

TP Test

Lien du projet sur GitHub : <https://github.com/Sakayaku/ILU4-Test>

Question 0 – Spécification :

L'algorithme de Bellman-Ford calcule les plus court chemins depuis un sommet source dans un graphe orienté.

Contrairement à l'algorithme de Dijkstra, cet algorithme autorise la présence d'arcs à poids négatif.

Il permet également de détecter la présence d'un circuit absorbant, c'est à dire de poids total négatif, et donc dans lesquels il n'existe pas forcément de plus court chemin entre deux sommets.

Les entrées du programme sont le nombre de sommet, le nombre d'arcs, le sommet de départ et le poids de chaque arc (le graphe).

La sortie sont les distances minimales entre chaque sommet et la source. En cas de cycle de poids négatif détecté on renvoie une distance de zéro et on affiche un message indiquant qu'un circuit de poids négatif a été trouvé.

Le programme « Test » lit dans un fichier texte des jeux de tests et leurs oracles. Tout ce qui est lu dans le fichier de test est lu en tant que chaîne de caractères et tout est convertit ensuite en Integer lorsque cela est nécessaire.

Question 2 – Instructions graphe particulier :

Les instructions du programme BellmanFord permettant de trouver le plus court chemin sont les suivantes :

```
Test - ILU4-Test/src/main/BellmanFord.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

package main;
// Java Program to implement[]
import java.util.Arrays;

// Bellman For Algorithm
public class BellmanFord {
    // Graph is Created Using Edge Class
    public static class Edge {
        public int source;
        public int destination;
        public int weight;

        Edge() {
            source = destination = weight = 0;
        }

        int V, E;
        public Edge edge[];

        // Constructor to initialize the graph
        public BellmanFord(int v, int e) {
            V = v;
            E = e;
            edge = new Edge[e];
            for (int i = 0; i < e; ++i)
                edge[i] = new Edge();
        }

        // Bellman-Ford Algorithm to find shortest paths from source to all vertices
        public int[] BellmanFordAlgo(BellmanFord graph, int source) {
            int V = graph.V, E = graph.E;
            int dist[] = new int[V];
            int circuitN[] = new int [1];

            // Step 1: Initialize distances from source to all other vertices as INFINITE
            Arrays.fill(dist, Integer.MAX_VALUE);
            dist[source] = 0;

            // Step 2: Relax all edges |V| - 1 times.
            for (int i = 1; i < V; ++i) {
                for (int j = 0; j < E; ++j) {
                    int u = graph.edge[j].source;
                    int v = graph.edge[j].destination;
                    int weight = graph.edge[j].weight;
                    if (dist[u] != Integer.MAX_VALUE && dist[u] + weight < dist[v])
                        dist[v] = dist[u] + weight;
                }
            }

            // Step 3: Check for negative-weight cycles
            for (int j = 0; j < E; ++j) {
                int u = graph.edge[j].source;
                int v = graph.edge[j].destination;
                int weight = graph.edge[j].weight;
                if (dist[u] != Integer.MAX_VALUE && dist[u] + weight < dist[v]) {
                    System.out.println(Couleur.VIOLET+"Le graphe contient un circuit de poids négatif!" +Couleur.RESET);
                    return circuitN;
                }
            }

            // Print distances from source to all vertices
            printDistances(dist, V);
            return dist;
        }

        // Print distances from source to all vertices
        static void printDistances(int dist[], int V) {
            System.out.println("Vertex Distance from Source:");
            for (int i = 0; i < V && V<100; ++i) //pour ne pas afficher 1 028 653 052 valeurs avec les tests aux limites
                System.out.println(i + "\t\t" + dist[i]);
        }

        // Main method to test the Bellman-Ford algorithm
        public static void main(String[] args) {
            int V = 5;
            int E = 8;
            BellmanFord graph = new BellmanFord(V, E);

            // Define edges
            // Edge 0-1
            graph.edge[0].source = 0;
            graph.edge[0].destination = 1;
            graph.edge[0].weight = -1;

            // Edge 0-2
            graph.edge[1].source = 0;
            graph.edge[1].destination = 2;
            graph.edge[1].weight = 4;

            // Edge 1-2
            graph.edge[2].source = 1;
            graph.edge[2].destination = 2;
            graph.edge[2].weight = 3;

            // Edge 1-3
            graph.edge[3].source = 1;
            graph.edge[3].destination = 3;
            graph.edge[3].weight = 2;

            // Edge 1-4
            graph.edge[4].source = 1;
            graph.edge[4].destination = 4;
            graph.edge[4].weight = 4;
        }
    }
}
```

The screenshot shows the Eclipse IDE with a Java project. The main editor displays the `BellmanFord.java` file, which defines a graph and executes the Bellman-Ford algorithm. The code includes comments for each edge and the algorithm execution. The console window at the bottom shows the test results, including the graph structure and the output of the Bellman-Ford algorithm.

```
Test - ILU4-Test/src/main/BellmanFord.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

JeuxDeTestEtOracle.txt BellmanFord.java x

102 graph.edge[3].destination = 3;
103 graph.edge[3].weight = 2;
104
105 // Edge 1-4
106 graph.edge[4].source = 1;
107 graph.edge[4].destination = 4;
108 graph.edge[4].weight = 2;
109
110 // Edge 3-2
111 graph.edge[5].source = 3;
112 graph.edge[5].destination = 2;
113 graph.edge[5].weight = 5;
114
115 // Edge 3-1
116 graph.edge[6].source = 3;
117 graph.edge[6].destination = 1;
118 graph.edge[6].weight = 1;
119
120 // Edge 4-3
121 graph.edge[7].source = 4;
122 graph.edge[7].destination = 3;
123 graph.edge[7].weight = -3;
124
125 // Execute Bellman-Ford algorithm
126 graph.BellmanFordAlgo(graph, 0);
127 }
128 }
129 }

Problems Javadoc Declaration Console x Coverage
<terminated> Test (1) [Java Application] C:\Users\Solène\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_23.0.2.v20250131-0604\jre\bin\javaw.exe (28 avr. 2025, 19:55:55 - 19:55:56 elapsed: 0:00:01.195) [pid: 5064]

----- Résultat Instructions pour graphe particulier -----

[6, 9, 0, 0, 1, 6, 0, 2, 4, 0, 3, 5, 1, 4, -1, 2, 1, -2, 2, 4, 3, 3, 2, -2, 3, 5, -1, 4, 5, 3]
Nombre de sommet : 6
Nombre d'arcs : 9
Sommet de départ : 0
Graphe : [0,1,6, 0,2,4, 0,3,5, 1,4,-1, 2,1,-2, 2,4,3, 3,2,-2, 3,5,-1, 4,5,3]
Vertex Distance from Source:
0      0
1      1
2      3
3      5
4      0
5      3
Oracle : [0, 1, 3, 5, 0, 3] --> Pass

Writable Smart Insert 74 : 87 : 2402
```

Question 3 - Couverture totale des instructions :

```

49     int weight = graph.edge[j].weight;
50     if (dist[u] != Integer.MAX_VALUE && dist[u] + weight < dist[v])
51         dist[v] = dist[u] + weight;
52     }
53 }
54
55 // Step 3: Check for negative-weight cycles
56 for (int j = 0; j < E; ++j) {
57     int u = graph.edge[j].source;
58     int v = graph.edge[j].destination;
59     int weight = graph.edge[j].weight;
60     if (dist[u] != Integer.MAX_VALUE && dist[u] + weight < dist[v]) {
61         System.out.println(Couleur.VIOLET+"Le graphe contient un circuit de poids négatif!"+Couleur.RESET);
62         return circuitN;
63     }
64 }
65
66 // Print distances from source to all vertices
67 printDistances(dist, V);
68 return dist;
69 }
70
71 // Print distances from source to all vertices
72 static void printDistances(int dist[], int V) {
73     System.out.println("Vertex Distance from Source:");
74     for (int i = 0; i < V && V<100; ++i) //pour ne pas afficher 1 028 653 052 valeurs avec les tests aux limites
75         System.out.println(i + "\t\t" + dist[i]);
76 }
77

```

Element	Coverage	Covered Instru...	Missed Instru...	Total Instructio...
Test (1) (28 avr. 2025 20:02:52)				
ILU4-Test	44,7 %	599	742	1341
src	44,7 %	599	742	1341
main	34,3 %	271	518	789
BellmanFord_mutants.java	0,0 %	0	358	358
BellmanFord.java	55,3 %	198	160	358
BellmanFord	53,8 %	186	160	346
main(String[])	0,0 %	0	160	160
Edge	100,0 %	12	0	12
printDistances(int[], int)	100,0 %	28	0	28
BellmanFord(int, int)	100,0 %	27	0	27
BellmanFordAlgo(Bell...	100,0 %	131	0	131
Couleur.java	100,0 %	73	0	73
test	59,4 %	328	224	552

Comme mis en évidence dans cette capture d'écran, la suite de test donnée à l'occasion de la question 1 couvre déjà le maximum d'instructions du programme BellmanFord. On ne peut pas trouver une suite de tests qui couvre toutes les instructions à proprement parler car le main ne sera jamais exécuté par les tests.

Question 4 – Tests par la méthode des partitions et des catégories :

Comme expliqué précédemment, le programme « Test » lit dans un fichier texte des jeux de tests et leurs oracles. Tout ce qui est lu dans le fichier de test est lu en tant que chaîne de caractères et tout est convertit ensuite en Integer lorsque cela est nécessaire à l'aide de la méthode Integer.parseInt(). Cela signifie qu'il faut gérer les exceptions où la chaîne de caractère lue ne correspond pas à int. En utilisant la méthode des partitions sur le code « BellmanFord », les classes d'équivalence suivantes sont obtenues:

Exigence	Classes Valides	Classe Invalides
Nombres d'entrées	4 entrées : -Nombre de sommet -Nombre d'arc -Sommet de départ -Graphe	-Toutes les classes qui ont plus d'entrées -Toutes les classes qui ont moins d'entrées

Type d'entrée	-Integer -Integer -Integer -BelmanFord	-Non Integer -Non Integer -Non Integer -Non BelmanFord
Valeurs valides	- >0 - >0 - >=0 et <nombre de sommet - N'importe quel graphe qui respecte le nombre d'arcs, ne contenant que les nombres < nombre de sommet et >=0, qui contient le sommet de départ et qui réunit tous les sommets	- <=0 - <=0 - <0 ou >=nombre de sommet - N'importe quel graphe qui ne respecte pas le nombre d'arcs, contenant des nombres >= nombre de sommet et/ou <0, ne contenant pas le sommet de départ ou ne réunissant pas tous les sommets
Jeux de tests	<p>1. Nombre de sommet : 5 Nombre d'arcs: 8 Sommet de départ : 0 Graphe : 0,1,- 1_0,2,4_1,2,3_1,3,2_1,4,2_3, 2,5_3,1,1_4,3,-3</p> <p>2. Nombre de sommet : 8 Nombre d'arcs: 8 Sommet de départ : 0 Graphe : 0,1,2_2,3,6_4,3,8_5,7,2_6,4, 2_3,5,9_0,2,1_7,6,1</p> <p>3. Nombre de sommet : 3 Nombre d'arcs: 2 Sommet de départ : 1 Graphe : 1,0,2_0,2,3</p>	<p>1. <i>Nombre de sommet : Non indiqué</i> Nombre d'arcs : 8 Sommet de départ : 0 Graphe : 0,1,- 1_0,2,4_1,2,3_1,3,2_ 1,4,2_3,2,5_3,1,1_4, 3,-3</p> <p>2. Nombre de sommet : 5 <i>Nombre d'arcs : Non indiqué</i> Sommet de départ : 0 Graphe : 0,1,- 1_0,2,4_1,2,3_1,3,2_ 1,4,2_3,2,5_3,1,1_4, 3,-3</p> <p>3. Nombre de sommet : 5 Nombre d'arcs : 8 <i>Sommet de départ : Non indiqué</i> Graphe : 0,1,- 1_0,2,4_1,2,3_1,3,2_</p>

		<p>1,4,2_3,2,5_3,1,1_4, 3,-3</p> <p>4. Nombre de sommet : 5 Nombre d'arcs : 8 Sommet de départ : 0 Graphe : <i>Non indiqué</i></p> <p>5. Nombre de sommet : 5 Nombre d'arcs : 8 Sommet de départ : 0 <i>Entrée rajoutée : 7</i> Graphe : 0,1,- 1_0,2,4_1,2,3_1,3,2_ 1,4,2_3,2,5_3,1,1_4, 3,-3</p> <p>6. Nombre de sommet : 5 Nombre d'arcs : 8 Sommet de départ : 0 <i>Entrée rajoutée :</i> <i>0,1,-</i> <i>1_0,2,4_1,2,3_1,3,2_ 1,4,2_3,2,5_3,1,1_4, 3,-3</i> Graphe : 0,1,- 1_0,2,4_1,2,3_1,3,2_ 1,4,2_3,2,5_3,1,1_4, 3,-3</p> <p>7. Nombre de sommet : 5 Nombre d'arcs : 8 Sommet de départ : 0 <i>Entrée rajoutée :</i> <i>0,1,-</i></p>
--	--	---

		<p> $1_0,2,4_1,2,3_1,3,2_1,4,2_3,2,5_3,1,1_4,3,-3$ <i>Entrée rajoutée :</i> $0,1,-$ $1_0,2,4_1,2,3_1,3,2_1,4,2_3,2,5_3,1,1_4,3,-3$ Graphe : $0,1,-$ $1_0,2,4_1,2,3_1,3,2_1,4,2_3,2,5_3,1,1_4,3,-3$ </p> <p> 8. <i>Nombre de sommet :</i> <i>« a » (lettre)</i> Nombre d'arcs : 8 Sommet de départ : 0 Graphe : $0,1,-$ $1_0,2,4_1,2,3_1,3,2_1,4,2_3,2,5_3,1,1_4,3,-3$ </p> <p> 9. <i>Nombre de sommet:</i> 5 <i>Nombre d'arcs :</i> <i>« X » (lettre)</i> Sommet de départ : 0 Graphe : $0,1,-$ $1_0,2,4_1,2,3_1,3,2_1,4,2_3,2,5_3,1,1_4,3,-3$ </p> <p> 10. <i>Nombre de sommet:</i> 5 Nombre d'arcs : 8 <i>Sommet de départ :</i> <i>« s » (lettre)</i> Graphe : $0,1,-$ $1_0,2,4_1,2,3_1,3,2_1,4,2_3,2,5_3,1,1_4,3,-3$ </p>
--	--	--

		<p>11.Nombre de sommet: 5 Nombre d'arcs : 8 <i>Sommet de départ : 5 (égal au nombre de sommet)</i> Graphe : 0,1,- 1_0,2,4_1,2,3_1,3,2_ 1,4,2_3,2,5_3,1,1_4, 3,-3</p> <p>12.Nombre de sommet: 5 Nombre d'arcs : 8 <i>Sommet de départ : 6 (supérieur au nombre de sommet)</i> Graphe : 0,1,- 1_0,2,4_1,2,3_1,3,2_ 1,4,2_3,2,5_3,1,1_4, 3,-3</p> <p>13.Nombre de sommet: -1 (inférieur à 0) Nombre d'arcs : 8 Sommet de départ : 0 Graphe : 0,1,- 1_0,2,4_1,2,3_1,3,2_ 1,4,2_3,2,5_3,1,1_4, 3,-3</p> <p>14.Nombre de sommet: 5 <i>Nombre d'arcs : -1 (inférieur à 0)</i> Sommet de départ : 0 Graphe : 0,1,- 1_0,2,4_1,2,3_1,3,2_ 1,4,2_3,2,5_3,1,1_4, 3,-3</p> <p>15.Nombre de sommet:</p>
--	--	---

		<p>5</p> <p>Nombre d'arcs : 8</p> <p>Sommet de départ : -1 (inférieur à 0)</p> <p>Graphe : 0,1,- 1_0,2,4_1,2,3_1,3,2_ 1,4,2_3,2,5_3,1,1_4, 3,-3</p> <p>16.Nombre de sommet: 5</p> <p>Nombre d'arcs : 8</p> <p>Sommet de départ : 0</p> <p>Graphe : 0,1,- 1_0,2,4_1,2,3_1,3,2_ 1,4,2_3,2,5_3,1,1 (Ne respecte pas le nombre d'arcs)</p> <p>17.Nombre de sommet: 5</p> <p>Nombre d'arcs : 8</p> <p>Sommet de départ : 0</p> <p>Graphe : 0,1,- 1_0,2,4_1,2,3_1,3,2_ 1,4,2_3,2,5_3,1,1_5, 3,-3_4,1,2 (Contient un arc de trop)</p> <p>18.Nombre de sommet: 5</p> <p>Nombre d'arcs : 8</p> <p>Sommet de départ : 0</p> <p>Graphe : 0,1,- 1_0,2,4_1,2,3_1,3,2_ 1,4,2_3,2,5_3,1,1_5, 3,-3 (Contient un sommet qui n'existe pas)</p>
--	--	--

Question 5 – Tests par la méthode des limites :

Comme nous utilisons des List pour stocker les nombres et les lire nous ne pouvons pas tester comme cas limite les maximum et minimum que peuvent prendre comme valeur un entier car la valeur limite que peut supporter une liste est plus petite que cela. Après quelques tests, la limite pour ma machine est 1 028 653 052 contre 2 147 483 647 en théorie, ainsi quand j'écris MaxInt dans le tableau ci-dessous c'est à ce nombre que je réfère. J'ajoute que les tests aux limites ont cet inconvénient de travailler avec des très grand ou très petits nombres, et comme mon programme demande en entrée un oracle de la même taille que le nombre de sommet et un graphe de la même taille que le nombre d'arcs, pour s'épargner des ressources en cas d'erreur de saisie et vérifier la cohérence, les tests seront marqués comme fail puisqu'ils lanceront inévitablement une exception de ma part. Il est théoriquement possible de travailler avec ce genre de nombre, mais il m'est décemment impossible d'insérer un oracle de plus de 1 028 653 052 caractères avec mon matériel (j'ai essayé). On remarque tout de même que le traitement avec 1 028 653 052 arcs et sommets peut se faire et se fait puisqu'on discerne un ralentissement lors du lancement des tests à ce moment là, c'est seulement par nécessité que les exceptions sont lancées.

Exigence	Classe valides	Classes invalides
Nombre de sommet entier > 0	1 2 MaxInt MaxInt-1	0 MaxInt+1
Nombre d'arcs entier > 0	1 2 MaxInt MaxInt-1	0 MaxInt+1
Sommet de départ entier >=0 et <nombre de sommet (intervalle de valeur [0, nombre de sommet -1])	0 1 nombre de sommet -1 nombre de sommet -2	-1 nombre de sommet
Jeux de tests	1. Nombre de sommet : 1 Nombre d'arcs: 1 Sommet de départ : 0 Graphe : 0,0,0 2. Nombre de sommet : 2	1. Nombre de sommet : 0 Nombre d'arcs: 1 Sommet de départ : 0 Graphe : 0,0,0 2. Nombre de sommet : 1028653053

	<p>Nombre d'arcs: 2 Sommet de départ : 0 Graphe : 0,1,4</p>	<p>Nombre d'arcs: 1 Sommet de départ : 0 Graphe : 0,1,4</p>
	<p>3. Nombre de sommet : 1 028 653 052 Nombre d'arcs: 1 Sommet de départ : 0 Graphe : 0,1,4</p>	<p>3. Nombre de sommet : 8 Nombre d'arcs: 0 Sommet de départ : 0 Graphe : 0,0,0</p>
	<p>4. Nombre de sommet : 1 027 999 998 Nombre d'arcs: 1 Sommet de départ : 0 Graphe : 0,1,4</p>	<p>4. Nombre de sommet : 8 Nombre d'arcs: 1028653053 Sommet de départ : 0 Graphe : 0,0,0</p>
	<p>5. Nombre de sommet : 1 Nombre d'arcs: 1 Sommet de départ : 0 Graphe : 0,0,4</p>	<p>5. Nombre de sommet : 8 Nombre d'arcs: 8 Sommet de départ : -1 Graphe :</p>
	<p>6. Nombre de sommet : 3 Nombre d'arcs: 2 Sommet de départ : 0 Graphe : 0,1,4_1,2,4</p>	<p>0,1,2_2,3,6_4,3,8_5, 7,2_6,4,2_3,5,9_0,2, 1_7,6,1</p>
	<p>7. Nombre de sommet : 4 Nombre d'arcs: 1 028 653 052 Sommet de départ : 0 Graphe : 0,1,4_1,2,4</p>	<p>6. Nombre de sommet : 8 Nombre d'arcs: 8 Sommet de départ : 8 Graphe :</p>
	<p>8. Nombre de sommet : 4 Nombre d'arcs: 1</p>	<p>0,1,2_2,3,6_4,3,8_5, 7,2_6,4,2_3,5,9_0,2, 1_7,6,1</p>

	027 999 998 Sommet de départ : 0 Graphe : 0,1,4_1,2,4 9. Nombre de sommet : 4 Nombre d'arcs: 3 Sommet de départ : 0 Graphe : 0,1,4_1,2,4_2,3,-3 10. Nombre de sommet : 4 Nombre d'arcs: 3 Sommet de départ : 1 Graphe : 1,0,4_1,2,4_2,3,-3 11. Nombre de sommet : 4 Nombre d'arcs: 3 Sommet de départ : 3 Graphe : 3,1,4_1,2,4_2,0,-3 12. Nombre de sommet : 4 Nombre d'arcs: 3 Sommet de départ : 2 Graphe : 2,1,4_2,3,5_3,0,-3	
--	---	--

Question 6 - Tests de mutation :

Mutants	Mutation	Ligne	Jeu de test tuant	Jeu de test survivant	Commentaire
M1 : Modifier	< → >	30	Nombre de sommet : 5	Aucun	NullPointerException

un opérateur relationnel			<p>Nombre d'arcs : 8</p> <p>Sommet de départ : 0</p> <p>Graphe : 0,1,-1_0,2,4_1,2,3_1,3,2_1,4,2_3,2,5_3,1,1_4,3,-3</p>		on
		45	<p>Nombre de sommet : 4</p> <p>Nombre d'arcs : 6</p> <p>Sommet de départ : 0</p> <p>Graphe : 0,1,-3_0,3,4_1,2,5_1,3,2_2,0,6_3,2,-1</p>	<p>Nombre de sommet : 8</p> <p>Nombre d'arcs: 8</p> <p>Sommet de départ : 8</p> <p>Graphe : 0,1,2_2,3,6_4,3,8_5,7,2_6,4,2_3,5,9_0,2,1_7,6,1</p>	Chaque test passant en temps normal est noté comme ayant un cycle de poids négatif
		46	Idem	Idem	Les lignes 45 et 46 étant des boucles imbriquées, changer un des deux opérateur revient au même effet que le mutant précédent : on n'entre pas dans la boucle
		60	<p>Nombre de sommet : 5</p> <p>Nombre d'arcs : 7</p> <p>Sommet de départ : 0</p> <p>Graphe : 0,1,-3_0,3,4_1,2,5_1,3,2</p>	<p>Nombre de sommet : 5</p> <p>Nombre d'arcs : 7</p> <p>Sommet de départ : 0</p> <p>Graphe : 0,1,-3_0,3,4_1,2,5_1,3,</p>	Même commentaire que pour la ligne 45

			_2,0,6_3,2,-1_0,4,-2 Avec le bon oracle	2_2,0,6_3,2,-1_0,4,-2 Avec le mauvais oracle	
M2 : Modifier une variable	$E \rightarrow V$	28	Nombre de sommet : 5 Nombre d'arcs : 7 Sommet de départ : 0 Graphe : 0,1,-3_0,3,4_1,2,5_1,3,2_2,0,6_3,2,-1_0,4,-2	Aucun	NullPointerException
		46	Nombre de sommet : 5 Nombre d'arcs : 8 Sommet de départ : 0 Graphe : 0,1,-1_0,2,4_1,2,3_1,3,2_1,4,2_3,2,5_3,1,1_4,3,-3	Nombre de sommet : 8 Nombre d'arcs: 8 Sommet de départ : 0 Graphe : 0,1,2_2,3,6_4,3,8_5,7,2_6,4,2_3,5,9_0,2,1_7,6,1	Comme précédemment, le fait que l'on modifie la deuxième valeur de la boucle imbriquée indique qu'un cycle de poids négatif est détecté.
		56	Idem	Idem	Idem
M3 : Modifier un opérateur arithmétique	$++i \rightarrow --i$	30	Tous les tests ne lançant pas une exception en temps normal	Nombre de sommet : 5 Nombre d'arcs : Non indiqué Sommet de départ : 0 Graphe : 0,1,-1_0,2,4_1,2,3_1,3,2_1,4,2_3,2,5_3,1,1_4,3,-3	La boucle de la ligne 30 commençant à 0, décrémenter au lieu d'incrémenter lancera fatalement l'exception correspondante.

		45	<p>Nombre de sommet : 3</p> <p>Nombre d'arcs : 3</p> <p>Sommet de départ : 0</p> <p>Graphe : 0,2,- 1_1,0,-3_2,1,-2</p>	<p>Nombre de sommet : 5</p> <p>Nombre d'arcs : 8</p> <p>Sommet de départ : 0</p> <p>Graphe : 0,1,- 1_0,2,4_1,2,3_1,3, 2_1,4,2_3,2,5_3,1, 1_4,3,-3</p>	<p>Exécution nettement plus longue</p>
--	--	----	--	---	--