */
27 public int height() { ... }
28
29
30 /**
31  * @pre           !this.isEmpty()
32  * @return        nombre de nœuds de this
33  */
34 public int numberOfNodes() { ... }
35
36
37 /**
38  * @param x       abscisse du point
39  * @param y       ordonnée du point
40  * @pre           !this.isEmpty()
41  * @return        true, si le point (x, y) est allumé dans this,
42  *                false sinon
43  */
44 public abstract boolean isPixelOn(int x, int y);
```

```
45
46
47  /**
48   * this devient identique à image.
49   *
50   * @pre           !image.isEmpty()
51   * @pre           this != image
52   */
53  public abstract void affect(AbstractImage image);
54
55
56  /**
57   * this devient inverse vidéo de this, pixel par pixel.
58   *
59   * @pre           !this.isEmpty()
60   */
61  public abstract void videoInverse();
62
63
64  /**
65   * this devient rotation de image à 180 degrés
66   *
67   * @param image  image pour rotation
68   * @pre           !image.isEmpty()
69   * @pre           this != image
70   */
71  public abstract void rotate180(AbstractImage image);
72
73
74  /**
75   * this devient image miroir vertical de image.
76   *
77   * @param image  image à agrandir
78   * @pre           !image.isEmpty()
79   * @pre           this != image
80   */
81  public abstract void mirrorV(AbstractImage image);
82
83
84  /**
85   * this devient image miroir horizontal de image.
86   *
87   * @param image  image à agrandir
88   * @pre           !image.isEmpty()
89   * @pre           this != image
```

```
90   */
91  public abstract void mirrorH(AbstractImage image);
92
93
94  /**
95   * this devient quart supérieur gauche de image.
96   *
97   * @param image  image à agrandir
98   * @pre          !image.isEmpty()
99   * @pre          this != image
100  */
101 public abstract void zoomIn(AbstractImage image);
102
103
104 /**
105  * Le quart supérieur gauche de this devient image (réduite),
106  * le reste de this devient éteint.
107  *
108  * @param image  image à réduire
109  * @pre          !image.isEmpty()
110  * @pre          this != image
111  */
112 public abstract void zoomOut(AbstractImage image) ;
113
114
115 /**
116  * this devient l'intersection de image1 et image2 au sens des pixels
117  * allumés.
118  *
119  * @param image1 première image
120  * @param image1 deuxième image
121  * @pre          !image1.isEmpty() && !image2.isEmpty()
122  * @pre          this != image1
123  * @pre          this != image2
124  */
125 public abstract void intersection(AbstractImage image1,
126                                   AbstractImage image2);
127
128
129 /**
130  * this devient l'union image1 et image2 au sens des pixels allumés.
131  *
132  * @param image1 première image
133  * @param image1 deuxième image
134  * @pre          !image1.isEmpty() && !image2.isEmpty()
```

```
135      * @pre          this != image1
136      * @pre          this != image2
137      */
138     public abstract void union(AbstractImage image1,
139                                AbstractImage image2);
140
141
142     /**
143      * Cette fonction ne doit pas utiliser la fonction isPixelOn.
144      *
145      * @pre          !this.isEmpty()
146      * @return       true, si tous les points de la forme (x, x)
147      *               (avec 0 <= x <= 255) sont, ou non, allumés dans this,
148      *               false sinon
149      */
150     public abstract boolean testDiagonal();
151
152
153     /**
154      * @param x1     abscisse du premier point
155      * @param y1     ordonnée du premier point
156      * @param x2     abscisse du deuxième point
157      * @param y2     ordonnée du deuxième point
158      * @pre          !this.isEmpty()
159      * @return       true si les deux points (x1, y1) et (x2, y2) sont
160      *               représentés par la même feuille de this, false sinon
161      */
162     public abstract boolean sameLeaf(int x1, int y1, int x2, int y2);
163
164
165     /**
166      * @param image  seconde image
167      * @pre          !this.isEmpty() && !image.isEmpty()
168      * @pre          this != image
169      * @return       true si this est inclus dans image au sens
170      *               des pixels allumés, false sinon
171      */
172     public abstract boolean isIncludedIn(AbstractImage image);
173
174
175     /**
176      * Affiche this sous forme d'arbre dans une fenêtre externe.
177      *
178      * @pre          !this.isEmpty()
179      */
```

```
180 public void plotTree() { ... }
181
182 }
```

Compléter la classe Image pour définir les 10 méthodes d'instance suivantes : *videoInverse, affect, rotate180, mirrorH, union, zoomIn, zoomOut, testDiagonal, isPixelOn, inIncludedIn*.

Les classes offertes sont disponibles dans le répertoire `/share/l3miage/prg1/tp6/algo` ou `G:\l3miage\prg1\tp6\algo` : *tp6.jar* contient tous les fichiers *.class* et *tp6.tar* contient les fichiers *.java*, ainsi que qu'une vingtaine de fichiers *.arb*.

Écriture d'une mise en œuvre

Compléter la classe *BinaryTree.java* du répertoire `G:\l3miage\prg1\tp6\meo` ou `/share/l3miage/prg1/tp6/meo` pour une mise en œuvre des arbres binaires par une représentation chaînée par références avec pile des pères.

A RENDRE

Lors de la dernière séance de la semaine du 13 novembre 2023 les méthodes d'instance correspondant aux commandes *affect, isPixelOn et union*.

Pour le vendredi 8 décembre 2023 au plus tard les classes *Image* (les 10 commandes) et *BinaryTree* (meo).