GIT Department of Computer Engineering
CSE 222/505 - Spring 2022
Homework #08 Report

Mert Gürşimşir
1901042646

PART 1 – TIME ANALYSIS OF MyGraph

CODE	ANALYSIS
public Vertex newVertex(String label, double weight){ return new Vertex(label, weight); }	$\theta(1)$
public void addVertex(Vertex new_vertex){ if (numV >= capacity) reallocate(); for (int i = 0; i < lastIndex; ++i){ if (edges[i] == null){ new_vertex.setID(i); edges[i] = new Pair(new_vertex, new LinkedList<Edge>()); numV++; return; } } new_vertex.setID(numV); edges[numV] = new Pair(new_vertex, new LinkedList<Edge>()); numV++; lastIndex++; }	O(|V|) Where |V| is number of vertices.
public void printGraph(){ for (int i = 0; i < lastIndex; ++i) if (edges[i] != null) System.out.println(edges[i]); } *Each edges[i] is a Pair object so "System.out.println(edges[i]);" will call the toString method of the Pair. That toString method will also call the toString methods of Vertex and Edge class. Runtime of Edge class' toString method is constant, runtime of Vertex class' toString method depends on the number of properties.*	O(|E| + |V| + |P|) Where |E| is number of edges, |V| is number of vertices, |P| is number of properties.
public void addEdge(int vertexID1, int vertexID2, double weight){ if (vertexID1 < 0 || vertexID1 >= lastIndex || vertexID2 < 0 || vertexID2 >= lastIndex) System.out.println("Unproper ID."); else if (weight < 0) System.out.println("Unproper weight."); else{ Edge toAdd = new Edge(vertexID1, vertexID2, weight); for (Edge e:edges[vertexID1].getValue()){ if (e.equals(toAdd)){ System.out.println("This edge already exists."); return;} } edges[vertexID1].value.add(toAdd); if (!directed) edges[vertexID2].value.add(new Edge(vertexID2, vertexID1, weight)); } }	O(|V|) Where |V| is number of vertices. One vertex can have edge to |V|-1 vertices which leads to O(|V|).

```
public void removeEdge(int vertexID1, int vertexID2){
	if (vertexID1 < 0 || vertexID1 >= lastIndex || vertexID2 < 0 || vertexID2 >=
lastIndex) System.out.println("Unproper ID.");
	else{
		boolean flag = true;
		Edge toDelete = new Edge(vertexID1, vertexID2,
Double.POSITIVE_INFINITY);
		Iterator<Edge> iter = edgeIterator(vertexID1);
		if (iter == null){
			System.out.println("This edge does not exist.");
			return;
		}
		while (iter.hasNext()){
			if (iter.next().equals(toDelete)){
				iter.remove();
				flag = false;
				break;
			}
		}
		if (flag){
			System.out.println("This edge does not exist.");
			return;
		}
		if (!directed){
			Edge toDelete2 = new Edge(vertexID2, vertexID1,
Double.POSITIVE_INFINITY);
			Iterator<Edge> iter2 = edgeIterator(vertexID2);
			while (iter2.hasNext()){
				if (iter2.next().equals(toDelete2)){
					iter2.remove();
					break;
				}
			}
		}
	}
}
```

O(|V|)
Where |V| is the number of vertices.

public void removeVertex(int vertexID){ if (vertexID < 0 || vertexID >= lastIndex) System.out.println("Unproper ID."); else{ for (int i = 0; i < lastIndex; ++i){ if (edges[i] != null && i != vertexID){ Iterator<Edge> iter = edgeIterator(i); while (iter.hasNext()){ if (iter.next().getDest() == vertexID){ iter.remove(); break; } } } } edges[vertexID] = null; numV--; } }	$O(\|V\|^2)$ Where $\|V\|$ is the number of vertices.
public void removeVertex(String label){ for (int i = 0; i < lastIndex; ++i){ if (edges[i] != null){ if (edges[i].key.getLabel().equals(label)) removeVertex(i); } } }	$O(\|V\|^3)$ Where $\|V\|$ is the number of vertices.
public MyGraph filterVertices(String key, String filter){ try{ MyGraph toReturn = new MyGraph(capacity, directed); for (int i = 0; i < lastIndex; ++i){ if (edges[i] != null){ Map<String, String> properties = edges[i].getKey().getProperties(); for (Map.Entry<String, String> entry : properties.entrySet()){ if (entry.getKey().equals(key) && entry.getValue().equals(filter)){ toReturn.edges[i] = new Pair(null, new LinkedList<Edge>()); toReturn.edges[i].key = new Vertex(i, edges[i].getKey().getLabel(), edges[i].getKey().getWeight(), (HashMap<String,String>)edges[i].getKey().getProperties().clone()); Iterator<Edge> iter = edgeIterator(i); while (iter.hasNext()){ Edge edge = iter.next(); Map<String, String> properties2 = edges[edge.getDest()].getKey().getProperties(); for (Map.Entry<String, String> entry2 : properties2.entrySet()){ if (entry2.getKey().equals(key) && entry2.getValue().equals(filter)){ toReturn.edges[i].value.add(new Edge(edge.getSource(), edge.getDest(), edge.getWeight())); } } } toReturn.numV++; toReturn.lastIndex = i+1; break; } } } } return toReturn; }	$O(\|V\|^2.\|P\|)$ Where $\|V\|$ is the number of vertices and $\|P\|$ is the number of properties.

... /*Below part never happens*/ catch(Exception e){e.printStackTrace();} return new MyGraph(); }	
public Edge[][] exportMatrix(){ Edge[][] toReturn = new Edge[lastIndex][lastIndex]; for (int i = 0; i < lastIndex; ++i){ if (edges[i] != null){ Iterator<Edge> iter = edgeIterator(i); while(iter.hasNext()){ Edge edge = iter.next(); toReturn[edge.getSource()][edge.getDest()] = edge; } } } return toReturn; }	$O(\|V\|^2)$ Where $\|V\|$ is the number of vertices.

PART 2 – REPORT

1. SYSTEM REQUIREMENTS

To use this MyGraph object and its applications, you need to create an object by one of these 3 ways:

```
MyGraph tester = new MyGraph(1, false);
```

parameters are capacity and directed respectively – capacity cannot be smaller than 1, if so this will throw an exception

```
MyGraph tester2 = new MyGraph(false);
```

parameter is directed

```
MyGraph tester3 = new MyGraph();
```

Then you can use methods of the MyGraph.

To add vertex, you should create a new vertex first. Either with creating specially:

```
Vertex v1 = new Vertex("Mert", 42, "length", "7");
```

Or you can create by newVertex method by giving label and weight:

```
Vertex v11 = tester.newVertex("Turing", 17.);
```

You can add property to vertex by giving key and value respectively:

```
v1.addProperty("color", "red");
```

You can add vertex to the graph by simply give the vertex as parameter to addVertex method:

```
tester.addVertex(v1);
```

You can print the graph by:

```
tester.printGraph();
```

You can add edge by giving source, destination and weight respectively as parameter to addEdge method:

```
tester.addEdge(0, 1, 11);
```

Or you can give new Edge object as parameter to insert method:

```
tester3.insert(tester3.new Edge(0,1,4));
```

You can remove vertex either by index or label:

```
tester.removeVertex("Michio");
tester.removeVertex(0);
```

You can remove edge by giving source and destination respectively as parameter to removeEdge method:

```
tester.removeEdge(1, 6);
```

You can create adjacency matrix representation by exportMatrix method and print it by printMatrix method:

```
MyGraph.Edge[][] matrix = tester.exportMatrix();
tester.printMatrix(matrix);
```

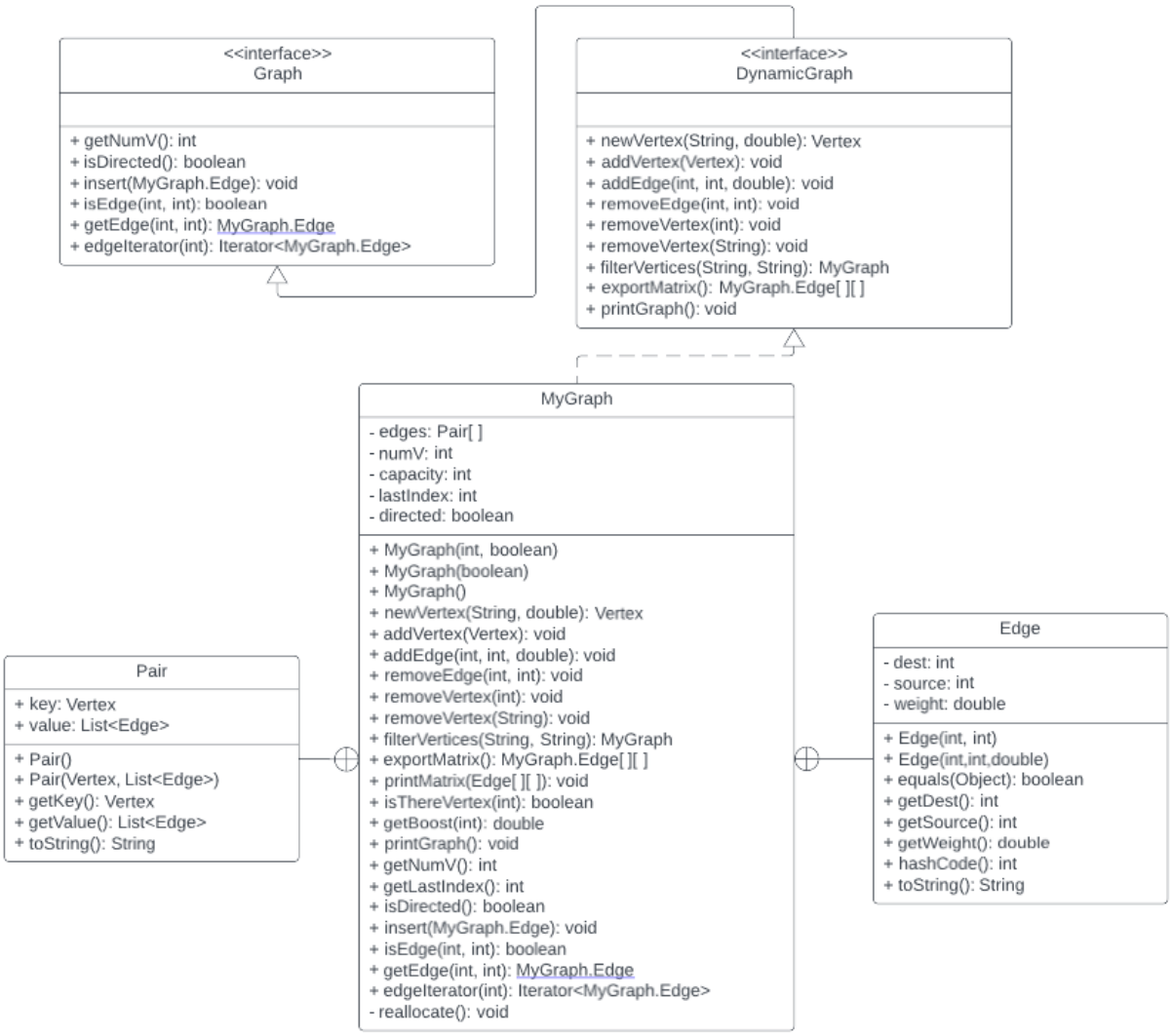
You can traverse through graph by static differenceOfTraversals method. You should give graph as parameter. This method will print proper informations and return the difference between the total distances of two traversal methods (BFS and DFS):

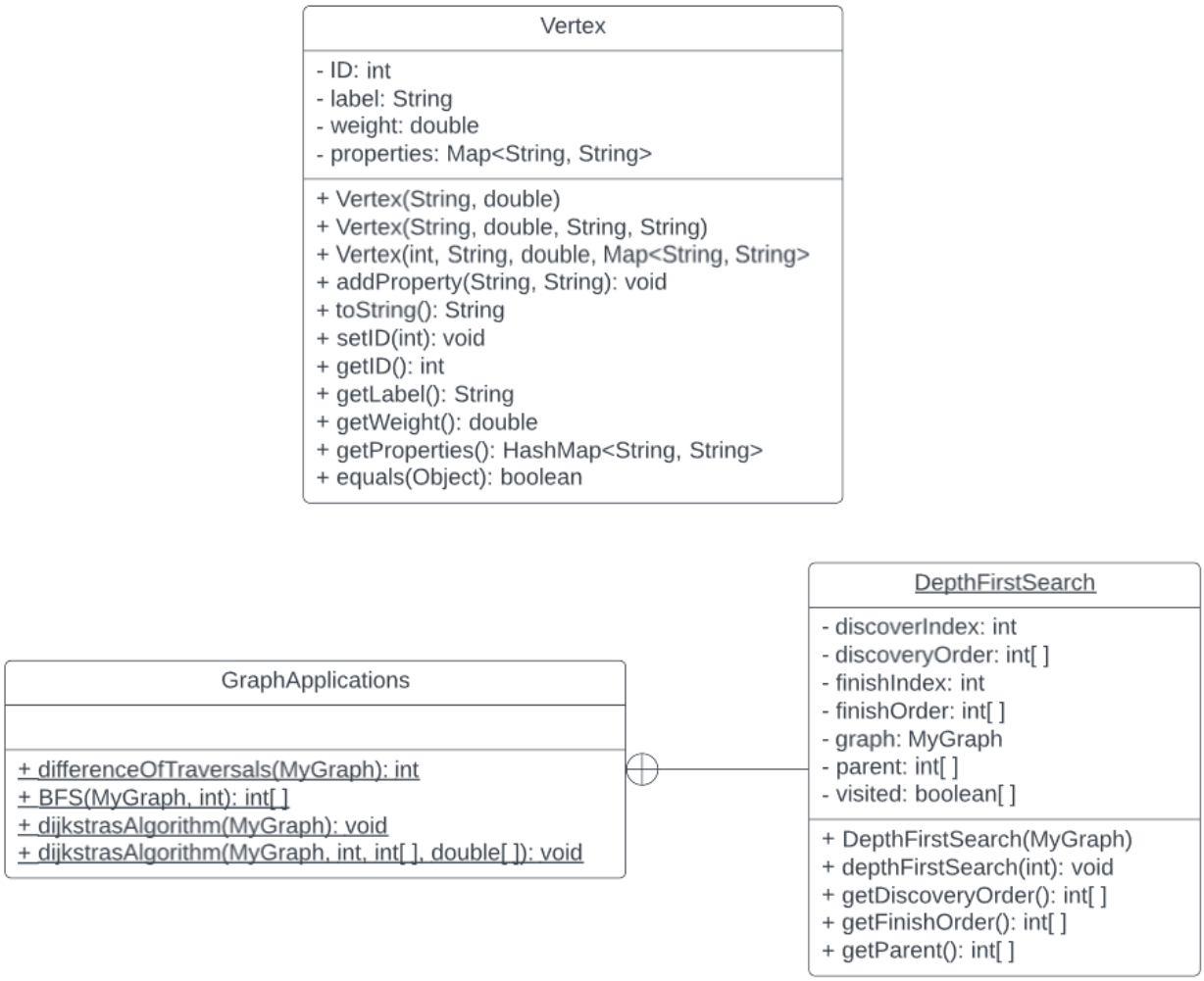
```
int difference = GraphApplications.differenceOfTraversals(tester);
System.out.printf("Difference of traversals: %d\n\n", difference);
```

You can find the shortest path by dijkstras algorithm with boosting values:

```
GraphApplications.dijkstrasAlgorithm(tester);
```

2. CLASS DIAGRAMS




Vertex
- ID: int
- label: String
- weight: double
- properties: Map<String, String>
+ Vertex(String, double)
+ Vertex(String, double, String, String)
+ Vertex(int, String, double, Map<String, String>
+ addProperty(String, String): void
+ toString(): String
+ setID(int): void
+ getID(): int
+ getLabel(): String
+ getWeight(): double
+ getProperties(): HashMap<String, String>
+ equals(Object): boolean
DepthFirstSearch
- discoverIndex: int
- discoveryOrder: int[ ]
- finishIndex: int
- finishOrder: int[ ]
- graph: MyGraph
- parent: int[ ]
- visited: boolean[ ]
+ DepthFirstSearch(MyGraph)
+ depthFirstSearch(int): void
+ getDiscoveryOrder(): int[ ]
+ getFinishOrder(): int[ ]
+ getParent(): int[ ]
GraphApplications
+ differenceOfTraversals(MyGraph): int
+ BFS(MyGraph, int): int[ ]
+ dijkstrasAlgorithm(MyGraph): void
+ dijkstrasAlgorithm(MyGraph, int, int[ ], double[ ]): void

3. PROBLEM SOLUTION APPROACH

Firstly, I have tried to do my best to implement new dynamic graph (MyGraph) since I was going to use this graph for traversals and dijkstra's algorithm.

Firstly, I have started with implementing the Vertex class. I decided to put properties in a HashMap to handle them easily. Also I have added addProperty method for this class to add properties easily. After I had completed vertex class with keeping also labels and weights, I have started to implement MyGraph class.

It was hard for me to decide proper way to keep the edges and vertices properly. Best solution for me is creating the "Pair" class and keeping the "Pair"s which includes "Vertex" and "LinkedList<Edge>" as array elements. After this process, it was easy to implement given methods except one thing. This one thing is removing the vertex. First thing that comes to my mind to handle this is shifting the array when a vertex is removed but this will cause change in the IDs of the vertices. So I keep null for the proper position when a vertex is deleted.

After design of the this special process, I have started to implement BFS, and DFS. It was not hard for me to implement BFS, but it is a little hard to implement DFS because of special requirement to check edges. I handled this problem by creating the order first. Than it was easy to complete traversal. The special thing that I have to be careful is null indices in the array. So I handled this by skipping that places.

Lastly, I have implemented the dijkstra's algorithm. Firstly, I have analyzed the code in the book very carefully. Then implement this regular algorithm to MyGraph. After it had worked, I have brought boostings into play. I have subtracted these boosting value from the distance when it is seen in the path. So with this, dijkstra's algorithm has run correctly too.

4. TEST CASES

Test cases include:

- **Adding property to vertex**
- **Adding vertex to graph**
- **Adding edge to graph**
- **Printing the graph**
- **Filtering the vertices and printing this new subgraph**
- **Removing the vertex**
- **Removing the edge**
- **Traverse through graph**
    - **With BFS**
    - **With DFS**
- **Print out the difference of total distances of two traversals**
- **Export adjacency matrix representation of the graph**
- **Applying Dijkstra's algorithm to find the shortest path**
    - **By using boosting values**

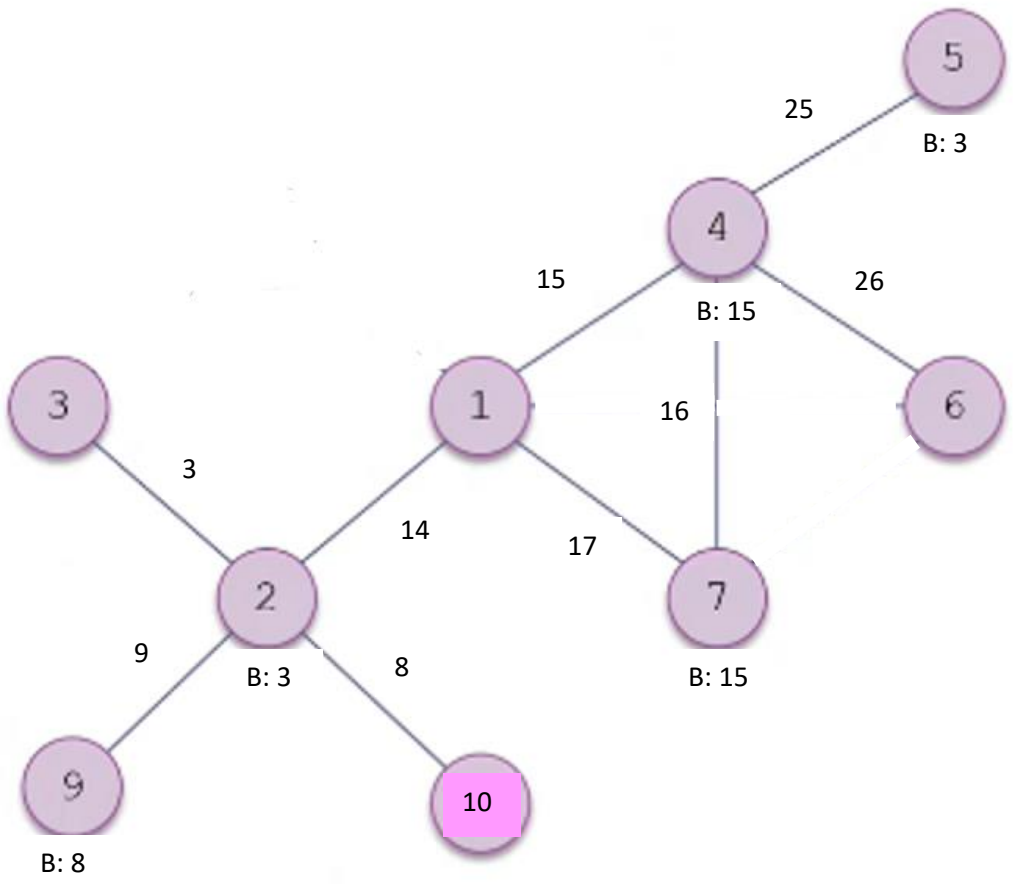
```
System.out.println("WELCOME TO TEST OF THE DYNAMIC GRAPH AND ITS APPLICATIONS");
System.out.println("------------------------------------------------------------\n");

System.out.println("Creating new undirected graph and adding 11 new vertex to it:");
MyGraph tester = new MyGraph(1, false);
Vertex v1 = new Vertex("Mert", 42, "length", "7");
Vertex v2 = new Vertex("Ahsen", 30, "length", "7");
Vertex v3 = new Vertex("Sevilgen", 100, "boosting", "3");
Vertex v4 = new Vertex("Burak", 100, "color", "red");
Vertex v5 = new Vertex("Ferda", 100, "boosting", "15");
Vertex v6 = new Vertex("Murakami", 33, "boosting", "3");
Vertex v7 = new Vertex("Feynman", 62, "size", "57");
Vertex v8 = new Vertex("Stephen", 99, "boosting", "15");
Vertex v9 = new Vertex("Michio", 71, "color", "blue");
Vertex v10 = new Vertex("Hugo", 6, "boosting", "8");
Vertex v11 = tester.newVertex("Turing", 17.);
v1.addProperty("color", "red");
v2.addProperty("color", "red");
v11.addProperty("color", "red");
tester.addVertex(v1); tester.addVertex(v2); tester.addVertex(v3);
tester.addVertex(v4); tester.addVertex(v5); tester.addVertex(v6);
tester.addVertex(v7); tester.addVertex(v8); tester.addVertex(v9);
tester.addVertex(v10);tester.addVertex(v11);
tester.printGraph();

System.out.println("------------------------------------------------------------------------");

System.out.println("Adding edges to graph:\n");
tester.addEdge(0, 1, 11); tester.addEdge(0, 3, 13); tester.addEdge(1, 2, 14);
tester.addEdge(1, 4, 15); tester.addEdge(1, 6, 16); tester.addEdge(1, 7, 17);
tester.addEdge(2, 3, 3);  tester.addEdge(2, 8, 8);  tester.addEdge(2, 9, 9);
tester.addEdge(4, 5, 25); tester.addEdge(4, 6, 26); tester.addEdge(4, 7, 16);
tester.addEdge(6, 7, 6);  tester.addEdge(2, 10, 8);
tester.printGraph();

System.out.println("------------------------------------------------------------------------");

System.out.println("Now filtering the vertices with key \"color\" and with filter \"red\":\n");
tester.filterVertices("color", "red").printGraph();

System.out.println("------------------------------------------------------------------------");

System.out.println("Now removing the vertices with label \"Michio\" and with index 0:\n");
tester.removeVertex("Michio");
tester.removeVertex(0);
tester.printGraph();

System.out.println("------------------------------------------------------------------------");
```

```
System.out.println("Now removing the edges from 1 to 6 and 6 to 7:\n");
tester.removeEdge(1, 6); tester.removeEdge(6, 7);
tester.printGraph();

System.out.println("------------------------------------------------------------------------");

System.out.println("Now try to remove edge from non-existing index 0 to 4:\n");
tester.removeEdge(0, 4);

System.out.println("------------------------------------------------------------------------");

System.out.println("Also, if we try to delete a non-existing edge, it will indicate the error (try to delete edge from 1 to 10):\n");
tester.removeEdge(1, 10);

System.out.println("------------------------------------------------------------------------");

System.out.println("If we try to give negative weight, it will indicate the error (add edge between 1 and 10 with weight -1):\n");
tester.addEdge(1,10,-1);

System.out.println("------------------------------------------------------------------------");

System.out.println("Now exporting the matrix representation (-1 represents vertex doesn't exist):\n");
MyGraph.Edge[][] matrix = tester.exportMatrix();
tester.printMatrix(matrix);

System.out.println("------------------------------------------------------------------------");

System.out.println("Now test the traversals (-2 represents vertex with that index doesn't exist, -1 represents there is no predecessor):\n");
int difference = GraphApplications.differenceOfTraversals(tester);
System.out.printf("Difference of traversals: %d\n\n", difference);

System.out.println("------------------------------------------------------------------------");

System.out.println("Now test the dijkstras algorithm with boosting values (-2 represents vertex with that index doesn't exist, -1 represents there is no predecessor):\n");
GraphApplications.dijkstrasAlgorithm(tester);

System.out.println("\n\n========================================================================\n");
```

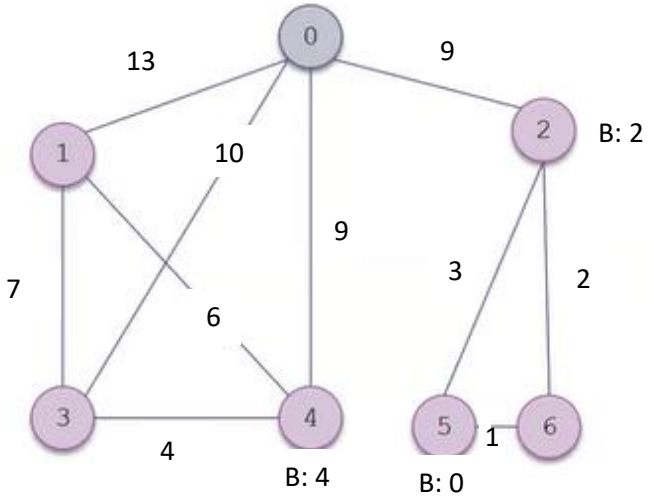
```
System.out.println("\n\n================================================================================\n");

System.out.println("Creating new graph and adding 9 new vertex to it:");
MyGraph tester2 = new MyGraph(1, false);
v3.addProperty("boosting", "2");
v5.addProperty("boosting", "4");
v6.addProperty("boosting", "0");
tester2.addVertex(v1); tester2.addVertex(v2); tester2.addVertex(v3);
tester2.addVertex(v4); tester2.addVertex(v5); tester2.addVertex(v6);
tester2.addVertex(v7);
tester2.printGraph();

System.out.println("--------------------------------------------------------------------------");

System.out.println("Adding edges to graph:\n");
tester2.addEdge(0,1,13); tester2.addEdge(0,2,9); tester2.addEdge(0,3,10);
tester2.addEdge(0,4,9);  tester2.addEdge(1,4,6); tester2.addEdge(1,3,7);
tester2.addEdge(3,4,4);  tester2.addEdge(2,5,3); tester2.addEdge(2,6,2); tester2.addEdge(5,6,1);
tester2.printGraph();

System.out.println("--------------------------------------------------------------------------");

System.out.println("Now test the traversals:\n");
difference = GraphApplications.differenceOfTraversals(tester2);
System.out.printf("Difference of traversals: %d\n\n", difference);

System.out.println("--------------------------------------------------------------------------");

System.out.println("Now test the dijkstras algorithm with boosting values:\n");
GraphApplications.dijkstrasAlgorithm(tester2);

System.out.println("\n\n================================================================================\n");
```

```
System.out.println("\n\n================================================================================\n");

System.out.println("Creating new directed graph and adding 4 new vertex to it:");
Vertex v111 = new Vertex("Mert", 42, "boosting", "2");
Vertex v222 = new Vertex("Ahsen", 30, "boosting", "3");
Vertex v333 = new Vertex("Sevilgen", 100, "boosting", "4");
Vertex v444 = new Vertex("Burak", 100, "boosting", "4");
MyGraph tester3 = new MyGraph();
tester3.addVertex(v111); tester3.addVertex(v222); tester3.addVertex(v333); tester3.addVertex(v444);
tester3.printGraph();

System.out.println("--------------------------------------------------------------------------");

System.out.println("Adding edges to graph:\n");
tester3.addEdge(0,1,4);   tester3.addEdge(0,2,4); tester3.addEdge(1,2,5);
tester3.addEdge(0,3,10); tester3.addEdge(2,3,5); tester3.addEdge(1,3,5);
tester3.printGraph();

System.out.println("--------------------------------------------------------------------------");

System.out.println("Now removing the edges connected to 0:\n");
tester3.removeEdge(0, 1); tester3.removeEdge(0, 2); tester3.removeEdge(0, 3);
tester3.printGraph();

System.out.println("--------------------------------------------------------------------------");

System.out.println("Now test the dijkstras algorithm with boosting values:\n");
GraphApplications.dijkstrasAlgorithm(tester3);

System.out.println("--------------------------------------------------------------------------");

System.out.println("Now adding the edges again:\n");
tester3.insert(tester3.new Edge(0,1,4)); tester3.insert(tester3.new Edge(0,2,4)); tester3.insert(tester3.new Edge(0,3,10));
tester3.printGraph();

System.out.println("--------------------------------------------------------------------------");

System.out.println("Now test the traversals:\n");
difference = GraphApplications.differenceOfTraversals(tester3);
System.out.printf("Difference of traversals: %d\n\n", difference);

System.out.println("--------------------------------------------------------------------------");

System.out.println("Now test the dijkstras algorithm with boosting values:\n");
GraphApplications.dijkstrasAlgorithm(tester3);

System.out.println("\n\n================================================================================\n");
```

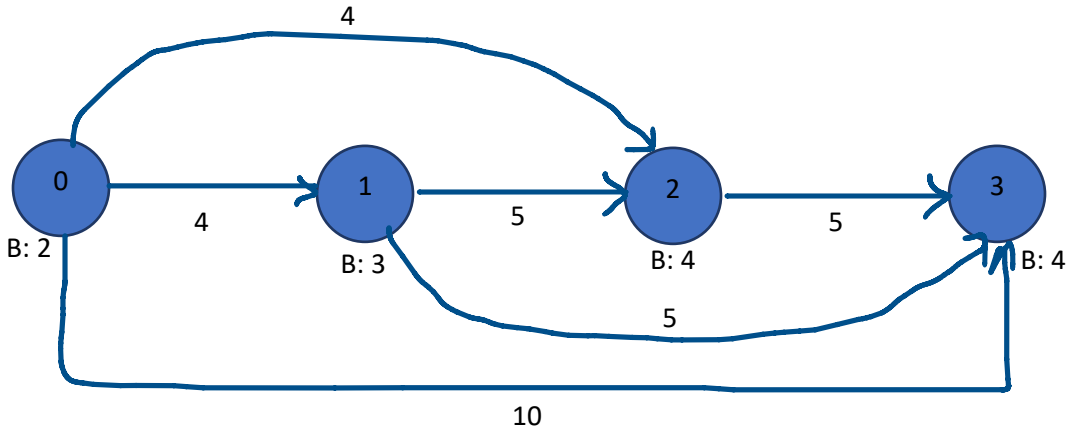
tester graph for illustration to follow traversals:



tester2 graph for illustration to follow traversals:



tester3 graph for illustration to follow traversals:



B → boosting values

5. RUNNING AND RESULTS

```
WELCOME TO TEST OF THE DYNAMIC GRAPH AND ITS APPLICATIONS
---------------------------------------------------------

Creating new undirected graph and adding 11 new vertex to it:
ID: 0, Label: Mert, Weight: 42.0, Key: color <=> Value: red, Key: length <=> Value: 7

ID: 1, Label: Ahsen, Weight: 30.0, Key: color <=> Value: red, Key: length <=> Value: 7

ID: 2, Label: Sevilgen, Weight: 100.0, Key: boosting <=> Value: 3

ID: 3, Label: Burak, Weight: 100.0, Key: color <=> Value: red

ID: 4, Label: Ferda, Weight: 100.0, Key: boosting <=> Value: 15

ID: 5, Label: Murakami, Weight: 33.0, Key: boosting <=> Value: 3

ID: 6, Label: Feynman, Weight: 62.0, Key: size <=> Value: 57

ID: 7, Label: Stephen, Weight: 99.0, Key: boosting <=> Value: 15

ID: 8, Label: Michio, Weight: 71.0, Key: color <=> Value: blue

ID: 9, Label: Hugo, Weight: 6.0, Key: boosting <=> Value: 8

ID: 10, Label: Turing, Weight: 17.0, Key: color <=> Value: red

---------------------------------------------------------------------
Adding edges to graph:

ID: 0, Label: Mert, Weight: 42.0, Key: color <=> Value: red, Key: length <=> Value: 7
     --> Source: 0, Dest: 1, Weight: 11.0
     --> Source: 0, Dest: 3, Weight: 13.0

ID: 1, Label: Ahsen, Weight: 30.0, Key: color <=> Value: red, Key: length <=> Value: 7
     --> Source: 1, Dest: 0, Weight: 11.0
     --> Source: 1, Dest: 2, Weight: 14.0
     --> Source: 1, Dest: 4, Weight: 15.0
     --> Source: 1, Dest: 6, Weight: 16.0
     --> Source: 1, Dest: 7, Weight: 17.0

ID: 2, Label: Sevilgen, Weight: 100.0, Key: boosting <=> Value: 3
     --> Source: 2, Dest: 1, Weight: 14.0
     --> Source: 2, Dest: 3, Weight: 3.0
     --> Source: 2, Dest: 8, Weight: 8.0
     --> Source: 2, Dest: 9, Weight: 9.0
     --> Source: 2, Dest: 10, Weight: 8.0
```

```
ID: 3, Label: Burak, Weight: 100.0, Key: color <=> Value: red
     --> Source: 3, Dest: 0, Weight: 13.0
     --> Source: 3, Dest: 2, Weight: 3.0

ID: 4, Label: Ferda, Weight: 100.0, Key: boosting <=> Value: 15
     --> Source: 4, Dest: 1, Weight: 15.0
     --> Source: 4, Dest: 5, Weight: 25.0
     --> Source: 4, Dest: 6, Weight: 26.0
     --> Source: 4, Dest: 7, Weight: 16.0

ID: 5, Label: Murakami, Weight: 33.0, Key: boosting <=> Value: 3
     --> Source: 5, Dest: 4, Weight: 25.0

ID: 6, Label: Feynman, Weight: 62.0, Key: size <=> Value: 57
     --> Source: 6, Dest: 1, Weight: 16.0
     --> Source: 6, Dest: 4, Weight: 26.0
     --> Source: 6, Dest: 7, Weight: 6.0

ID: 7, Label: Stephen, Weight: 99.0, Key: boosting <=> Value: 15
     --> Source: 7, Dest: 1, Weight: 17.0
     --> Source: 7, Dest: 4, Weight: 16.0
     --> Source: 7, Dest: 6, Weight: 6.0

ID: 8, Label: Michio, Weight: 71.0, Key: color <=> Value: blue
     --> Source: 8, Dest: 2, Weight: 8.0

ID: 9, Label: Hugo, Weight: 6.0, Key: boosting <=> Value: 8
     --> Source: 9, Dest: 2, Weight: 9.0

ID: 10, Label: Turing, Weight: 17.0, Key: color <=> Value: red
     --> Source: 10, Dest: 2, Weight: 8.0

------------------------------------------------------------------------
Now filtering the vertices with key "color" and with filter "red":

ID: 0, Label: Mert, Weight: 42.0, Key: color <=> Value: red, Key: length <=> Value: 7
     --> Source: 0, Dest: 1, Weight: 11.0
     --> Source: 0, Dest: 3, Weight: 13.0

ID: 1, Label: Ahsen, Weight: 30.0, Key: color <=> Value: red, Key: length <=> Value: 7
     --> Source: 1, Dest: 0, Weight: 11.0

ID: 3, Label: Burak, Weight: 100.0, Key: color <=> Value: red
     --> Source: 3, Dest: 0, Weight: 13.0
```

```
ID: 10, Label: Turing, Weight: 17.0, Key: color <=> Value: red

------------------------------------------------------------------------
Now removing the vertices with label "Michio" and with index 0:

ID: 1, Label: Ahsen, Weight: 30.0, Key: color <=> Value: red, Key: length <=> Value: 7
     --> Source: 1, Dest: 2, Weight: 14.0
     --> Source: 1, Dest: 4, Weight: 15.0
     --> Source: 1, Dest: 6, Weight: 16.0
     --> Source: 1, Dest: 7, Weight: 17.0

ID: 2, Label: Sevilgen, Weight: 100.0, Key: boosting <=> Value: 3
     --> Source: 2, Dest: 1, Weight: 14.0
     --> Source: 2, Dest: 3, Weight: 3.0
     --> Source: 2, Dest: 9, Weight: 9.0
     --> Source: 2, Dest: 10, Weight: 8.0

ID: 3, Label: Burak, Weight: 100.0, Key: color <=> Value: red
     --> Source: 3, Dest: 2, Weight: 3.0

ID: 4, Label: Ferda, Weight: 100.0, Key: boosting <=> Value: 15
     --> Source: 4, Dest: 1, Weight: 15.0
     --> Source: 4, Dest: 5, Weight: 25.0
     --> Source: 4, Dest: 6, Weight: 26.0
     --> Source: 4, Dest: 7, Weight: 16.0

ID: 5, Label: Murakami, Weight: 33.0, Key: boosting <=> Value: 3
     --> Source: 5, Dest: 4, Weight: 25.0

ID: 6, Label: Feynman, Weight: 62.0, Key: size <=> Value: 57
     --> Source: 6, Dest: 1, Weight: 16.0
     --> Source: 6, Dest: 4, Weight: 26.0
     --> Source: 6, Dest: 7, Weight: 6.0

ID: 7, Label: Stephen, Weight: 99.0, Key: boosting <=> Value: 15
     --> Source: 7, Dest: 1, Weight: 17.0
     --> Source: 7, Dest: 4, Weight: 16.0
     --> Source: 7, Dest: 6, Weight: 6.0

ID: 9, Label: Hugo, Weight: 6.0, Key: boosting <=> Value: 8
     --> Source: 9, Dest: 2, Weight: 9.0

ID: 10, Label: Turing, Weight: 17.0, Key: color <=> Value: red
     --> Source: 10, Dest: 2, Weight: 8.0

------------------------------------------------------------------------
```

```
Now removing the edges from 1 to 6 and 6 to 7:

ID: 1, Label: Ahsen, Weight: 30.0, Key: color <=> Value: red, Key: length <=> Value: 7
     --> Source: 1, Dest: 2, Weight: 14.0
     --> Source: 1, Dest: 4, Weight: 15.0
     --> Source: 1, Dest: 7, Weight: 17.0

ID: 2, Label: Sevilgen, Weight: 100.0, Key: boosting <=> Value: 3
     --> Source: 2, Dest: 1, Weight: 14.0
     --> Source: 2, Dest: 3, Weight: 3.0
     --> Source: 2, Dest: 9, Weight: 9.0
     --> Source: 2, Dest: 10, Weight: 8.0

ID: 3, Label: Burak, Weight: 100.0, Key: color <=> Value: red
     --> Source: 3, Dest: 2, Weight: 3.0

ID: 4, Label: Ferda, Weight: 100.0, Key: boosting <=> Value: 15
     --> Source: 4, Dest: 1, Weight: 15.0
     --> Source: 4, Dest: 5, Weight: 25.0
     --> Source: 4, Dest: 6, Weight: 26.0
     --> Source: 4, Dest: 7, Weight: 16.0

ID: 5, Label: Murakami, Weight: 33.0, Key: boosting <=> Value: 3
     --> Source: 5, Dest: 4, Weight: 25.0

ID: 6, Label: Feynman, Weight: 62.0, Key: size <=> Value: 57
     --> Source: 6, Dest: 4, Weight: 26.0

ID: 7, Label: Stephen, Weight: 99.0, Key: boosting <=> Value: 15
     --> Source: 7, Dest: 1, Weight: 17.0
     --> Source: 7, Dest: 4, Weight: 16.0

ID: 9, Label: Hugo, Weight: 6.0, Key: boosting <=> Value: 8
     --> Source: 9, Dest: 2, Weight: 9.0

ID: 10, Label: Turing, Weight: 17.0, Key: color <=> Value: red
     --> Source: 10, Dest: 2, Weight: 8.0

------------------------------------------------------------------------
Now try to remove edge from non-existing index 0 to 4:

This edge does not exist.
------------------------------------------------------------------------
Also, if we try to delete a non-existing edge, it will indicate the error (try to delete edge from 1 to 10):

This edge does not exist.
```

```
------------------------------------------------------------------------
If we try to give negative weight, it will indicate the error (add edge between 1 and 10 with weight -1):

Unproper weight.
------------------------------------------------------------------------
Now exporting the matrix representation (-1 represents vertex doesn't exist):

 -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1
  0   0   1   0   1   0   0   1   0   0   0
  0   1   0   1   0   0   0   0   0   1   1
  0   0   1   0   0   0   0   0   0   0   0
  0   1   0   0   0   1   1   1   0   0   0
  0   0   0   0   1   0   0   0   0   0   0
  0   0   0   0   1   0   0   0   0   0   0
  0   1   0   0   1   0   0   0   0   0   0
 -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1
  0   0   1   0   0   0   0   0   0   0   0
  0   0   1   0   0   0   0   0   0   0   0
------------------------------------------------------------------------
Now test the traversals (-2 represents vertex with that index doesn't exist, -1 represents there is no predecessor):

Parent array of the BFS: -2 -1  1  2  1  4  4  1 -2  2  2

Parent array of the DFS: -2 -1  1  2  1  4  4  4 -2  2  2
Discovery order of the DFS:  1  2  3 10  9  4  7  5  6
Finish order of the DFS:  3 10  9  2  7  5  6  4  1

Total distance of the BFS path: 13
Total distance of the DFS: 14
Difference of traversals: 1

------------------------------------------------------------------------
Now test the dijkstras algorithm with boosting values (-2 represents vertex with that index doesn't exist, -1 represents there is no predecessor):

Predecessor array --> -2 -1 1 2 1 4 4 4 -2 2 2
Distance array --> -2.00 0.00 14.00 14.00 15.00 25.00 26.00 16.00 -2.00 20.00 19.00


========================================================================
```

```
Creating new graph and adding 9 new vertex to it:
ID: 0, Label: Mert, Weight: 42.0, Key: color <=> Value: red, Key: length <=> Value: 7

ID: 1, Label: Ahsen, Weight: 30.0, Key: color <=> Value: red, Key: length <=> Value: 7

ID: 2, Label: Sevilgen, Weight: 100.0, Key: boosting <=> Value: 2

ID: 3, Label: Burak, Weight: 100.0, Key: color <=> Value: red

ID: 4, Label: Ferda, Weight: 100.0, Key: boosting <=> Value: 4

ID: 5, Label: Murakami, Weight: 33.0, Key: boosting <=> Value: 0

ID: 6, Label: Feynman, Weight: 62.0, Key: size <=> Value: 57

------------------------------------------------------------------------
Adding edges to graph:

ID: 0, Label: Mert, Weight: 42.0, Key: color <=> Value: red, Key: length <=> Value: 7
     --> Source: 0, Dest: 1, Weight: 13.0
     --> Source: 0, Dest: 2, Weight: 9.0
     --> Source: 0, Dest: 3, Weight: 10.0
     --> Source: 0, Dest: 4, Weight: 9.0

ID: 1, Label: Ahsen, Weight: 30.0, Key: color <=> Value: red, Key: length <=> Value: 7
     --> Source: 1, Dest: 0, Weight: 13.0
     --> Source: 1, Dest: 4, Weight: 6.0
     --> Source: 1, Dest: 3, Weight: 7.0

ID: 2, Label: Sevilgen, Weight: 100.0, Key: boosting <=> Value: 2
     --> Source: 2, Dest: 0, Weight: 9.0
     --> Source: 2, Dest: 5, Weight: 3.0
     --> Source: 2, Dest: 6, Weight: 2.0

ID: 3, Label: Burak, Weight: 100.0, Key: color <=> Value: red
     --> Source: 3, Dest: 0, Weight: 10.0
     --> Source: 3, Dest: 1, Weight: 7.0
     --> Source: 3, Dest: 4, Weight: 4.0

ID: 4, Label: Ferda, Weight: 100.0, Key: boosting <=> Value: 4
     --> Source: 4, Dest: 0, Weight: 9.0
     --> Source: 4, Dest: 1, Weight: 6.0
     --> Source: 4, Dest: 3, Weight: 4.0
```

```
ID: 5, Label: Murakami, Weight: 33.0, Key: boosting <=> Value: 0
     --> Source: 5, Dest: 2, Weight: 3.0
     --> Source: 5, Dest: 6, Weight: 1.0

ID: 6, Label: Feynman, Weight: 62.0, Key: size <=> Value: 57
     --> Source: 6, Dest: 2, Weight: 2.0
     --> Source: 6, Dest: 5, Weight: 1.0

------------------------------------------------------------------------
Now test the traversals:

Parent array of the BFS: -1  0  0  0  0  2  2

Parent array of the DFS: -1  3  0  4  0  6  2
Discovery order of the DFS:  0  2  6  5  4  3  1
Finish order of the DFS:  5  6  2  1  3  4  0

Total distance of the BFS path: 8
Total distance of the DFS: 12
Difference of traversals: 4

------------------------------------------------------------------------
Now test the dijkstras algorithm with boosting values:

Predecessor array --> -1 4 0 4 0 2 2
Distance array --> 0.00 11.00 9.00 9.00 9.00 10.00 9.00


========================================================================
```

```
========================================================================

Creating new directed graph and adding 4 new vertex to it:
ID: 0, Label: Mert, Weight: 42.0, Key: boosting <=> Value: 2

ID: 1, Label: Ahsen, Weight: 30.0, Key: boosting <=> Value: 3

ID: 2, Label: Sevilgen, Weight: 100.0, Key: boosting <=> Value: 4

ID: 3, Label: Burak, Weight: 100.0, Key: boosting <=> Value: 4

------------------------------------------------------------------------
Adding edges to graph:

ID: 0, Label: Mert, Weight: 42.0, Key: boosting <=> Value: 2
     --> Source: 0, Dest: 1, Weight: 4.0
     --> Source: 0, Dest: 2, Weight: 4.0
     --> Source: 0, Dest: 3, Weight: 10.0

ID: 1, Label: Ahsen, Weight: 30.0, Key: boosting <=> Value: 3
     --> Source: 1, Dest: 2, Weight: 5.0
     --> Source: 1, Dest: 3, Weight: 5.0

ID: 2, Label: Sevilgen, Weight: 100.0, Key: boosting <=> Value: 4
     --> Source: 2, Dest: 3, Weight: 5.0

ID: 3, Label: Burak, Weight: 100.0, Key: boosting <=> Value: 4

------------------------------------------------------------------------
Now removing the edges connected to 0:

ID: 0, Label: Mert, Weight: 42.0, Key: boosting <=> Value: 2

ID: 1, Label: Ahsen, Weight: 30.0, Key: boosting <=> Value: 3
     --> Source: 1, Dest: 2, Weight: 5.0
     --> Source: 1, Dest: 3, Weight: 5.0

ID: 2, Label: Sevilgen, Weight: 100.0, Key: boosting <=> Value: 4
     --> Source: 2, Dest: 3, Weight: 5.0

ID: 3, Label: Burak, Weight: 100.0, Key: boosting <=> Value: 4

------------------------------------------------------------------------
```

```
------------------------------------------------------------------------
Now test the dijkstras algorithm with boosting values:

Predecessor array --> -1 -1 -1 -1
Distance array --> 0.00 Infinity Infinity Infinity
------------------------------------------------------------------------
Now adding the edges again:

ID: 0, Label: Mert, Weight: 42.0, Key: boosting <=> Value: 2
     --> Source: 0, Dest: 1, Weight: 4.0
     --> Source: 0, Dest: 2, Weight: 4.0
     --> Source: 0, Dest: 3, Weight: 10.0

ID: 1, Label: Ahsen, Weight: 30.0, Key: boosting <=> Value: 3
     --> Source: 1, Dest: 2, Weight: 5.0
     --> Source: 1, Dest: 3, Weight: 5.0

ID: 2, Label: Sevilgen, Weight: 100.0, Key: boosting <=> Value: 4
     --> Source: 2, Dest: 3, Weight: 5.0

ID: 3, Label: Burak, Weight: 100.0, Key: boosting <=> Value: 4

------------------------------------------------------------------------
Now test the traversals:

Parent array of the BFS: -1  0  0  0

Parent array of the DFS: -1  0  1  2
Discovery order of the DFS:  0  1  2  3
Finish order of the DFS:  3  2  1  0

Total distance of the BFS path: 3
Total distance of the DFS: 6
Difference of traversals: 3

------------------------------------------------------------------------
Now test the dijkstras algorithm with boosting values:

Predecessor array --> -1 0 0 2
Distance array --> 0.00 4.00 4.00 5.00


========================================================================
```