

# Génie Logiciel partie tests fonctionnels et structurels

Groupe TP51 : 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>,
                "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 résultat
    Result result =
        TestGraphs.Dijkstra(graph,tester.startDijkstra[i]);
    // Comparaison du résultat au résultat attendu
    TestResult testResult = expectedResults[i].equals(result);
}

```

La fonction de comparaison donne de nombreuses informations sur les différences afin d'aider le débogage et à la fin du programme il est indiqué le nombre de test réussi sur le nombre de test tenté.

```

// 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("Resultat : ["+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": 5,
          "poids": 3.0
        }
      ]
    }
  ]
}
```

```

{
    "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 :**



GitHub - dijkstra/src/test/Test.java - Eclipse IDE

```
1 package test;
2
3 import java.io.FileReader;
4
5 public class Test {
6
7     public static class Tester {
8         private Graph[] graphs;
9         private int[] startDijkstra;
10    public Tester(Graph[] graphs, int[] startDijkstra) {
11        this.graphs = graphs;
12        this.startDijkstra = startDijkstra;
13    }
14}
15
16    private static final String testFile = "./tl.json";
17    private static final String resultFile = "./result.json";
18
19    private void addPath(Graph graph, JSONArray chemins, Boolean isOriented) {
20        for(Object oChemin : chemins) {
21            JSONObject jChemin = (JSONObject) oChemin;
22            Long depart = (Long) jChemin.get("depart");
23            Long arrivee = (Long) jChemin.get("arrivee");
24            double poids = (Double) jChemin.get("poids");
25            graph.addEdge(depart.intValue(), arrivee.intValue(), poids.floatValue());
26        }
27    }
28
29    /**
30     * Makes a array of the graphs from the .JSON
31     * @return the array of graphs
32     */
33    private static Tester getTest() {
34        int[] startDijkstra = null;
35        JSONParser parser = new JSONParser();
36
37        try {
38            Object obj = parser.parse(new FileReader(testFile));
39            JSONObject jsonObject = ((JSONObject) obj);
40            JSONArray graphs = (JSONArray) jsonObject.get("graphs");
41
42            int size = graphs.size();
43            int i = 0;
44            testGraphs = new Graph[size];
45            startDijkstra = new int[size];
46
47            for (Object oGraph : graphs) {
48                JSONObject jGraph = (JSONObject) oGraph;
49                Long sommets = (Long) jGraph.get("sommets");
50                int[] parents = (int[]) jGraph.get("parents");
51                Graph graph = new Graph(sommets);
52                Long start = (Long) jGraph.get("departDijkstra");
53                int ISoriented = jGraph.get("estOriente");
54                Boolean isOriented = Boolean.valueOf(jGraph.get("chemins").get("isOriented"));
55                startDijkstra[i] = start;
56                addPath(graph, (JSONArray) jGraph.get("chemins"), isOriented);
57                startDijkstra[i] = start;
58                testGraphs[i] = graph;
59            }
60        } catch (Exception e) {
61            e.printStackTrace();
62        }
63        Tester t = new Tester(testGraphs, startDijkstra);
64        return t;
65    }
66
67    /**
68     * Read in the .JSON the expected results...
69     * @return the array of expected results
70     */
71    private static Result[] getResults() {
72        Result[] testResults = null;
73        JSONParser parser = new JSONParser();
74
75        try {
76            Object obj = parser.parse(new FileReader(resultFile));
77            JSONObject jsonObject = (JSONObject) obj;
78            JSONArray results = (JSONArray) jsonObject.get("results");
79            int size = results.size();
80            int i = 0;
81            testResults = new Result[size];
82
83            for (Object oResult : results) {
84                JSONObject jResult = (JSONObject) oResult;
85                JSONArray jDistances = (JSONArray) jResult.get("distances");
86                JSONArray jParents = (JSONArray) jResult.get("parents");
87                double[] distances = jDistances.stream().mapToDouble(o -> (Double)o).toArray();
88                long[] parents = Arrays.stream((Parents.toArray()), mapToLong(o -> (Long)o), toArray());
89                testResults[i++] = new Result(Converter.doubleToInt(distances), Converter.longToInt(parents));
90            }
91        } catch (Exception e) {
92            e.printStackTrace();
93        }
94        return testResults;
95    }
96
97    public static void main(String[] args) {
98        Tester tester = getTest();
99        Graph[] graphs = tester.graphs;
100    Result[] expectedResult = Test.getResults();
101    // Some verifications on the read data...
102
103    // Some verifications on the read data...
104    if(graphs.length == 0) {
105        System.err.println("Erreur il n'y aucun graph, ou la lecture n'a pas fonctionnée...");
106        return;
107    }
108    if(expectedResults.length == 0) {
109        System.err.println("Erreur il n'y aucun resultat, ou la lecture n'a pas fonctionnée...");
110        return;
111    }
112    if(graphs.length != expectedResult.length) {
113        System.err.println("Erreur il n'y pas autant de graph (" + graphs.length + ") que de resultat (" + expectedResult.length + ")");
114        return;
115    }
116
117    // Computes the result for all graphs and compare to the expected results
118    int i = 0;
119    int passed = 0;
120    for(Graph graph : graphs) {
121        Result result = TestGraphs.Dijkstra(graph, tester.startDijkstra[i]);
122        //System.out.println(result);
123        if(result.equals(expectedResults[i])) {
124            passed += 1;
125        }
126        i++;
127    }
128    System.out.println("Resultat : [" + passed + "/" + i + "]");
129
130}
```

GitHub - dijkstra/src/test/Test.java - Eclipse IDE

```
1 package test;
2
3 import java.io.FileReader;
4
5 public class Test {
6
7     public static class Tester {
8         private Graph[] graphs;
9         private int[] startDijkstra;
10    public Tester(Graph[] graphs, int[] startDijkstra) {
11        this.graphs = graphs;
12        this.startDijkstra = startDijkstra;
13    }
14}
15
16    private void addPath(Graph graph, JSONArray chemins, Boolean isOriented) {
17        for(Object oChemin : chemins) {
18            JSONObject jChemin = (JSONObject) oChemin;
19            Long depart = (Long) jChemin.get("depart");
20            Long arrivee = (Long) jChemin.get("arrivee");
21            double poids = (Double) jChemin.get("poids");
22            graph.addEdge(depart.intValue(), arrivee.intValue(), poids.floatValue());
23        }
24    }
25
26    /**
27     * Makes a array of the graphs from the .JSON
28     * @return the array of graphs
29     */
30    private static Tester getTest() {
31        int[] startDijkstra = null;
32        JSONParser parser = new JSONParser();
33
34        try {
35            Object obj = parser.parse(new FileReader(testFile));
36            JSONObject jsonObject = ((JSONObject) obj);
37            JSONArray graphs = (JSONArray) jsonObject.get("graphs");
38
39            int size = graphs.size();
40            int i = 0;
41            testGraphs = new Graph[size];
42            startDijkstra = new int[size];
43
44            for (Object oGraph : graphs) {
45                JSONObject jGraph = (JSONObject) oGraph;
46                Long sommets = (Long) jGraph.get("sommets");
47                int[] parents = (int[]) jGraph.get("parents");
48                Graph graph = new Graph(sommets);
49                Long start = (Long) jGraph.get("departDijkstra");
50                int ISoriented = jGraph.get("estOriente");
51                Boolean isOriented = Boolean.valueOf(jGraph.get("chemins").get("isOriented"));
52                startDijkstra[i] = start;
53                addPath(graph, (JSONArray) jGraph.get("chemins"), isOriented);
54                startDijkstra[i] = start;
55                testGraphs[i] = graph;
56            }
57        } catch (Exception e) {
58            e.printStackTrace();
59        }
60        Tester t = new Tester(testGraphs, startDijkstra);
61        return t;
62    }
63
64    /**
65     * Read in the .JSON the expected results...
66     * @return the array of expected results
67     */
68    private static Result[] getResults() {
69        Result[] testResults = null;
70        JSONParser parser = new JSONParser();
71
72        try {
73            Object obj = parser.parse(new FileReader(resultFile));
74            JSONObject jsonObject = (JSONObject) obj;
75            JSONArray results = (JSONArray) jsonObject.get("results");
76            int size = results.size();
77            int i = 0;
78            testResults = new Result[size];
79
80            for (Object oResult : results) {
81                JSONObject jResult = (JSONObject) oResult;
82                JSONArray jDistances = (JSONArray) jResult.get("distances");
83                JSONArray jParents = (JSONArray) jResult.get("parents");
84                double[] distances = jDistances.stream().mapToDouble(o -> (Double)o).toArray();
85                long[] parents = Arrays.stream((Parents.toArray()), mapToLong(o -> (Long)o), toArray());
86                testResults[i++] = new Result(Converter.doubleToInt(distances), Converter.longToInt(parents));
87            }
88        } catch (Exception e) {
89            e.printStackTrace();
90        }
91        return testResults;
92    }
93
94    public static void main(String[] args) {
95        Tester tester = getTest();
96        Graph[] graphs = tester.graphs;
97        Result[] expectedResult = Test.getResults();
98
99        // Some verifications on the read data...
100        if(graphs.length == 0) {
101            System.err.println("Erreur il n'y aucun graph, ou la lecture n'a pas fonctionnée...");
102            return;
103        }
104        if(expectedResults.length == 0) {
105            System.err.println("Erreur il n'y aucun resultat, ou la lecture n'a pas fonctionnée...");
106            return;
107        }
108        if(graphs.length != expectedResult.length) {
109            System.err.println("Erreur il n'y pas autant de graph (" + graphs.length + ") que de resultat (" + expectedResult.length + ")");
110            return;
111        }
112
113        // Computes the result for all graphs and compare to the expected results
114        int i = 0;
115        int passed = 0;
116        for(Graph graph : graphs) {
117            Result result = TestGraphs.Dijkstra(graph, tester.startDijkstra[i]);
118            //System.out.println(result);
119            if(result.equals(expectedResults[i])) {
120                passed += 1;
121            }
122            i++;
123        }
124        System.out.println("Resultat : [" + passed + "/" + i + "]");
125
126    }
127
128}
```

GitHub - dijkstra/src/test/Test.java - Eclipse IDE

```
1 package test;
2
3 import java.io.FileReader;
4
5 public class Test {
6
7     public static class Tester {
8         private Graph[] graphs;
9         private int[] startDijkstra;
10    public Tester(Graph[] graphs, int[] startDijkstra) {
11        this.graphs = graphs;
12        this.startDijkstra = startDijkstra;
13    }
14}
15
16    private void addPath(Graph graph, JSONArray chemins, Boolean isOriented) {
17        for(Object oChemin : chemins) {
18            JSONObject jChemin = (JSONObject) oChemin;
19            Long depart = (Long) jChemin.get("depart");
20            Long arrivee = (Long) jChemin.get("arrivee");
21            double poids = (Double) jChemin.get("poids");
22            graph.addEdge(depart.intValue(), arrivee.intValue(), poids.floatValue());
23        }
24    }
25
26    /**
27     * Makes a array of the graphs from the .JSON
28     * @return the array of graphs
29     */
30    private static Tester getTest() {
31        int[] startDijkstra = null;
32        JSONParser parser = new JSONParser();
33
34        try {
35            Object obj = parser.parse(new FileReader(testFile));
36            JSONObject jsonObject = ((JSONObject) obj);
37            JSONArray graphs = (JSONArray) jsonObject.get("graphs");
38
39            int size = graphs.size();
40            int i = 0;
41            testGraphs = new Graph[size];
42            startDijkstra = new int[size];
43
44            for (Object oGraph : graphs) {
45                JSONObject jGraph = (JSONObject) oGraph;
46                Long sommets = (Long) jGraph.get("sommets");
47                int[] parents = (int[]) jGraph.get("parents");
48                Graph graph = new Graph(sommets);
49                Long start = (Long) jGraph.get("departDijkstra");
50                int ISoriented = jGraph.get("estOriente");
51                Boolean isOriented = Boolean.valueOf(jGraph.get("chemins").get("isOriented"));
52                startDijkstra[i] = start;
53                addPath(graph, (JSONArray) jGraph.get("chemins"), isOriented);
54                startDijkstra[i] = start;
55                testGraphs[i] = graph;
56            }
57        } catch (Exception e) {
58            e.printStackTrace();
59        }
60        Tester t = new Tester(testGraphs, startDijkstra);
61        return t;
62    }
63
64    /**
65     * Read in the .JSON the expected results...
66     * @return the array of expected results
67     */
68    private static Result[] getResults() {
69        Result[] testResults = null;
70        JSONParser parser = new JSONParser();
71
72        try {
73            Object obj = parser.parse(new FileReader(resultFile));
74            JSONObject jsonObject = (JSONObject) obj;
75            JSONArray results = (JSONArray) jsonObject.get("results");
76            int size = results.size();
77            int i = 0;
78            testResults = new Result[size];
79
80            for (Object oResult : results) {
81                JSONObject jResult = (JSONObject) oResult;
82                JSONArray jDistances = (JSONArray) jResult.get("distances");
83                JSONArray jParents = (JSONArray) jResult.get("parents");
84                double[] distances = jDistances.stream().mapToDouble(o -> (Double)o).toArray();
85                long[] parents = Arrays.stream((Parents.toArray()), mapToLong(o -> (Long)o), toArray());
86                testResults[i++] = new Result(Converter.doubleToInt(distances), Converter.longToInt(parents));
87            }
88        } catch (Exception e) {
89            e.printStackTrace();
90        }
91        return testResults;
92    }
93
94    public static void main(String[] args) {
95        Tester tester = getTest();
96        Graph[] graphs = tester.graphs;
97        Result[] expectedResult = Test.getResults();
98
99        // Some verifications on the read data...
100        if(graphs.length == 0) {
101            System.err.println("Erreur il n'y aucun graph, ou la lecture n'a pas fonctionnée...");
102            return;
103        }
104        if(expectedResults.length == 0) {
105            System.err.println("Erreur il n'y aucun resultat, ou la lecture n'a pas fonctionnée...");
106            return;
107        }
108        if(graphs.length != expectedResult.length) {
109            System.err.println("Erreur il n'y pas autant de graph (" + graphs.length + ") que de resultat (" + expectedResult.length + ")");
110            return;
111        }
112
113        // Computes the result for all graphs and compare to the expected results
114        int i = 0;
115        int passed = 0;
116        for(Graph graph : graphs) {
117            Result result = TestGraphs.Dijkstra(graph, tester.startDijkstra[i]);
118            //System.out.println(result);
119            if(result.equals(expectedResults[i])) {
120                passed += 1;
121            }
122            i++;
123        }
124        System.out.println("Resultat : [" + passed + "/" + i + "]");
125
126    }
127
128}
```

```
145
146
147 }
148 }
149 }
```

```
GitHub - dijkstra/src/dijkstra/testGraphs.java - Eclipse IDE
Edge.java Graph.java Result.java Vertex.java Convor.java TestResult.java Test.java test.txt TestGraphs.java

1 package dijkstra;
2
3 import java.util.Iterator;
4
5 public class TestGraphs {
6
7     public static void main(String[] args) {
8         Graph g = new Graph();
9
10        System.out.println("Graph:");
11
12        // add Edges
13        g.addEdge(0, 1, 5.27);
14        g.addEdge(0, 2, 8.97);
15        g.addEdge(0, 3, 10.17);
16        g.addEdge(1, 3, 5.93);
17        g.addEdge(1, 4, 15.29);
18        g.addEdge(2, 3, 7.37);
19        g.addEdge(2, 4, 2.37);
20        g.addEdge(3, 4, 8.57);
21        g.addEdge(4, 2, 2.77);
22
23        // print Graph
24        g.printGraph();
25
26        // Dijkstras Shortest Path Algorithm
27        System.out.println("Dijkstra Shortest Path:");
28        Dijkstra(g, 0);
29    }
30
31    public static Result Dijkstra(Graph g, int startVertex) {
32        // for storing distances after removing vertex from Queue
33        float[] distances = new float[g.getCount()];
34        // for storing parents after removing vertex from Queue
35        int[] parents = new int[g.getCount()];
36        for (int i = 0; i < g.getCount(); i++) {
37            parents[i] = -1;
38        }
39
40        // set up vertex queue
41        PriorityQueue<Vertex> Q = new PriorityQueue<Vertex>();
42        for (int i = 0; i < g.getCount(); i++) {
43            if (i != startVertex) {
44                Q.add(new Vertex(i));
45            }
46        }
47
48        // add startVertex
49        Vertex node = new Vertex(startVertex);
50        node.setDistance(0);
51        Q.add(node);
52
53        // loop through all vertices
54        while (!Q.isEmpty()) {
55            // get vertex with shortest distance
56            Vertex u = Q.remove();
57
58            // iterate through all neighbours
59            Iterator<Edge> it = u.neighbours(u.getId()).iterator();
60            while (it.hasNext()) {
61                Edge e = it.next();
62                Iterator<Vertex> it2 = e.iterator();
63                while (it2.hasNext()) {
64                    Vertex v = it2.next();
65                    // check if vertex was visited already
66                    if (e.getEndPoint() != v.getId()) {
67                        continue;
68                    }
69                    // check distance
70                    if (v.getDistance() > u.getDistance() + e.getWeight()) {
71                        v.setDistance(u.getDistance() + e.getWeight());
72                        v.setParent(v.getId());
73                        parents[v.getId()] = v.getParent().getId();
74                    }
75                }
76            }
77        }
78    }
79
80    // print final shortest paths
81    /*
82     * System.out.println("Vertex\tDistance\tParent Vertex");
83     * for (int i = 0; i < g.getCount(); i++) {
84     *     System.out.println(i + "\t" + distances[i] + "\t" + parents[i]);
85     * }
86     */
87
88    return new Result(distances, parents);
89
90 }
91 }
```

```
GitHub - dijkstra/src/dijkstra/TestGraphs.java - Eclipse IDE
Edge.java Graph.java Result.java Vertex.java Convor.java TestResult.java Test.java test.txt TestGraphs.java

3 parents[i] = -1;
38
39
40    // set up vertex queue
41    PriorityQueue<Vertex> Q = new PriorityQueue<Vertex>();
42    for (int i = 0; i < g.getCount(); i++) {
43        if (i != startVertex) {
44            Q.add(new Vertex(i));
45        }
46    }
47
48    // add startVertex
49    Vertex node = new Vertex(startVertex);
50    node.setDistance(0);
51    Q.add(node);
52
53    // loop through all vertices
54    while (!Q.isEmpty()) {
55        // get vertex with shortest distance
56        Vertex u = Q.remove();
57        distances[u.getId()] = u.getDistance();
58
59        // iterate through all neighbours
60        Iterator<Edge> it = u.neighbours(u.getId()).iterator();
61        while (it.hasNext()) {
62            Edge e = it.next();
63            Iterator<Vertex> it2 = e.iterator();
64            while (it2.hasNext()) {
65                Vertex v = it2.next();
66                // check if vertex was visited already
67                if (e.getEndPoint() != v.getId()) {
68                    continue;
69                }
70                // check distance
71                if (v.getDistance() > u.getDistance() + e.getWeight()) {
72                    v.setDistance(u.getDistance() + e.getWeight());
73                    v.setParent(v.getId());
74                    parents[v.getId()] = v.getParent().getId();
75                }
76            }
77        }
78    }
79
80    // print final shortest paths
81    /*
82     * System.out.println("Vertex\tDistance\tParent Vertex");
83     * for (int i = 0; i < g.getCount(); i++) {
84     *     System.out.println(i + "\t" + distances[i] + "\t" + parents[i]);
85     * }
86     */
87
88    return new Result(distances, parents);
89
90 }
91 }
```

GitHub - dijkstra/src/dijkstra/Vertex.java - Eclipse IDE

```
1 package dijkstra;
2
3 public class Vertex implements Comparable<Vertex>{
4     private int id;
5     private float distance;
6     private Vertex parent;
7
8     public Vertex(){
9         distance = Float.MAX_VALUE; // "infinity"
10        parent = null;
11    }
12
13    public Vertex(int id){
14        this.id = id;
15        distance = Float.MAX_VALUE; // "infinity"
16        parent = null;
17    }
18
19    public int getId() {
20        return id;
21    }
22
23    public void setId(int id) {
24        this.id = id;
25    }
26
27    public float getDistance() {
28        return distance;
29    }
30
31    public void setDistance(float distance) {
32        this.distance = distance;
33    }
34
35    public Vertex getParent() {
36        return parent;
37    }
38
39    public void setParent(Vertex parent){
40        this.parent = parent;
41    }
42
43    public int compareTo(Vertex other) {
44        return Double.compare(this.distance, other.distance);
45    }
46 }
```

GitHub - dijkstra/src/dijkstra/Edge.java - Eclipse IDE

```
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 getStartPoint() {
15        return startPoint;
16    }
17
18    public int getEndPoint() {
19        return endPoint;
20    }
21
22    public float getWeight() {
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 }
```

GitHub - dijkstra/src/dijkstra/Graph.java - Eclipse IDE

```
1 package dijkstra;
2
3 import java.util.Iterator;
4
5 public class Graph {
6     private int vCount;
7     private PriorityQueue<Edge>[] adj;
8
9     public int getvCount() {
10         return vCount;
11     }
12
13     @SuppressWarnings("unchecked")
14     public Graph(int vCount) {
15         this.vCount = vCount;
16         adj = new PriorityQueue[vCount];
17         for (int i = 0; i < vCount; i++) {
18             adj[i] = new PriorityQueue<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(Edge e) {
27         adj[e.getStartPoint()].remove(e);
28     }
29
30     public void removeEdge(int i, int j) {
31         Iterator<Edge> it = adj[i].iterator();
32         Edge other = new Edge(i, j, 0);
33         while (it.hasNext()) {
34             if (it.next().equals(other)) {
35                 it.remove();
36                 return;
37             }
38         }
39     }
40
41     public boolean hasEdge(Edge e) {
42         return adj[e.getStartPoint()].contains(e);
43     }
44
45     public PriorityQueue<Edge> neighbours(int vertex) {
46         return adj[vertex];
47     }
48
49     public void print() {
50         for (int i = 0; i < vCount; i++) {
51             PriorityQueue<Edge> edges = neighbours(i);
52             Iterator<Edge> it = edges.iterator();
53             System.out.print(i + ": ");
54             for (int j = 0; j < edges.size(); j++) {
55                 System.out.print(it.next() + " ");
56             }
57             System.out.println();
58         }
59     }
60
61     public String toString() {
62         StringBuilder builder = new StringBuilder();
63         for (int i = 0; i < vCount; i++) {
64             PriorityQueue<Edge> edges = neighbours(i);
65             Iterator<Edge> it = edges.iterator();
66             builder.append(i + ": ");
67             for (int j = 0; j < edges.size(); j++) {
68                 builder.append(it.next() + " ");
69             }
70             builder.append("\n");
71         }
72         return builder.toString();
73     }
74
75 }
```

GitHub - dijkstra/src/dijkstra/Graph.java - Eclipse IDE

```
1 package dijkstra;
2
3 import java.util.Iterator;
4
5 public class Graph {
6     private int vCount;
7     private PriorityQueue<Edge>[] adj;
8
9     public int getvCount() {
10         return vCount;
11     }
12
13     @SuppressWarnings("unchecked")
14     public Graph(int vCount) {
15         this.vCount = vCount;
16         adj = new PriorityQueue[vCount];
17         for (int i = 0; i < vCount; i++) {
18             adj[i] = new PriorityQueue<Edge>();
19         }
20     }
21
22     public void addEdge(Edge e) {
23         adj[e.getStartPoint()].add(e);
24     }
25
26     public void removeEdge(Edge e) {
27         adj[e.getStartPoint()].remove(e);
28     }
29
30     public void removeEdge(int i, int j) {
31         Iterator<Edge> it = adj[i].iterator();
32         Edge other = new Edge(i, j, 0);
33         while (it.hasNext()) {
34             if (it.next().equals(other)) {
35                 it.remove();
36                 return;
37             }
38         }
39     }
40
41     public boolean hasEdge(Edge e) {
42         return adj[e.getStartPoint()].contains(e);
43     }
44
45     public PriorityQueue<Edge> neighbours(int vertex) {
46         return adj[vertex];
47     }
48
49     public void print() {
50         for (int i = 0; i < vCount; i++) {
51             PriorityQueue<Edge> edges = neighbours(i);
52             Iterator<Edge> it = edges.iterator();
53             System.out.print(i + ": ");
54             for (int j = 0; j < edges.size(); j++) {
55                 System.out.print(it.next() + " ");
56             }
57             System.out.println();
58         }
59     }
60
61     public String toString() {
62         StringBuilder builder = new StringBuilder();
63         for (int i = 0; i < vCount; i++) {
64             PriorityQueue<Edge> edges = neighbours(i);
65             Iterator<Edge> it = edges.iterator();
66             builder.append(i + ": ");
67             for (int j = 0; j < edges.size(); j++) {
68                 builder.append(it.next() + " ");
69             }
70             builder.append("\n");
71         }
72         return builder.toString();
73     }
74
75 }
```

```
1 package dijkstra;
2
3 import utilities.TestResult;
4
5 public class Result {
6     private float[] distances;
7     private int[] parents;
8
9     public Result(float[] distances, int[] parents) {
10        super();
11        this.distances = distances;
12        this.parents = parents;
13    }
14
15    public float[] getDistances() {
16        return distances;
17    }
18
19    public int[]getParents() {
20        return parents;
21    }
22
23    public String toString() {
24        StringBuilder builder = new StringBuilder("Vertex\tDistance\tParent\tVertex\n");
25        for (int i = 0; i < parents.length; i++) {
26            builder.append(i + "\t" + distances[i] + "\t" + parents[i] + "\n");
27        }
28        return builder.toString();
29    }
30
31    public TestResult equals(Result r) {
32        // Compare the arrays sizes
33        if(r.distances.length!=distances.length) {
34            return new TestResult("Difference taille des tableaux distances");
35        }
36        if(r.parents.length!=parents.length) {
37            return new TestResult("Difference taille des tableaux parents");
38        }
39        if(r.parents.length!=distances.length) {
40            return new TestResult("Difference entre les tailles des tableaux distances et parents");
41        }
42        // Compare the arrays :
43        for(int i=0;i<r.distances.length;i++) {
44            if(r.distances[i]!=distances[i]) {
45                return new TestResult("Difference à l'indice : "+i+" result.distances["+i+"] et expectedResult.distances["+i+"]");
46            }
47            if(r.parents[i]!=parents[i]) {
48                return new TestResult("Difference à l'indice : "+i+" result.parents["+i+"] et expectedResult.parents["+i+"]");
49            }
50        }
51        return new TestResult();
52    }
53
54
55 }
```

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

## 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 qq 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

### Préambule

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

### Préambule

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

## Quatrième mutant :

```
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.