

Génie Logiciel partie tests fonctionnels et structurels

Groupe TP41 : Floréal RISSO et Téo TINARRAGE

Question 0

- a) Assurez-vous que vous disposez d'une implémentation en Java de l'algorithme de Dijkstra, que vous avez bien identifié les entrées et les sorties du programme. L'implémentation utilisé est celle proposée dans le sujet.
- b) Mettez au propre la spécification de ce programme. L'implémentation utilisée est celle proposée en exemple, étendue par nous-même. Elle applique l'algorithme *et* réalise les tests sur ses résultats (au lieu d'avoir un script de test qui appelle un programme implémentant l'algorithme, tout est réalisé par un seul programme).
Les fichiers de l'implémentation d'origine sont présents tels quels et nos ajouts se situent dans d'autres classes qui exploitent les précédentes.

Partie Algo de Dijkstra

Entrées : liste non-vide d'*arcs* (*orientés ou non*) *pondérés* (c'est à dire numéro de sommet de départ, numéro de sommet d'arrivée, poids(float)) décrivant un graphe (l'existence d'un sommet est impliquée par l'existence d'un arc l'ayant comme extrémité), ainsi que le *numéro du sommet de départ* pour le calcul des distances, au format JSON*

Sortie : *Distance* entre chaque sommet du graphe et le sommet de départ choisi, ("Distance" désignant la somme des poids des arcs parcourus par le chemin minimisant cette somme)

*structure attendue du JSON :

Pour les données d'entrée

```
{  
    "graphs" : [  
        {  
            "departDijkstra" : <numéro du sommet de départ de l'algo>,  
            "estOriente" : <booléen>  
            "chemins" : [  
                {  
                    "depart" : <numéro sommet de départ>,  
                    "arrivée" : <numéro sommet d'arrivée>,  
                    "distance" : <float>  
                }  
            ]  
        }  
    ]  
}
```

```

        "poids" : <poids de l'arc (flottant)>
    } ,
    ...
]
}
]
```

Partie Test

Entrée : Résultats attendus (oracles) pour l'application de l'algorithme aux données d'entrées spécifiées

Sortie : résultat des tests.

Pour les résultats attendus :

```
{
  "results": [
    "distances" : [...] <liste des distances attendues>
    "parents" : [...] <liste des parents de chaque sommets>
  ]
}
```

Question 1

Le programme de test est test.java. Son fonctionnement est simple, dans un premier temps il lit les jeux de tests et les oracles.

```

private static final String testFile = "./tests.json";
private static final String resultFile = "./results.json";

// La fonction qui lit les jeux de tests écrit dans le fichier "./tests.json".
private static Tester getTests();

// La classe qui contient le contenu du fichier de test.
public static class Tester {
    private Graph[] graphs; // les graphes
    private int[] startDijkstra; // le sommet de départ
}

// La fonction qui lit les oracles
private static Result[] getResults();

// La classe qui contient le résultat attendu d'un jeu de test.
public class Result {
```

```

    private float[] distances;
    private int[] parents;
}

```

Ensuite, on appelle la fonction qui calcule la distance de Dijkstra sur l'ensemble des jeux de tests et on compare le résultat attendu du résultat obtenu.

```

// Parcourt des jeux de tests
for(Graph graph : graphs) {
    // Calcul du resultat
    Result result = TestGraphs.Dijkstra(graph,tester.startDijkstra[i]);
    // Comparaison du resultat au resultat attendu
    TestResult testResult = expectedResults[i].equals(result);
}

// La fonction qui vérifie l'égalité des résultats attendue et calculé.
public TestResult equals(Result r) {
    // Des branchements if / else
    // ...
    return new TestResult("Un message expliquant la différence...");
}

// La classe renournée par equals :
public class TestResult {
    // Les résultats sont les mêmes
    private boolean areEquals;
    // Le message d'erreur
    private String error;
}

// Enfin le print final qui compare les nombres de test réussi et échoué.
System.out.println("Résultat : ["+passed+"/"+i+"]");

```

Question 2

Voici les captures d'écrans montrant les lignes de code exécutés avec le jeu donné dans l'énoncé.

Le jeu de test :

```
{  
  "graphs": [  
    {  
      "estOriente": false,  
      "departDijkstra": 0,  
      "sommets": 6,  
      "chemins": [  
        {  
          "depart": 0,  
          "arrivee": 1,  
          "poids": 2.0  
        },  
        {  
          "depart": 0,  
          "arrivee": 4,  
          "poids": 3.0  
        },  
        {  
          "depart": 1,  
          "arrivee": 4,  
          "poids": 4.0  
        },  
        {  
          "depart": 1,  
          "arrivee": 2,  
          "poids": 5.0  
        },  
        {  
          "depart": 1,  
          "arrivee": 5,  
          "poids": 2.0  
        },  
        {  
          "depart": 2,  
          "arrivee": 3,  
          "poids": 2.0  
        },  
        {  
          "depart": 2,  
          "arrivee": 5,  
          "poids": 4.0  
        },  
        {
```

```
        "depart": 3,
        "arrivee": 4,
        "poids": 2.0
    },
    {
        "depart": 3,
        "arrivee": 5,
        "poids": 5.0
    },
    {
        "depart": 4,
        "arrivee": 5,
        "poids": 2.0
    }
]
}
}
```

Le résultat attendu :

```
{
    "results": [
        {
            "distances" : [0.0, 2.0, 7.0,3.4028235E38,3.0,4.0],
            "parents" : [-1,0,1,-1,0,1]
        }
    ]
}
```

La couverture :



The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** GitHub - dijkstra/src/test/Test.java - Eclipse IDE
- Toolbar:** Contains icons for Save, Undo, Redo, Cut, Copy, Paste, Find, and others.
- Left Margin:** Shows code navigation with markers (red, green, yellow) and line numbers.
- Code Editor:** Displays Java code for a Dijkstra's algorithm implementation. The code includes imports for java.io.FileReader, java.util.ArrayList, and various classes like Edge.java, Graph.java, etc. It defines a Test class with a static method test that reads a JSON file, creates a graph, and performs a Dijkstra search. The code uses annotations like @Test and @Before to mark test cases and setup methods.
- Bottom Status Bar:** Shows the current file (Test.java), line number (115), and character position (1:3414).

```

1 package dijkstra;
2 import java.util.ArrayList;
3 import java.util.Arrays;
4 import java.util.List;
5 import org.json.JSONArray;
6 import org.json.JSONObject;
7 import org.json.JSONUtil;
8
9 public class Test {
10     public static void main(String[] args) {
11         Graphs graphs = new Graphs();
12         TestResult testResult = graphs.readGraphs();
13         Result[] expectedResults = graphs.getExpectedResults();
14
15         for (Result result : testResult) {
16             for (Object o : result) {
17                 if (o instanceof JSONObject) {
18                     JSONObject jResult = (JSONObject) o;
19                     int size = jResult.size();
20                     int[] distances = new int[size];
21                     Result[] parents = new Result[size];
22                     long[] testResults = new long[size];
23
24                     for (Object oResult : jResult) {
25                         if (oResult instanceof JSONObject) {
26                             JSONObject jResult = (JSONObject) oResult;
27                             JSONArray jParents = jResult.getJSONArray("parents");
28                             JSONArray jDistances = jResult.getJSONArray("distances");
29                             double[] parentsDouble = jParents.stream().map(o -> (Object) o).mapToDouble(o -> (Dijkstra) o).toarray();
30                             double[] distancesDouble = jDistances.stream().map(o -> (Object) o).mapToDouble(o -> (Long) o).toarray();
31                             parents[parentsDouble[0]] = new Result(parentsDouble[0], distancesDouble[0]);
32                             distances[distancesDouble[0]] = distancesDouble[0];
33                             testResults[distancesDouble[0]] = distancesDouble[0];
34                         }
35                     }
36                     testResult[i] = new TestResult(convertor.doubleToFloat(distances), convertor.longToInt(parents));
37                 }
38             }
39         }
40         System.out.println(testResult);
41     }
42 }

```

```

1 package dijkstra;
2 import java.util.Iterator;
3
4 public class TestGraphs {
5
6     public static void main(String[] args) {
7         Graph g = new Graph();
8
9         System.out.println("Graph:");
10        g.printGraph();
11
12        System.out.println("Shortest Path Algorithm");
13        System.out.println("Dijkstra Shortest Path");
14        Dijkstra(g, 0);
15
16        System.out.println("Result:");
17        g.printGraph();
18    }
19
20    public static Result dijkstra(Graph g, int startVertex) {
21        // For storing distances after removing vertex from queue
22        float[] distances = new float[g.getVertexCount()];
23        // For storing father id after removing vertex from queue
24        int[] parents = new int[g.getVertexCount()];
25        // For storing vertex count after removing vertex from queue
26        int[] removedVertices = new int[g.getVertexCount()];
27
28        for (int i = 0; i < g.getVertexCount(); i++) {
29            parents[i] = -1;
30            removedVertices[i] = 0;
31        }
32
33        // set up vertex queue
34        PriorityQueue<Vertex> q = new PriorityQueue<Vertex>();
35        for (int i = 0; i < g.getVertexCount(); i++) {
36            if (i != startVertex) {
37                q.add(i);
38            }
39        }
40
41        // add start vertex
42        Vertex node = new Vertex(startVertex);
43        q.add(node);
44        q.add(node);
45
46        // loop through all vertices
47        while (!q.isEmpty()) {
48            // get vertex with smallest distance
49            Vertex u = q.remove();
50
51            for (Vertex v : u.getNeighbors()) {
52                if (v.isVisited() == false) {
53                    v.setDistance(u.getDistance() + 1);
54                    v.setPredecessor(u);
55                }
56            }
57        }
58    }
59
60    public static void main(String[] args) {
61        Graph g = new Graph();
62        g.addVertex(0);
63        g.addVertex(1);
64        g.addVertex(2);
65        g.addVertex(3);
66        g.addVertex(4);
67
68        g.addEdge(0, 1, 5.2f);
69        g.addEdge(0, 2, 10.3f);
70        g.addEdge(1, 2, 10.3f);
71        g.addEdge(1, 3, 5.2f);
72        g.addEdge(2, 3, 5.2f);
73        g.addEdge(3, 4, 8.5f);
74        g.addEdge(4, 2, 7.7f);
75
76        g.printGraph();
77    }
78
79    public static void main(String[] args) {
80        Graph g = new Graph();
81        g.addVertex(0);
82        g.addVertex(1);
83        g.addVertex(2);
84        g.addVertex(3);
85        g.addVertex(4);
86
87        g.addEdge(0, 1, 5.2f);
88        g.addEdge(0, 2, 10.3f);
89        g.addEdge(1, 2, 10.3f);
90        g.addEdge(1, 3, 5.2f);
91        g.addEdge(2, 3, 5.2f);
92        g.addEdge(3, 4, 8.5f);
93        g.addEdge(4, 2, 7.7f);
94
95        g.printGraph();
96    }
97
98    public static void main(String[] args) {
99        Graph g = new Graph();
100       g.addVertex(0);
101       g.addVertex(1);
102       g.addVertex(2);
103       g.addVertex(3);
104       g.addVertex(4);
105
106       g.addEdge(0, 1, 5.2f);
107       g.addEdge(0, 2, 10.3f);
108       g.addEdge(1, 2, 10.3f);
109       g.addEdge(1, 3, 5.2f);
110       g.addEdge(2, 3, 5.2f);
111       g.addEdge(3, 4, 8.5f);
112       g.addEdge(4, 2, 7.7f);
113
114       g.printGraph();
115    }
116
117    public static void main(String[] args) {
118        Graph g = new Graph();
119        g.addVertex(0);
120        g.addVertex(1);
121        g.addVertex(2);
122        g.addVertex(3);
123        g.addVertex(4);
124
125        g.addEdge(0, 1, 5.2f);
126        g.addEdge(0, 2, 10.3f);
127        g.addEdge(1, 2, 10.3f);
128        g.addEdge(1, 3, 5.2f);
129        g.addEdge(2, 3, 5.2f);
130        g.addEdge(3, 4, 8.5f);
131        g.addEdge(4, 2, 7.7f);
132
133        g.printGraph();
134    }
135
136    public static void main(String[] args) {
137        Graph g = new Graph();
138        g.addVertex(0);
139        g.addVertex(1);
140        g.addVertex(2);
141        g.addVertex(3);
142        g.addVertex(4);
143
144        g.addEdge(0, 1, 5.2f);
145        g.addEdge(0, 2, 10.3f);
146        g.addEdge(1, 2, 10.3f);
147        g.addEdge(1, 3, 5.2f);
148        g.addEdge(2, 3, 5.2f);
149        g.addEdge(3, 4, 8.5f);
150        g.addEdge(4, 2, 7.7f);
151
152        g.printGraph();
153    }
154
155    public static void main(String[] args) {
156        Graph g = new Graph();
157        g.addVertex(0);
158        g.addVertex(1);
159        g.addVertex(2);
160        g.addVertex(3);
161        g.addVertex(4);
162
163        g.addEdge(0, 1, 5.2f);
164        g.addEdge(0, 2, 10.3f);
165        g.addEdge(1, 2, 10.3f);
166        g.addEdge(1, 3, 5.2f);
167        g.addEdge(2, 3, 5.2f);
168        g.addEdge(3, 4, 8.5f);
169        g.addEdge(4, 2, 7.7f);
170
171        g.printGraph();
172    }
173
174    public static void main(String[] args) {
175        Graph g = new Graph();
176        g.addVertex(0);
177        g.addVertex(1);
178        g.addVertex(2);
179        g.addVertex(3);
180        g.addVertex(4);
181
182        g.addEdge(0, 1, 5.2f);
183        g.addEdge(0, 2, 10.3f);
184        g.addEdge(1, 2, 10.3f);
185        g.addEdge(1, 3, 5.2f);
186        g.addEdge(2, 3, 5.2f);
187        g.addEdge(3, 4, 8.5f);
188        g.addEdge(4, 2, 7.7f);
189
190        g.printGraph();
191    }
192
193    public static void main(String[] args) {
194        Graph g = new Graph();
195        g.addVertex(0);
196        g.addVertex(1);
197        g.addVertex(2);
198        g.addVertex(3);
199        g.addVertex(4);
200
201        g.addEdge(0, 1, 5.2f);
202        g.addEdge(0, 2, 10.3f);
203        g.addEdge(1, 2, 10.3f);
204        g.addEdge(1, 3, 5.2f);
205        g.addEdge(2, 3, 5.2f);
206        g.addEdge(3, 4, 8.5f);
207        g.addEdge(4, 2, 7.7f);
208
209        g.printGraph();
210    }
211
212    public static void main(String[] args) {
213        Graph g = new Graph();
214        g.addVertex(0);
215        g.addVertex(1);
216        g.addVertex(2);
217        g.addVertex(3);
218        g.addVertex(4);
219
220        g.addEdge(0, 1, 5.2f);
221        g.addEdge(0, 2, 10.3f);
222        g.addEdge(1, 2, 10.3f);
223        g.addEdge(1, 3, 5.2f);
224        g.addEdge(2, 3, 5.2f);
225        g.addEdge(3, 4, 8.5f);
226        g.addEdge(4, 2, 7.7f);
227
228        g.printGraph();
229    }
230
231    public static void main(String[] args) {
232        Graph g = new Graph();
233        g.addVertex(0);
234        g.addVertex(1);
235        g.addVertex(2);
236        g.addVertex(3);
237        g.addVertex(4);
238
239        g.addEdge(0, 1, 5.2f);
240        g.addEdge(0, 2, 10.3f);
241        g.addEdge(1, 2, 10.3f);
242        g.addEdge(1, 3, 5.2f);
243        g.addEdge(2, 3, 5.2f);
244        g.addEdge(3, 4, 8.5f);
245        g.addEdge(4, 2, 7.7f);
246
247        g.printGraph();
248    }
249
250    public static void main(String[] args) {
251        Graph g = new Graph();
252        g.addVertex(0);
253        g.addVertex(1);
254        g.addVertex(2);
255        g.addVertex(3);
256        g.addVertex(4);
257
258        g.addEdge(0, 1, 5.2f);
259        g.addEdge(0, 2, 10.3f);
260        g.addEdge(1, 2, 10.3f);
261        g.addEdge(1, 3, 5.2f);
262        g.addEdge(2, 3, 5.2f);
263        g.addEdge(3, 4, 8.5f);
264        g.addEdge(4, 2, 7.7f);
265
266        g.printGraph();
267    }
268
269    public static void main(String[] args) {
270        Graph g = new Graph();
271        g.addVertex(0);
272        g.addVertex(1);
273        g.addVertex(2);
274        g.addVertex(3);
275        g.addVertex(4);
276
277        g.addEdge(0, 1, 5.2f);
278        g.addEdge(0, 2, 10.3f);
279        g.addEdge(1, 2, 10.3f);
280        g.addEdge(1, 3, 5.2f);
281        g.addEdge(2, 3, 5.2f);
282        g.addEdge(3, 4, 8.5f);
283        g.addEdge(4, 2, 7.7f);
284
285        g.printGraph();
286    }
287
288    public static void main(String[] args) {
289        Graph g = new Graph();
290        g.addVertex(0);
291        g.addVertex(1);
292        g.addVertex(2);
293        g.addVertex(3);
294        g.addVertex(4);
295
296        g.addEdge(0, 1, 5.2f);
297        g.addEdge(0, 2, 10.3f);
298        g.addEdge(1, 2, 10.3f);
299        g.addEdge(1, 3, 5.2f);
300        g.addEdge(2, 3, 5.2f);
301        g.addEdge(3, 4, 8.5f);
302        g.addEdge(4, 2, 7.7f);
303
304        g.printGraph();
305    }
306
307    public static void main(String[] args) {
308        Graph g = new Graph();
309        g.addVertex(0);
310        g.addVertex(1);
311        g.addVertex(2);
312        g.addVertex(3);
313        g.addVertex(4);
314
315        g.addEdge(0, 1, 5.2f);
316        g.addEdge(0, 2, 10.3f);
317        g.addEdge(1, 2, 10.3f);
318        g.addEdge(1, 3, 5.2f);
319        g.addEdge(2, 3, 5.2f);
320        g.addEdge(3, 4, 8.5f);
321        g.addEdge(4, 2, 7.7f);
322
323        g.printGraph();
324    }
325
326    public static void main(String[] args) {
327        Graph g = new Graph();
328        g.addVertex(0);
329        g.addVertex(1);
330        g.addVertex(2);
331        g.addVertex(3);
332        g.addVertex(4);
333
334        g.addEdge(0, 1, 5.2f);
335        g.addEdge(0, 2, 10.3f);
336        g.addEdge(1, 2, 10.3f);
337        g.addEdge(1, 3, 5.2f);
338        g.addEdge(2, 3, 5.2f);
339        g.addEdge(3, 4, 8.5f);
340        g.addEdge(4, 2, 7.7f);
341
342        g.printGraph();
343    }
344
345    public static void main(String[] args) {
346        Graph g = new Graph();
347        g.addVertex(0);
348        g.addVertex(1);
349        g.addVertex(2);
350        g.addVertex(3);
351        g.addVertex(4);
352
353        g.addEdge(0, 1, 5.2f);
354        g.addEdge(0, 2, 10.3f);
355        g.addEdge(1, 2, 10.3f);
356        g.addEdge(1, 3, 5.2f);
357        g.addEdge(2, 3, 5.2f);
358        g.addEdge(3, 4, 8.5f);
359        g.addEdge(4, 2, 7.7f);
360
361        g.printGraph();
362    }
363
364    public static void main(String[] args) {
365        Graph g = new Graph();
366        g.addVertex(0);
367        g.addVertex(1);
368        g.addVertex(2);
369        g.addVertex(3);
370        g.addVertex(4);
371
372        g.addEdge(0, 1, 5.2f);
373        g.addEdge(0, 2, 10.3f);
374        g.addEdge(1, 2, 10.3f);
375        g.addEdge(1, 3, 5.2f);
376        g.addEdge(2, 3, 5.2f);
377        g.addEdge(3, 4, 8.5f);
378        g.addEdge(4, 2, 7.7f);
379
380        g.printGraph();
381    }
382
383    public static void main(String[] args) {
384        Graph g = new Graph();
385        g.addVertex(0);
386        g.addVertex(1);
387        g.addVertex(2);
388        g.addVertex(3);
389        g.addVertex(4);
390
391        g.addEdge(0, 1, 5.2f);
392        g.addEdge(0, 2, 10.3f);
393        g.addEdge(1, 2, 10.3f);
394        g.addEdge(1, 3, 5.2f);
395        g.addEdge(2, 3, 5.2f);
396        g.addEdge(3, 4, 8.5f);
397        g.addEdge(4, 2, 7.7f);
398
399        g.printGraph();
400    }
401
402    public static void main(String[] args) {
403        Graph g = new Graph();
404        g.addVertex(0);
405        g.addVertex(1);
406        g.addVertex(2);
407        g.addVertex(3);
408        g.addVertex(4);
409
410        g.addEdge(0, 1, 5.2f);
411        g.addEdge(0, 2, 10.3f);
412        g.addEdge(1, 2, 10.3f);
413        g.addEdge(1, 3, 5.2f);
414        g.addEdge(2, 3, 5.2f);
415        g.addEdge(3, 4, 8.5f);
416        g.addEdge(4, 2, 7.7f);
417
418        g.printGraph();
419    }
420
421    public static void main(String[] args) {
422        Graph g = new Graph();
423        g.addVertex(0);
424        g.addVertex(1);
425        g.addVertex(2);
426        g.addVertex(3);
427        g.addVertex(4);
428
429        g.addEdge(0, 1, 5.2f);
430        g.addEdge(0, 2, 10.3f);
431        g.addEdge(1, 2, 10.3f);
432        g.addEdge(1, 3, 5.2f);
433        g.addEdge(2, 3, 5.2f);
434        g.addEdge(3, 4, 8.5f);
435        g.addEdge(4, 2, 7.7f);
436
437        g.printGraph();
438    }
439
440    public static void main(String[] args) {
441        Graph g = new Graph();
442        g.addVertex(0);
443        g.addVertex(1);
444        g.addVertex(2);
445        g.addVertex(3);
446        g.addVertex(4);
447
448        g.addEdge(0, 1, 5.2f);
449        g.addEdge(0, 2, 10.3f);
450        g.addEdge(1, 2, 10.3f);
451        g.addEdge(1, 3, 5.2f);
452        g.addEdge(2, 3, 5.2f);
453        g.addEdge(3, 4, 8.5f);
454        g.addEdge(4, 2, 7.7f);
455
456        g.printGraph();
457    }
458
459    public static void main(String[] args) {
460        Graph g = new Graph();
461        g.addVertex(0);
462        g.addVertex(1);
463        g.addVertex(2);
464        g.addVertex(3);
465        g.addVertex(4);
466
467        g.addEdge(0, 1, 5.2f);
468        g.addEdge(0, 2, 10.3f);
469        g.addEdge(1, 2, 10.3f);
470        g.addEdge(1, 3, 5.2f);
471        g.addEdge(2, 3, 5.2f);
472        g.addEdge(3, 4, 8.5f);
473        g.addEdge(4, 2, 7.7f);
474
475        g.printGraph();
476    }
477
478    public static void main(String[] args) {
479        Graph g = new Graph();
480        g.addVertex(0);
481        g.addVertex(1);
482        g.addVertex(2);
483        g.addVertex(3);
484        g.addVertex(4);
485
486        g.addEdge(0, 1, 5.2f);
487        g.addEdge(0, 2, 10.3f);
488        g.addEdge(1, 2, 10.3f);
489        g.addEdge(1, 3, 5.2f);
490        g.addEdge(2, 3, 5.2f);
491        g.addEdge(3, 4, 8.5f);
492        g.addEdge(4, 2, 7.7f);
493
494        g.printGraph();
495    }
496
497    public static void main(String[] args) {
498        Graph g = new Graph();
499        g.addVertex(0);
500        g.addVertex(1);
501        g.addVertex(2);
502        g.addVertex(3);
503        g.addVertex(4);
504
505        g.addEdge(0, 1, 5.2f);
506        g.addEdge(0, 2, 10.3f);
507        g.addEdge(1, 2, 10.3f);
508        g.addEdge(1, 3, 5.2f);
509        g.addEdge(2, 3, 5.2f);
510        g.addEdge(3, 4, 8.5f);
511        g.addEdge(4, 2, 7.7f);
512
513        g.printGraph();
514    }
515
516    public static void main(String[] args) {
517        Graph g = new Graph();
518        g.addVertex(0);
519        g.addVertex(1);
520        g.addVertex(2);
521        g.addVertex(3);
522        g.addVertex(4);
523
524        g.addEdge(0, 1, 5.2f);
525        g.addEdge(0, 2, 10.3f);
526        g.addEdge(1, 2, 10.3f);
527        g.addEdge(1, 3, 5.2f);
528        g.addEdge(2, 3, 5.2f);
529        g.addEdge(3, 4, 8.5f);
530        g.addEdge(4, 2, 7.7f);
531
532        g.printGraph();
533    }
534
535    public static void main(String[] args) {
536        Graph g = new Graph();
537        g.addVertex(0);
538        g.addVertex(1);
539        g.addVertex(2);
540        g.addVertex(3);
541        g.addVertex(4);
542
543        g.addEdge(0, 1, 5.2f);
544        g.addEdge(0, 2, 10.3f);
545        g.addEdge(1, 2, 10.3f);
546        g.addEdge(1, 3, 5.2f);
547        g.addEdge(2, 3, 5.2f);
548        g.addEdge(3, 4, 8.5f);
549        g.addEdge(4, 2, 7.7f);
550
551        g.printGraph();
552    }
553
554    public static void main(String[] args) {
555        Graph g = new Graph();
556        g.addVertex(0);
557        g.addVertex(1);
558        g.addVertex(2);
559        g.addVertex(3);
560        g.addVertex(4);
561
562        g.addEdge(0, 1, 5.2f);
563        g.addEdge(0, 2, 10.3f);
564        g.addEdge(1, 2, 10.3f);
565        g.addEdge(1, 3, 5.2f);
566        g.addEdge(2, 3, 5.2f);
567        g.addEdge(3, 4, 8.5f);
568        g.addEdge(4, 2, 7.7f);
569
570        g.printGraph();
571    }
572
573    public static void main(String[] args) {
574        Graph g = new Graph();
575        g.addVertex(0);
576        g.addVertex(1);
577        g.addVertex(2);
578        g.addVertex(3);
579        g.addVertex(4);
580
581        g.addEdge(0, 1, 5.2f);
582        g.addEdge(0, 2, 10.3f);
583        g.addEdge(1, 2, 10.3f);
584        g.addEdge(1, 3, 5.2f);
585        g.addEdge(2, 3, 5.2f);
586        g.addEdge(3, 4, 8.5f);
587        g.addEdge(4, 2, 7.7f);
588
589        g.printGraph();
590    }
591
592    public static void main(String[] args) {
593        Graph g = new Graph();
594        g.addVertex(0);
595        g.addVertex(1);
596        g.addVertex(2);
597        g.addVertex(3);
598        g.addVertex(4);
599
600        g.addEdge(0, 1, 5.2f);
601        g.addEdge(0, 2, 10.3f);
602        g.addEdge(1, 2, 10.3f);
603        g.addEdge(1, 3, 5.2f);
604        g.addEdge(2, 3, 5.2f);
605        g.addEdge(3, 4, 8.5f);
606        g.addEdge(4, 2, 7.7f);
607
608        g.printGraph();
609    }
610
611    public static void main(String[] args) {
612        Graph g = new Graph();
613        g.addVertex(0);
614        g.addVertex(1);
615        g.addVertex(2);
616        g.addVertex(3);
617        g.addVertex(4);
618
619        g.addEdge(0, 1, 5.2f);
620        g.addEdge(0, 2, 10.3f);
621        g.addEdge(1, 2, 10.3f);
622        g.addEdge(1, 3, 5.2f);
623        g.addEdge(2, 3, 5.2f);
624        g.addEdge(3, 4, 8.5f);
625        g.addEdge(4, 2, 7.7f);
626
627        g.printGraph();
628    }
629
630    public static void main(String[] args) {
631        Graph g = new Graph();
632        g.addVertex(0);
633        g.addVertex(1);
634        g.addVertex(2);
635        g.addVertex(3);
636        g.addVertex(4);
637
638        g.addEdge(0, 1, 5.2f);
639        g.addEdge(0, 2, 10.3f);
640        g.addEdge(1, 2, 10.3f);
641        g.addEdge(1, 3, 5.2f);
642        g.addEdge(2, 3, 5.2f);
643        g.addEdge(3, 4, 8.5f);
644        g.addEdge(4, 2, 7.7f);
645
646        g.printGraph();
647    }
648
649    public static void main(String[] args) {
650        Graph g = new Graph();
651        g.addVertex(0);
652        g.addVertex(1);
653        g.addVertex(2);
654        g.addVertex(3);
655        g.addVertex(4);
656
657        g.addEdge(0, 1, 5.2f);
658        g.addEdge(0, 2, 10.3f);
659        g.addEdge(1, 2, 10.3f);
660        g.addEdge(1, 3, 5.2f);
661        g.addEdge(2, 3, 5.2f);
662        g.addEdge(3, 4, 8.5f);
663        g.addEdge(4, 2, 7.7f);
664
665        g.printGraph();
666    }
667
668    public static void main(String[] args) {
669        Graph g = new Graph();
670        g.addVertex(0);
671        g.addVertex(1);
672        g.addVertex(2);
673        g.addVertex(3);
674        g.addVertex(4);
675
676        g.addEdge(0, 1, 5.2f);
677        g.addEdge(0, 2, 10.3f);
678        g.addEdge(1, 2, 10.3f);
679        g.addEdge(1, 3, 5.2f);
680        g.addEdge(2, 3, 5.2f);
681        g.addEdge(3, 4, 8.5f);
682        g.addEdge(4, 2, 7.7f);
683
684        g.printGraph();
685    }
686
687    public static void main(String[] args) {
688        Graph g = new Graph();
689        g.addVertex(0);
690        g.addVertex(1);
691        g.addVertex(2);
692        g.addVertex(3);
693        g.addVertex(4);
694
695        g.addEdge(0, 1, 5.2f);
696        g.addEdge(0, 2, 10.3f);
697        g.addEdge(1, 2, 10.3f);
698        g.addEdge(1, 3, 5.2f);
699        g.addEdge(2, 3, 5.2f);
700        g.addEdge(3, 4, 8.5f);
701        g.addEdge(4, 2, 7.7f);
702
703        g.printGraph();
704    }
705
706    public static void main(String[] args) {
707        Graph g = new Graph();
708        g.addVertex(0);
709        g.addVertex(1);
710        g.addVertex(2);
711        g.addVertex(3);
712        g.addVertex(4);
713
714        g.addEdge(0, 1, 5.2f);
715        g.addEdge(0, 2, 10.3f);
716        g.addEdge(1, 2, 10.3f);
717        g.addEdge(1, 3, 5.2f);
718        g.addEdge(2, 3, 5.2f);
719        g.addEdge(3, 4, 8.5f);
720        g.addEdge(4, 2, 7.7f);
721
722        g.printGraph();
723    }
724
725    public static void main(String[] args) {
726        Graph g = new Graph();
727        g.addVertex(0);
728        g.addVertex(1);
729        g.addVertex(2);
730        g.addVertex(3);
731        g.addVertex(4);
732
733        g.addEdge(0, 1, 5.2f);
734        g.addEdge(0, 2, 10.3f);
735        g.addEdge(1, 2, 10.3f);
736        g.addEdge(1, 3, 5.2f);
737        g.addEdge(2, 3, 5.2f);
738        g.addEdge(3, 4, 8.5f);
739        g.addEdge(4, 2, 7.7f);
740
741        g.printGraph();
742    }
743
744    public static void main(String[] args) {
745        Graph g = new Graph();
746        g.addVertex(0);
747        g.addVertex(1);
748        g.addVertex(2);
749        g.addVertex(3);
750        g.addVertex(4);
751
752        g.addEdge(0, 1, 5.2f);
753        g.addEdge(0, 2, 10.3f);
754        g.addEdge(1, 2, 10.3f);
755        g.addEdge(1, 3, 5.2f);
756        g.addEdge(2, 3, 5.2f);
757        g.addEdge(3, 4, 8.5f);
758        g.addEdge(4, 2, 7.7f);
759
760        g.printGraph();
761    }
762
763    public static void main(String[] args) {
764        Graph g = new Graph();
765        g.addVertex(0);
766        g.addVertex(1);
767        g.addVertex(2);
768        g.addVertex(3);
769        g.addVertex(4);
770
771        g.addEdge(0, 1, 5.2f);
772        g.addEdge(0, 2, 10.3f);
773        g.addEdge(1, 2, 10.3f);
774        g.addEdge(1, 3, 5.2f);
775        g.addEdge(2, 3, 5.2f);
776        g.addEdge(3, 4, 8.5f);
777        g.addEdge(4, 2, 7.7f);
778
779        g.printGraph();
780    }
781
782    public static void main(String[] args) {
783        Graph g = new Graph();
784        g.addVertex(0);
785        g.addVertex(1);
786        g.addVertex(2);
787        g.addVertex(3);
788        g.addVertex(4);
789
790        g.addEdge(0, 1, 5.2f);
791        g.addEdge(0, 2, 10.3f);
792        g.addEdge(1, 2, 10.3f);
793        g.addEdge(1, 3, 5.2f);
794        g.addEdge(2, 3, 5.2f);
795        g.addEdge(3, 4, 8.5f);
796        g.addEdge(4, 2, 7.7f);
797
798        g.printGraph();
799    }
800
801    public static void main(String[] args) {
802        Graph g = new Graph();
803        g.addVertex(0);
804        g.addVertex(1);
805        g.addVertex(2);
806        g.addVertex(3);
807        g.addVertex(4);
808
809        g.addEdge(0, 1, 5.2f);
810        g.addEdge(0, 2, 10.3f);
811        g.addEdge(1, 2, 10.3f);
812        g.addEdge(1, 3, 5.2f);
813        g.addEdge(2, 3, 5.2f);
814        g.addEdge(3, 4, 8.5f);
815        g.addEdge(4, 2, 7.7f);
816
817        g.printGraph();
818    }
819
820    public static void main(String[] args) {
821        Graph g = new Graph();
822        g.addVertex(0);
823        g.addVertex(1);
824        g.addVertex(2);
825        g.addVertex(3);
826        g.addVertex(4);
827
828        g.addEdge(0, 1, 5.2f);
829        g.addEdge(0, 2, 10.3f);
830        g.addEdge(1, 2, 10.3f);
831        g.addEdge(1, 3, 5.2f);
832        g.addEdge(2, 3, 5.2f);
833        g.addEdge(3, 4, 8.5f);
834        g.addEdge(4, 2, 7.7f);
835
836        g.printGraph();
837    }
838
839    public static void main(String[] args) {
840        Graph g = new Graph();
841        g.addVertex(0);
842        g.addVertex(1);
843        g.addVertex(2);
844        g.addVertex(3);
845        g.addVertex(4);
846
847        g.addEdge(0, 1, 5.2f);
848        g.addEdge(0, 2, 10.3f);
849        g.addEdge(1, 2, 10.3f);
850        g.addEdge(1, 3, 5.2f);
851        g.addEdge(2, 3, 5.2f);
852        g.addEdge(3, 4, 8.5f);
853        g.addEdge(4, 2, 7.7f);
854
855        g.printGraph();
856    }
857
858    public static void main(String[] args) {
859        Graph g = new Graph();
860        g.addVertex(0);
861        g.addVertex(1);
862        g.addVertex(2);
863        g.addVertex(3);
864        g.addVertex(4);
865
866        g.addEdge(0, 1, 5.2f);
867        g.addEdge(0, 2, 10.3f);
868        g.addEdge(1, 2, 10.3f);
869        g.addEdge(1, 3, 5.2f);
870        g.addEdge(2, 3, 5.2f);
871        g.addEdge(3, 4, 8.5f);
872        g.addEdge(4, 2, 7.7f);
873
874        g.printGraph();
875    }
876
877    public static void main(String[] args) {
878        Graph g = new Graph();
879        g.addVertex(0);
880        g.addVertex(1);
881        g.addVertex(2);
882        g.addVertex(3);
883        g.addVertex(4);
884
885        g.addEdge(0, 1, 5.2f);
886        g.addEdge(0, 2, 10.3f);
887        g.addEdge(1, 2, 10.3f);
888        g.addEdge(1, 3, 5.2f);
889        g.addEdge(2, 3, 5.2f);
890        g.addEdge(3, 4, 8.5f);
891        g.addEdge(4, 2, 7.7f);
892
893        g.printGraph();
894    }
895
896    public static void main(String[] args) {
897        Graph g = new Graph();
898        g.addVertex(0);
899        g.addVertex(1);
900        g.addVertex(2);
901        g.addVertex(3);
902        g.addVertex(4);
903
904        g.addEdge(0, 1, 5.2f);
905        g.addEdge(0, 2, 10.3f);
906        g.addEdge(1, 2, 10.3f);
907        g.addEdge(1, 3, 5.2f);
908        g.addEdge(2, 3, 5.2f);
909        g.addEdge(3, 4, 8.5f);
910        g.addEdge(4, 2, 7.7f);
911
912        g.printGraph();
913    }
914
915    public static void main(String[] args) {
916        Graph g = new Graph();
917        g.addVertex(0);
918        g.addVertex(1);
919        g.addVertex(2);
920        g.addVertex(3);
921        g.addVertex(4);
922
923        g.addEdge(0, 1, 5.2f);
924        g.addEdge(0, 2, 10.3f);
925        g.addEdge(1, 
```

The screenshot shows the Eclipse IDE interface with the TestGraphs.java file open. The code implements the Dijkstra's algorithm. It starts by setting up a vertex queue and initializing distances and parents arrays. It then iterates through all vertices in the queue, updating their distances and parents based on the shortest path found so far. Finally, it prints the shortest paths from the start vertex to all other vertices.

```
37     parents[i] = -1;
38 }
39 // set up vertex queue
40 Queue<Vertex> Q = new PriorityQueue<Vertex>();
41 for (int i = 0; i < g.getVCount(); i++) {
42     if (i != startVertex) {
43         Q.add(new Vertex(i));
44     }
45 }
46 // add start vertex
47 Vertex node = new Vertex(startVertex);
48 node.setDistance(0);
49 Q.add(node);
50
51 while (!Q.isEmpty()) {
52     // get vertex with shortest distance
53     Vertex u = Q.remove();
54     distances[u.getId()] = u.getDistance();
55
56     // iterate through all vertices
57     Iterator<Edge> it = g.neighbours(u.getId()).iterator();
58     while (it.hasNext()) {
59         Edge e = it.next();
60         Iterator<Vertex> it2 = e.iterator();
61         while (it2.hasNext()) {
62             Vertex v = it2.next();
63             if (v.isVisited()) {
64                 continue;
65             }
66             if (e.getEndPoint() == v.getId()) {
67                 if (e.getEndPoint() == v.getId()) {
68                     if (v.getDistance() > u.getDistance() + e.getWeight()) {
69                         v.setDistance(u.getDistance() + e.getWeight());
70                         v.setParent(u.getId());
71                         parents[v.getId()] = v.getParent().getId();
72                     }
73                 }
74             }
75         }
76     }
77 }
78
79 // print final shortest paths
80
81 // print final shortest paths
82 System.out.println("vertex\tdistance\tparent\tvertex");
83 for (int i = 0; i < g.getVCount(); i++) {
84     System.out.println(i + "\t" + distances[i] + "\t" + parents[i]);
85 }
86
87 return new Result(distances, parents);
88 }
89 }
90 }
```

The screenshot shows the Eclipse IDE interface with the Vertex.java file open. This class implements the Comparable interface and represents a vertex in the graph. It has fields for id, distance, and parent. The constructor takes an int id. The get() and setId() methods return and set the id respectively. The getDistance() and setDistance() methods return and set the distance respectively. The getParent() and setParent() methods return and set the parent respectively. The compareTo() method compares the distance of this vertex with another vertex.

```
1 package dijkstra;
2 public class Vertex implements Comparable<Vertex>{
3     private int id;
4     private float distance;
5     private Vertex parent;
6
7     public Vertex() {
8         distance = Float.MAX_VALUE; // "infinity"
9         parent = null;
10    }
11
12    public Vertex(int id){
13        this.id = id;
14        distance = Float.MAX_VALUE; // "infinity"
15        parent = null;
16    }
17
18    public int getId() {
19        return id;
20    }
21
22    public void setId(int id) {
23        this.id = id;
24    }
25
26    public float getDistance() {
27        return distance;
28    }
29
30    public void setDistance(float distance) {
31        this.distance = distance;
32    }
33
34    public Vertex getParent() {
35        return parent;
36    }
37
38    public void setParent(Vertex parent){
39        this.parent = parent;
40    }
41
42    public int compareTo(Vertex other) {
43        return Double.compare(this.distance, other.distance);
44    }
45 }
```

The screenshot shows the Eclipse IDE interface with the Edge.java file open. The code defines a class Edge that implements Comparable<Edge>. It has private fields startPoint, endPoint, and weight. The constructor takes startPoint, endPoint, and weight as parameters and initializes the fields. The class includes methods to get startPoint, endPoint, and weight, and to check if two edges are equal based on their start and end points. It also includes a compareTo method and a toString method that returns a string in the format "startPoint->endPoint (weight)".

```
1 package dijkstra;
2
3 public class Edge implements Comparable<Edge>{
4     private int startPoint;
5     private int endPoint;
6     private float weight;
7
8     public Edge(int startPoint, int endPoint, float weight) {
9         this.startPoint = startPoint;
10        this.endPoint = endPoint;
11        this.weight = weight;
12    }
13
14    public int get startPoint() {
15        return startPoint;
16    }
17
18    public int get endPoint() {
19        return endPoint;
20    }
21
22    public float get weight() {
23        return weight;
24    }
25
26    public boolean equals(Edge other) {
27        if (this.startPoint == other.startPoint) {
28            if (this.endPoint == other.endPoint) {
29                return true;
30            }
31        }
32        return false;
33    }
34
35    public int compareTo(Edge other) {
36        return Double.compare(this.weight, other.weight);
37    }
38
39    public String toString() {
40        return startPoint + "-" + endPoint + " (" + weight + ")";
41    }
42
43 }
```

The screenshot shows the Eclipse IDE interface with the Graph.java file open. The code defines a class Graph that uses a Priority Queue to manage edges. It has a private field adj of type Priority Queue<Edge>[] and a public field getCount(). The constructor initializes adj with a size of vCount. The addEdge method adds a new edge between vertices i and j with weight. The removeEdge method removes the edge between vertices i and j. The hasEdge method checks if there is an edge between vertices i and j. The neighbours method returns a Priority Queue of edges for a given vertex. The printEdges method prints all edges in the graph.

```
1 package dijkstra;
2
3 import java.util.Iterator;
4
5 public class Graph {
6     private int vCount;
7     private Priority Queue<Edge>[] adj;
8
9     public int getCount() {
10        return vCount;
11    }
12
13    @SuppressWarnings("unchecked")
14    public Graph(int vCount) {
15        this.vCount = vCount;
16        adj = new Priority Queue[vCount];
17        for (int i = 0; i < vCount; i++) {
18            adj[i] = new Priority Queue<Edge>();
19        }
20    }
21
22    public void addEdge(int i, int j, float weight) {
23        adj[i].add(new Edge(i, j, weight));
24    }
25
26    public void removeEdge(int i, int j) {
27        adj[i].remove(new Edge(i, j));
28    }
29
30    public boolean hasEdge(Edge e) {
31        return adj[e.get startPoint()].contains(e);
32    }
33
34    public Priority Queue<Edge> neighbours(int vertex) {
35        return adj[vertex];
36    }
37
38    public void printEdges() {
39        for (int i = 0; i < vCount; i++) {
40            Iterator<Edge> it = adj[i].iterator();
41            while (it.hasNext()) {
42                if (it.next() != null) {
43                    it.remove();
44                }
45            }
46        }
47    }
48
49    public void print() {
50        for (int i = 0; i < vCount; i++) {
51            Iterator<Edge> it = adj[i].iterator();
52            while (it.hasNext()) {
53                System.out.println(it.next());
54            }
55        }
56    }
57 }
```

The screenshot shows the Eclipse IDE interface with the Graph.java file open. The code defines a class Edge with methods for adding edges, getting neighbors, printing edges, and building a string representation of the graph's edges.

```
1 package dijkstra;
2 import java.util.ArrayList;
3 import java.util.Iterator;
4 import java.util.List;
5 import java.util.Set;
6 import java.util.SortedSet;
7 import java.util.TreeSet;
8
9 public class Edge {
10     private int id;
11     private Vertex start;
12     private Vertex end;
13     private float weight;
14
15     public Edge(int id, Vertex start, Vertex end, float weight) {
16         this.id = id;
17         this.start = start;
18         this.end = end;
19         this.weight = weight;
20     }
21
22     public void addNeighbour(Vertex v) {
23         adj[id].add(new Edge(id, v, weight));
24     }
25
26     public void removeNeighbour(Vertex v) {
27         adj[id].remove(new Edge(id, v, weight));
28     }
29
30     public Vertex getEndPoint() {
31         return end;
32     }
33
34     public void removeEdge(Edge e) {
35         Iterator<Edge> it = adj[id].iterator();
36         Edge other = new Edge(id, 0, 0);
37         while (it.hasNext()) {
38             if (it.next().equals(other)) {
39                 it.remove();
40                 return;
41             }
42         }
43     }
44
45     public boolean hasEdge(Edge e) {
46         return adj[e.getId()].contains(e);
47     }
48
49     public PrintStream neighbours(int vertex) {
50         return adj[vertex];
51     }
52
53     public void print() {
54         for (int i = 0; i < id; i++) {
55             for (Vertex v : edges[i]) {
56                 Iterator<Edge> it = edges[i].iterator();
57                 while (it.hasNext()) {
58                     if (it.next().getEndPoint() == v) {
59                         System.out.print(it.next() + " ");
60                     }
61                 }
62             }
63         }
64     }
65
66     public String toString() {
67         StringBuilder builder = new StringBuilder();
68         for (int i = 0; i < id; i++) {
69             for (Vertex v : edges[i]) {
70                 Iterator<Edge> it = edges[i].iterator();
71                 while (it.hasNext()) {
72                     if (it.next().getEndPoint() == v) {
73                         builder.append(it.next() + " ");
74                     }
75                 }
76             }
77         }
78         return builder.toString();
79     }
80 }
```

The screenshot shows the Eclipse IDE interface with the Result.java file open. The code defines a class Result with methods for getting distances and parents, and a testToString method for generating a readable string representation of the results.

```
1 package dijkstra;
2 import utilities.TestResult;
3
4 public class Result {
5     private float[] distances;
6     private int[] parents;
7
8     public Result(float[] distances, int[] parents) {
9         super();
10        this.distances = distances;
11        this.parents = parents;
12    }
13
14    public float[] getDistances() {
15        return distances;
16    }
17
18    public int[] getParents() {
19        return parents;
20    }
21
22    public String testToString() {
23        StringBuilder builder = new StringBuilder("Vertices\\Distance\\Parent Vertice\\n");
24        if (distances.length != distances.length) {
25            builder.append("TestResult\\Difference taille des tableaux distances\\n");
26        }
27        if (parents.length != parents.length) {
28            builder.append("TestResult\\Difference taille des tableaux parents\\n");
29        }
30        if (parents.length != distances.length) {
31            builder.append("TestResult\\Difference entre les tailles des tableaux distances et parents\\n");
32        }
33        // Compare the values
34        for (int i=0; i<parents.length;i++) {
35            if (distances[i] != expectedResult.distances[i]) {
36                builder.append("TestResult\\Difference à l'indice : "+i+" result.distances["+i+"] et expectedResult.distances["+i+"]\\n");
37            }
38            if (parents[i] != expectedResult.parents[i]) {
39                builder.append("TestResult\\Difference à l'indice : "+i+" result.parents["+i+"] et expectedResult.parents["+i+"]\\n");
40            }
41        }
42        return builder.toString();
43    }
44
45    public TestResult equals(Result r) {
46        if (r.distances.length!=distances.length) {
47            return new TestResult("Difference taille des tableaux distances");
48        }
49        if (r.parents.length!=parents.length) {
50            return new TestResult("Difference taille des tableaux parents");
51        }
52        if (r.distances.length!=parents.length) {
53            return new TestResult("Difference entre les tailles des tableaux distances et parents");
54        }
55    }
56 }
```

```

1 package utilities;
2
3 public class Converter {
4     /**
5      * Convert a Long[] to a int[]
6      * @param parents
7      * @return
8     */
9    public static int[] longToInt(Long[] parents) {
10        int size = parents.length;
11        int[] iTab = new int[size];
12        for (int i = 0; i < size; i++) {
13            iTab[i] = parents[i].intValue();
14        }
15        return iTab;
16    }
17
18
19
20    /**
21     * Convert a double[] to a float[]
22     * @param distances
23     * @return
24     */
25
26    public static float[] doubleToInt(double[] distances) {
27        int size = distances.length;
28        float[] fTab = new float[size];
29        for (int i = 0; i < size; i++) {
30            fTab[i] = distances[i];
31        }
32        return fTab;
33    }
34
35
36

```

Question 3

Dans la version originale de l'implémentation, qui nous est donnée dans le sujet, n'importe quelle suite de test passait par toutes les instructions, à l'exception de certains fonctions utilitaires qui ne sont jamais appelées.

Cependant, dans notre version (qui ajoute un système de lecture d'entrée sous forme de fichier JSON, et la comparaison de la sortie avec la sortie attendue, ce qui permet l'automatisation des tests, là où avec l'implémentatation initiale les données d'entrée devaient se trouver dans le code directement), Certaines instructions ne pourront pas être exécutées, notamment celles liées au contrôle d'erreur : en effet, on trouve beaucoup de conditions de la forme “si erreur, afficher quelque chose, et terminer le programme” : ainsi, s'il n'y a pas d'erreur les instructions d'affichage ne sont pas couvertes, s'il y a erreur le reste du programme n'est pas exécuté.

Autre exemple du même type, toutes les séquences de lecture/écriture de fichier impliquent l'utilisation d'un bloc try/catch :

- si une exception est levée avant la fin du bloc try, celui-ci est interrompu, les dernières instructions ne sont donc pas couvertes
- si aucune exception n'est levée, le bloc catch n'est jamais exécuté
Par conséquent, tout bloc try/catch implique qu'une certaine portion de code ne sera pas exécutée.

Globalement il est impossible de passer par toutes les instructions du programme, cependant les instructions servant à réaliser l'algorithme en lui-même

peuvent être toutes traversées par plus ou moins n'importe quelle suite de tests.

Question 4

Préembule

Java restreint fortement les possibilités dans les tests, car les types sont static et fort, ainsi il n'est pas possible de donner un mauvais type ou une valeur trop grande. Alors nous ne pourront pas tester les classes invalides, ni un nombre de sommet inférieure ou égale à zéro. Mais nous allons testé les poids (négatifs, nul, positif) et si un graphe est orienté ou non.

Partition des domaines des entrées

Variable	poid<0.0	poid=0	poid>0.0	poid in [-inf,inf]
Orienté	Test 2		Test 1	Test4
Non orienté		Test 3		Test 5

(cf. voir tests.json et results.json)

Question 5

Préembule

Il est difficile de trouver des mutants très pertinent vu la simplicité du code et le faible nombre d'instructions.

Premier mutant :

```
// TestGraph.java ligne 71
// ">" devient ">="
if (v.getDistance() >= u.getDistance() + e.getWeight()) {
    v.setDistance(u.getDistance() + e.getWeight());
    v.setParent(u);
    parents[v.getId()] = v.getParent().getId();
}
```

Ce mutant est choisi, car il est souvent compliqué de choisir entre “supérieure >” et “supérieure ou égale >=”.

[V] - Test : 0

```

[X] - Test : 1
|----> Difference à l'indice : 2 result.parents=1 et expectedResult.parents=0
[X] - Test : 2
|----> Difference à l'indice : 3 result.parents=1 et expectedResult.parents=2
[X] - Test : 3
|----> Difference à l'indice : 6 result.parents=7 et expectedResult.parents=5
[X] - Test : 4
|----> Difference à l'indice : 5 result.parents=4 et expectedResult.parents=-1
Resultat : [1/5]

```

Seul le test zéro passe, c'est normal, car tous les sommets ont des valeurs différentes et aucune somme ne donne le même résultat. ### Deuxième mutant :

```

// TestGraph.java ligne 67
// "!=" devient "=="
```



```

// check if vertex was visited already
if (e.getEndPoint() == v.getId()) {
    continue;
}

```

Ce mutant est choisi, car il est fréquent de se tromper dans les opérations logiques.

```

[X] - Test : 0
|----> Difference à l'indice : 1 result.distances=7.5 et expectedResult.distances=5.2
[X] - Test : 1
|----> Difference à l'indice : 1 result.distances=-4.3 et expectedResult.distances=-2.1
[X] - Test : 2
|----> Difference à l'indice : 3 result.parents=0 et expectedResult.parents=2
[X] - Test : 3
|----> Difference à l'indice : 1 result.distances=-8.0 et expectedResult.distances=2.0
[X] - Test : 4
|----> Difference à l'indice : 1 result.distances=-3.0 et expectedResult.distances=1.0
Resultat : [0/5]

```

Aucun test ne passe, ce qui est aussi normal. Ici on ne calcule plus le plus court chemin, car on peut passer dans des boucles.

Troisième mutant :

```

// TestGraph.java ligne 72
// le "+" devient "-" v.setDistance(u.getDistance() - e.getWeight());
```



```

if (v.getDistance() > u.getDistance() + e.getWeight()) {
    v.setDistance(u.getDistance() - e.getWeight());
}

```

```

    v.setParent(u);
    parents[v.getId()] = v.getParent().getId();
}

```

Ce mutant est choisi, car il est intéressant de vérifier s'il n'y a pas une erreur dans l'implémentation.

```

[X] - Test : 0
|----> Difference à l'indice : 1 result.distances=-11.8 et expectedResult.distances=5.2
[X] - Test : 1
|----> Difference à l'indice : 1 result.distances=2.1 et expectedResult.distances=-2.1
[V] - Test : 2
[X] - Test : 3
|----> Difference à l'indice : 1 result.distances=-2.0 et expectedResult.distances=2.0
[X] - Test : 4
|----> Difference à l'indice : 1 result.distances=-1.0 et expectedResult.distances=1.0
Resultat : [1/5]

```

Seul le test deux est validé, car il n'y a que des zéros et zéro est le neutre de l'addition.

```

if (v.getDistance() > u.getDistance() + e.getWeight()) {
    v.setDistance(u.getDistance() + e.getWeight());
    v.setParent(v);
    parents[v.getId()] = v.getParent().getId();
}

```

Ce mutant est choisi, car les noms des variables sont mal choisies “u” et “v” sont deux lettres très ressemblante, il est facile de les confondre.

```

[X] - Test : 0
|----> Difference à l'indice : 1 result.parents=1 et expectedResult.parents=0
[X] - Test : 1
|----> Difference à l'indice : 1 result.parents=1 et expectedResult.parents=0
[X] - Test : 2
|----> Difference à l'indice : 1 result.parents=1 et expectedResult.parents=0
[X] - Test : 3
|----> Difference à l'indice : 1 result.parents=1 et expectedResult.parents=0
[X] - Test : 4
|----> Difference à l'indice : 1 result.parents=1 et expectedResult.parents=0
Resultat : [0/5]

```

Le mutant démontre qu'il n'y a pas eu d'erreur.