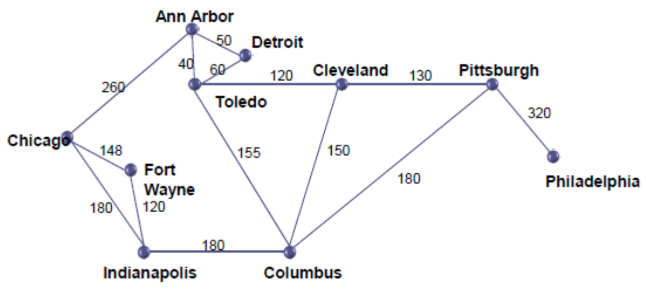
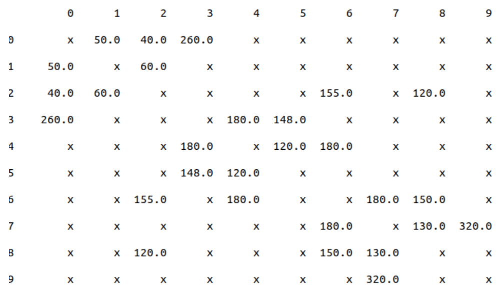
GTU Department of Computer Engineering
CSE 222/505 - Spring 2022
Homework 8 Report

Emirkan Burak Yılmaz
1901042659





```
         0      1      2      3      4      5      6      7      8      9
0        x   50.0   40.0  260.0      x      x      x      x      x      x
1     50.0      x   60.0      x      x      x      x      x      x      x
2     40.0   60.0      x      x      x      x  155.0      x  120.0      x
3    260.0      x      x      x  180.0  148.0      x      x      x      x
4        x      x      x  180.0      x  120.0  180.0      x      x      x
5        x      x      x  148.0  120.0      x      x      x      x      x
6        x      x  155.0      x  180.0      x      x  180.0  150.0      x
7        x      x      x      x      x      x  180.0      x  130.0  320.0
8        x      x  120.0      x      x      x  150.0  130.0      x      x
9        x      x      x      x      x      x      x  320.0      x      x
```

```
(0, Ann Arbor): --> [(8, Toledo) | 40.0] --> [(4, Detroit) | 50.0] --> [(1, Chicago) | 260.0]

(4, Detroit): --> [(0, Ann Arbor) | 50.0] --> [(8, Toledo) | 60.0]

(8, Toledo): --> [(0, Ann Arbor) | 40.0] --> [(4, Detroit) | 60.0] --> [(3, Columbus) | 155.0] --> [(2, Clevland) | 120.0]

(1, Chicago): --> [(0, Ann Arbor) | 260.0] --> [(5, Indianapolis) | 180.0] --> [(9, Fort Wayne) | 148.0]

(5, Indianapolis): --> [(1, Chicago) | 180.0] --> [(9, Fort Wayne) | 120.0] --> [(3, Columbus) | 180.0]

(9, Fort Wayne): --> [(1, Chicago) | 148.0] --> [(5, Indianapolis) | 120.0]

(3, Columbus): --> [(5, Indianapolis) | 180.0] --> [(8, Toledo) | 155.0] --> [(2, Clevland) | 150.0] --> [(6, Pitsburg) | 180.0]

(6, Pitsburg): --> [(3, Columbus) | 180.0] --> [(7, Philadelphia) | 320.0] --> [(2, Clevland) | 130.0]

(2, Clevland): --> [(3, Columbus) | 150.0] --> [(8, Toledo) | 120.0] --> [(6, Pitsburg) | 130.0]

(7, Philadelphia): --> [(6, Pitsburg) | 320.0]
```

1. SYSTEM REQUIREMENTS

Q1. DynamicGraph & MyGraph

- Define the DynamicGraph interface by extending the Graph interface in the book.
- DynamicGraph interface should have extra properties like adding/removing vertex/edges, exporting the graph as adjacency matrix and printing the graph in adjacency list format.
- Define the class MyGraph for the implementation of DynamicGraph interface and use adjacency list representation in the implementation.
- Define the Vertex generic class for representing the vertices in the graph. A vertex must have an index, a label, and a weight.
- The vertices may have user-defined additional properties.

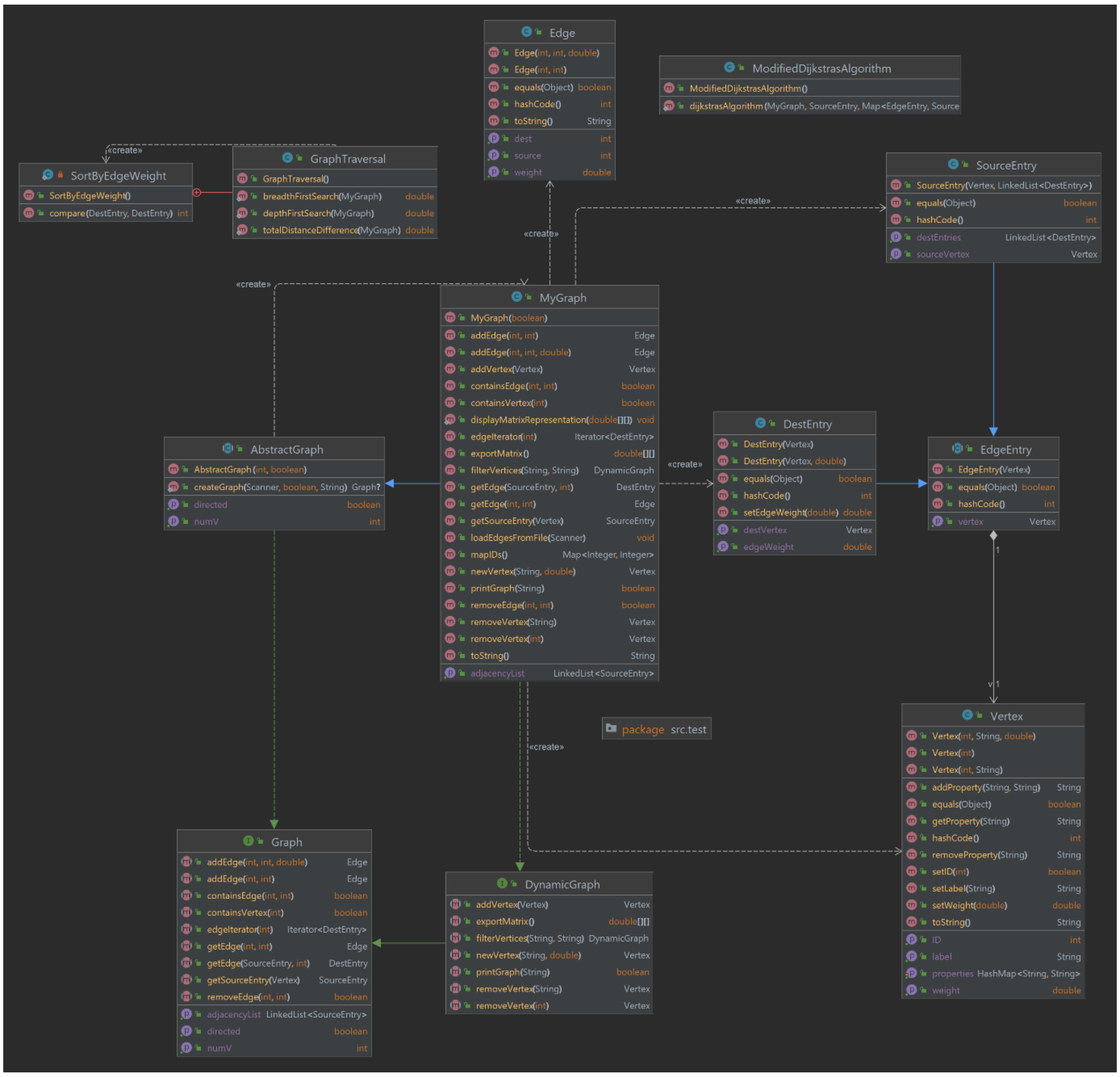
Q2. Total Distance Difference

- A method that takes a MyGraph object as a parameter and performs BFS and DFS traversals.
- The method calculates the total distance of the path for accessing each vertex during the traversal, and it returns the difference between the total distances of two traversal methods.
- During the BFS, if there are more than one alternative to access a vertex at a specific level, the shortest alternative should be considered.
- During the DFS, the vertices should be considered in distance order. So, from a vertex v, DFS should continue with a vertex w which has the smallest edge from v, among all adjacent vertices of v.

Q3. Modified Dijkstra Algorithm

- A method that takes a MyGraph object and a vertex as a parameter to perform a modified version of Dijkstra’s Algorithm.
- In this modified version, the algorithm considers boosting value of the vertices in addition to the edge weights.
- The boosting property is a user defined property that takes double values. The boosting values are subtracted from the total length of paths that they are contained in. The boosting properties of the head and tail vertices in a path aren’t considered by the algorithm.

2. CLASS DIAGRAMS



Edge
Edge(int, int, double)
Edge(int, int)
equals(Object) boolean
hashCode() int
toString() String
dest int
source int
weight double
ModifiedDijkstrasAlgorithm
ModifiedDijkstrasAlgorithm()
dijkstrasAlgorithm(MyGraph, SourceEntry, Map<EdgeEntry, Source
«create»
SortByEdgeWeight
SortByEdgeWeight()
compare(DestEntry, DestEntry) int
GraphTraversal
GraphTraversal()
breadthFirstSearch(MyGraph) double
depthFirstSearch(MyGraph) double
totalDistanceDifference(MyGraph) double
SourceEntry
SourceEntry(Vertex, LinkedList<DestEntry>)
equals(Object) boolean
hashCode() int
destEntries LinkedList<DestEntry>
sourceVertex Vertex
«create»
«create»
«create»
MyGraph
MyGraph(boolean)
addEdge(int, int) Edge
addEdge(int, int, double) Edge
addVertex(Vertex) Vertex
containsEdge(int, int) boolean
containsVertex(int) boolean
displayMatrixRepresentation(double[][]) void
edgeIterator(int) Iterator<DestEntry>
exportMatrix() double[][]
filterVertices(String, String) DynamicGraph
getEdge(SourceEntry, int) DestEntry
getEdge(int, int) Edge
getSourceEntry(Vertex) SourceEntry
loadEdgesFromFile(Scanner) void
mapIDs() Map<Integer, Integer>
newVertex(String, double) Vertex
printGraph(String) boolean
removeEdge(int, int) boolean
removeVertex(String) Vertex
removeVertex(int) Vertex
toString() String
adjacencyList LinkedList<SourceEntry>
DestEntry
DestEntry(Vertex)
DestEntry(Vertex, double)
equals(Object) boolean
hashCode() int
setEdgeWeight(double) double
destVertex Vertex
edgeWeight double
«create»
EdgeEntry
EdgeEntry(Vertex)
equals(Object) boolean
hashCode() int
vertex Vertex
1
v 1
AbstractGraph
AbstractGraph(int, boolean)
createGraph(Scanner, boolean, String) Graph?
directed boolean
numV int
package src.test
«create»
Vertex
Vertex(int, String, double)
Vertex(int)
Vertex(int, String)
addProperty(String, String) String
equals(Object) boolean
getProperty(String) String
hashCode() int
removeProperty(String) String
setID(int) boolean
setLabel(String) String
setWeight(double) double
toString() String
ID int
label String
properties HashMap<String, String>
weight double
Graph
addEdge(int, int, double) Edge
addEdge(int, int) Edge
containsEdge(int, int) boolean
containsVertex(int) boolean
edgeIterator(int) Iterator<DestEntry>
getEdge(int, int) Edge
getEdge(SourceEntry, int) DestEntry
getSourceEntry(Vertex) SourceEntry
removeEdge(int, int) boolean
adjacencyList LinkedList<SourceEntry>
directed boolean
numV int
DynamicGraph
addVertex(Vertex) Vertex
exportMatrix() double[][]
filterVertices(String, String) DynamicGraph
newVertex(String, double) Vertex
printGraph(String) boolean
removeVertex(String) Vertex
removeVertex(int) Vertex

3. PROBLEM SOLUTION APPROACH

Q1. DynamicGraph & MyGraph

The class Vertex is implemented and used inside the MyGraph class to represent the vertices inside the graph data structure. It contains the fields ID, label, weight, and properties. Properties are user-defined additional information about the vertex. Hash-map data structure is used to keep the properties. To add/remove/modify the properties for vertex, required methods are implemented.

MyGraph is an implementation of DynamicGraph interface. The class uses adjacency list representation to handle the edges between the vertices. To make the class dynamic, the adjacency list implemented by using linked list data structure. More specifically linked list consists of SourceEntry objects. SourceEntry class keeps a vertex which is used as source vertex of the edge and a linked list of objects DestEntry class. DestEntry class keeps a vertex which is used as destination vertex of the edge and the other component is double value called weight to keep the weight of the edge. So, to represent an edge the classes, SourceEntry and DestEntry classes are used. With this implementation adjacency list can grow dynamically with the help of link list data structure. The downside of using linked list versus array is the loss of constant access to vertices. However, the downside of array becomes much more especially for removing vertex (shifting the vertices). That's because linked list data structure is selected to implement adjacency list structure.

MyGraph can be created by static method createGraph which is used loadEdges method. To get proper execution from loadEdges method, the file should contain a series of lines, each line with two or three data values. The first is the source, the second is the destination, and the optional third is the weight. The same format is used in the printGraph method to print the graph a file. Also, the graph can be exported in matrix format with the exportMatrix method. The method ExportMatrix first maps the ID of vertices as they become 0 to n – 1 (n is the number of vertices). With this way each vertex can be represented in a matrix. The matrix contains double values to represent the weigh of the edge between vertices.

To filter the graph based on specific property, extra functionality of Vertex class is used. The existence of a specific property can be checked with the containsProperty method of Vertex class. MyGraph class handle the filtering process first by creating a new MyGraph object as subgraph of current graph. Then includes the vertices that have the desired property and the edges which between those edges.

- Execution time complexity of the methods that overridden DynamicGraph interface.
  n: number of vertices
  m: number of edges ($m < n^2$)

Method	Time Complexity	Explanation
newVertex	Θ (n)	A new vertex should have a unique ID. To satisfy this property all the existing IDs should be checked.
addVertex	O(n)	To determine the existence of a vertex, all the vertices should be check.
addEdge	O(n + m)	First the source vertex is found, and its edges should be checked before make modification or new insertion to the list.
removeEdge	O(n + m)	The source vertex is found, and the target edges removed from the list.
removeVertex	O(n)	Target vertex can be found in first or last time.
filterVertices	O(n + m)	First the vertices that have the desired property selected and included in the new graph which is done in Θ(n) time complexity. Insertion of each edge takes constant time and all the edges between those vertices are inserted in O(m) time complexity.
exportMatrix	$\Theta(n^2)$	First the current IDs should be mapped from 0 to n – 1 which is done in Θ(n) time complexity. Then the matrix initialized with infinity which is done in $\Theta(n^2)$ time complexity. After initialization is done all the edges marked with their weights in the matrix which takes Θ(m) time complexity. So overall algorithm takes $\Theta(n^2)$ time complexity.
printGraph	Θ (m)	All the edges should be printed.

Q2. Total Distance Difference

Total distance difference algorithm is uses two different methods breadthFirstSearch and DepthFirstSearch methods which are returns the total distance for accessing each vertex during the traversals. After getting the total distances of two traversal it returns the absolute value of the difference.

Total distance during BFS:

First declare the set identified to indicate a vertex is identified or not. Declare the map distances to record the distance for the specific vertex. With these two data structures, identification of vertices and their distance can be found easily. Finally declare a queue to keep the identified vertices. Start with first vertex at the adjacency list, set it identified and insert to the queue. After initializations are done, do the followings till the queue becomes empty. Poll a vertex current from the queue. Check its all the neighbors whether they are identified or not. At the same time check the distance for the neighbor vertex. If the distance on current path is smaller than its pre-calculated distance, then update the distance as current distance plus weight of the edge between current to neighbor. After finishing the BFS algorithm, sum all the distance for vertices to calculate the total distance.

Total distance during DFS:

Overall algorithm is applied by two methods. First one is the public one for initialization of required data structures and the second one which is private and recursive method, calculates the total distance of the graph by taking a start vertex and a double value as start distance for start vertex. In first method declare the set visited to determine whether a vertex visited or not. Call the second method for all the non-visited vertices by specifying start vertex and 0.0 as start distance. Sum all the return values and return the result as total distance for accessing each vertex during dept-first search.

In second method, put all the non-visited neighbors into a priority-queue based on minimum distance between start and neighbor. After determining non-visited neighbors in minimum distance priority, call the second method recursively for all non-visited neighbors by popping a vertex from the priority-queue. The recursive call is made by specifying neighbor as start vertex parameter and sum of start distance and the distance between start and neighbor as distance parameter. With this recursive call, algorithm always selects first the closest non-visited neighbor during depth-first search. Add all the return values comes from the recursive calls and return the total distance from start vertex to all the other connected vertices.

Q3. Modified Dijkstra Algorithm

The modified version is very similar to original algorithm. The original algorithm calculates the minimum distance by summing the weight of the edges on the path. However modified version also includes the boosting property of vertices. So, the boosting value of all the vertices except the head and tail vertices on the path are subtracted during the calculation of minimum distance.

The modified algorithm uses the map dist to keep the minimum distance of the vertices, the map pred to keep the predecessor/parent of the vertices and the set notFixed to keep track on the vertices which their minimum distance is not determined yet. Initially dist filled with the weight of edge between the start vertex which is provided by the user and current vertex. If there is no edge, then the distance becomes infinity. The map pred initialize by setting start as parent of all the vertices except the start itself. The set notFixed contains all the vertices except the start index. After initialization process is done, the algorithm repeats the followings till the set notFixed become empty which means minimum distance of all the vertices are found. Inside the loop first select the vertex which has minimum distance among the notFixed vertices. Set the selected vertex current as fixed by removing it from the set notFixed. Check the distances for neighbors of current by comparing the distance on the path of current with the existing distance of the neighbor. The distance on current is distance of current plus the distance between current and neighbor and minus the boosting value of current if current is not head vertex of the path. Remember algorithm follows the rule except head and tail vertices boosting value of the vertices should be subtracted from the distance. Whether a vertex is head or not can be easily found by checking its predecessor is null or not. If the distance on current is shorter than the existing distance of neighbor than update the distance of neighbor and set current as predecessor of neighbor to indicate neighbor follows the path of current. After finding minimum distance for all the vertices the loop is terminated and the two map dist and pred keep all the required information about the Dijkstra's algorithm.

4. TEST CASES

Q1 – DynamicGraph & MyGraph						
Test ID	Test Case	Test Steps	Test Data	Expected Results	Actual Results	Pass/Fail
T1	Add vertex	Try to add unique and duplicated vertices.	Set of vertices with their IDs. dataset: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 1, 5, 12, 13, 4}	All the vertices inserted. For the duplicated ones, most recent insertion replaces the previous one for a unique ID.	As Expected	**PASS**

T2	Add edge	Try to add edges by specifying the source and dest vertices. Use both valid and invalid vertex IDs for addEdge method.	Two parallel set for the source and destination vertex of the edge.
			
			source: {0, 15, 4, 9, 5, 1, 4, 12, 3, 7, 8, -1, 9, 7}
			
			dest = {1, 6, 3, 12, 0, 1, 4, 32, 2, 14, 3, 1, 5, 13}
				All the edges except the duplicated and invalid ones are inserted to the graph.	As Expected	**PASS**

T3	Remove vertex	Try to remove both an existing and non-existing vertices.	dataset: {1, 7, 5, 4, 14};	All the specified existing vertices removed with their edges.	As Expected	**PASS**

T4	Remove edge	Try to remove both an existing and non-existing edges.	source: {0, 15, 4, 9, 5, 1, 4, 12, 3, 7, 8, -1, 9, 7}
			
			dest = {1, 6, 3, 12, 0, 1, 4, 32, 2, 14, 3, 1, 5, 13}
				All the given valid edges are removed.	As Expected	**PASS**

T5	Contains edge/vertex	1. Import the graph. 2. Call contains edge/vertex methods.	File s1.txt	Checks whether edges or vertices exist in the graph.	As Expected	**PASS**
T6	Load edges from a file	1. Create an input file which contains the edges. 2. Call the CreateGraph method with the created file.	File s1.txt	All the edges and the vertices are inserted into the graph.	As Expected	**PASS**
T7	Print the graph	Call the method printGraph to print the graph in adjacency list format.	Randomly created edges	Current graph printed into the given file.	As Expected	**PASS**
T8	Filter vertices	1. Add specific properties such as color and charge to edges. 2. Filter the edges that has same key-value for filter property.	Randomly created edges on vertices from 0 to 14.	Creates a subgraph which contains the filtered vertices with their edges.	As Expected	**PASS**
T9	Export matrix representation	1. Add some weighted edges. 2. Call exportMatrix method.	File s1.txt	Returns an 2D double array which contains the edges inside the graph.	As Expected	**PASS**
T10	City map which shows the distances between the cities.	1. Import directed and weighted graph. The vertices have label. 2. Print adjacency list and adjacency matrix representation of the graph.	File s6.txt	Creates the graph and prints adjacency list and adjacency matrix representation of it.	As Expected	**PASS**

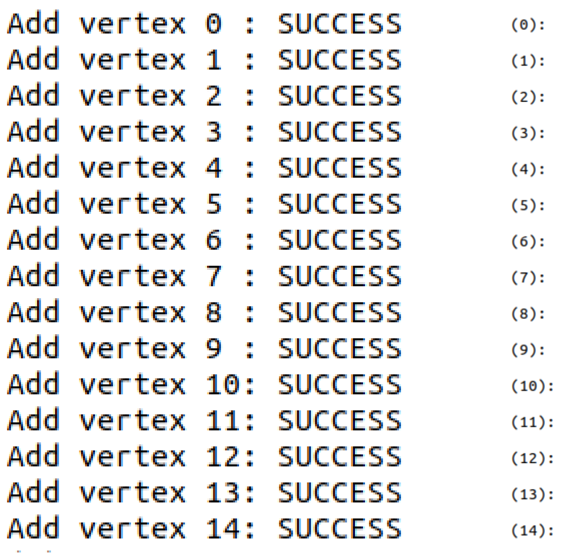
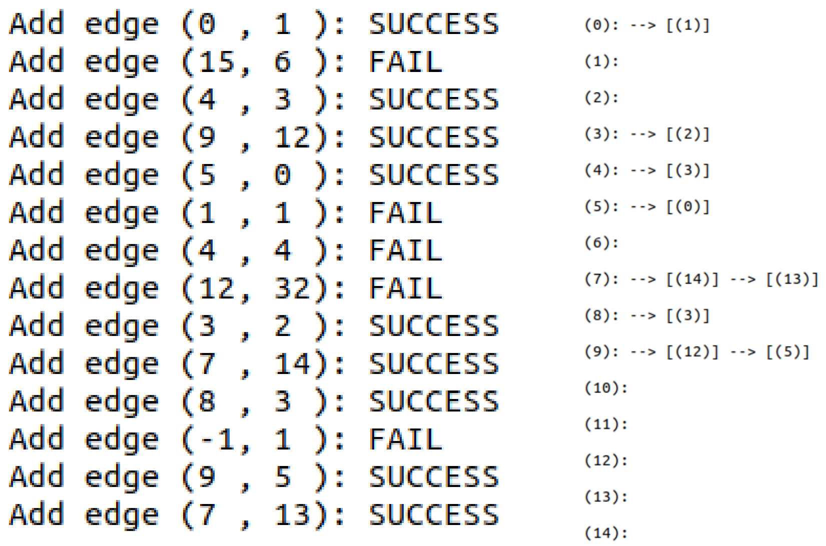
Q2 – Total Distance Difference

Test ID	Test Case	Test Steps	Test Data	Expected Results	Actual Results	Pass/Fail
T1	Calculate total distance of directed graph	1. Create the undirected graph by loading the edges from the input file. 2. Calculate the total distance.	File s3.txt	Total distances: BFS: 150.0 DFS: 170.0 Difference: 20.0	As Expected	**PASS**
T2	Calculate total distance of undirected graph	1. Create the directed graph by loading the edges from the input file. 2. Calculate the total distance.	File s4.txt	Total distances: BFS: 130.0 DFS: 165.0 Difference: 35.0	As Expected	**PASS**

Q3 – Modified Dijkstra Algorithm

Test ID	Test Case	Test Steps	Test Data	Expected Results	Actual Results	Pass/Fail
T1	Apply modified Dijkstra algorithm on an undirected graph	1. Create the graph by loading the edges from the input file. 2. Add boosting values to the vertices.	File s3.txt	The predecessor of each vertex and their shortest distance is returned.	As Expected	**PASS**
T2	Apply modified Dijkstra algorithm on a directed graph	1. Create the graph by loading the edges from the input file. 2. Add boosting values to the vertices.	File s6.txt	The predecessor of each vertex and their shortest distance is returned.	As Expected	**PASS**
T3	Random boosting values	1. Create the graph by loading the edges from the input file. 2. Add random boosting values to the vertices.	File s6.txt	The predecessor of each vertex and their shortest distance is returned.	As Expected	**PASS**

5. RUNNING AND RESULTS

Q1 - DynamicGraph & MyGraph	
Test ID	Test Result
T1	
T2	

```
Add vertex 0 : SUCCESS          (0):
Add vertex 1 : SUCCESS          (1):
Add vertex 2 : SUCCESS          (2):
Add vertex 3 : SUCCESS          (3):
Add vertex 4 : SUCCESS          (4):
Add vertex 5 : SUCCESS          (5):
Add vertex 6 : SUCCESS          (6):
Add vertex 7 : SUCCESS          (7):
Add vertex 8 : SUCCESS          (8):
Add vertex 9 : SUCCESS          (9):
Add vertex 10: SUCCESS          (10):
Add vertex 11: SUCCESS          (11):
Add vertex 12: SUCCESS          (12):
Add vertex 13: SUCCESS          (13):
Add vertex 14: SUCCESS          (14):
```

```
Add edge (0 , 1 ): SUCCESS      (0): --> [(1)]
Add edge (15, 6 ): FAIL         (1):
Add edge (4 , 3 ): SUCCESS      (2):
Add edge (9 , 12): SUCCESS      (3): --> [(2)]
Add edge (5 , 0 ): SUCCESS      (4): --> [(3)]
Add edge (1 , 1 ): FAIL         (5): --> [(0)]
Add edge (4 , 4 ): FAIL         (6):
Add edge (12, 32): FAIL         (7): --> [(14)] --> [(13)]
Add edge (3 , 2 ): SUCCESS      (8): --> [(3)]
Add edge (7 , 14): SUCCESS      (9): --> [(12)] --> [(5)]
Add edge (8 , 3 ): SUCCESS      (10):
Add edge (-1, 1 ): FAIL         (11):
Add edge (9 , 5 ): SUCCESS      (12):
Add edge (7 , 13): SUCCESS      (13):
                                (14):
```

T3

```
Remove vertex 1 : SUCCESS
Remove vertex 7 : SUCCESS
Remove vertex 5 : SUCCESS
Remove vertex 4 : SUCCESS
Remove vertex 14: SUCCESS
Remove vertex 21: FAIL
Remove vertex 43: FAIL
```

```
(0):
(2):
(3): --> [(2)]
(6):
(8): --> [(3)]
(9): --> [(12)]
(10):
(11):
(12):
(13):
```

T4

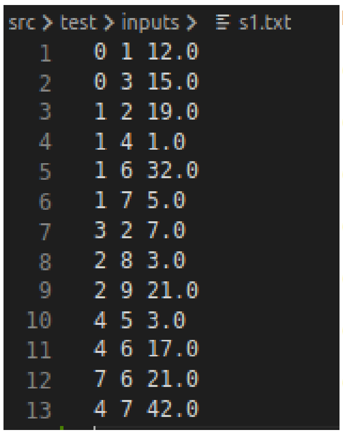
```
Remove edge (0 , 1 ): FAIL
Remove edge (15, 6 ): FAIL
Remove edge (4 , 3 ): FAIL
Remove edge (9 , 12): SUCCESS
Remove edge (5 , 0 ): FAIL
Remove edge (1 , 1 ): FAIL
Remove edge (4 , 4 ): FAIL
Remove edge (12, 32): FAIL
Remove edge (3 , 2 ): SUCCESS
Remove edge (7 , 14): FAIL
Remove edge (8 , 3 ): SUCCESS
Remove edge (-1, 1 ): FAIL
Remove edge (9 , 5 ): FAIL
Remove edge (7 , 13): FAIL
```

```
(0):
(2):
(3):
(6):
(8):
(9):
(10):
(11):
(12):
(13):
```

T5

```
Edges are loaded

(0): --> [(1) | 21.0] --> [(2) | 12.0] --> [(3) | 8.0] --> [(4) | 31.0]

(1): --> [(3) | 15.0] --> [(4) | 34.0]

(2): --> [(5) | 25.0] --> [(6) | 12.0]

(3): --> [(4) | 19.0]

(4):

(5): --> [(6) | 50.0]

(6):


containsEdge (1 , 2 ): false
containsEdge (5 , 6 ): true
containsEdge (3 , 11): false
containsVertex (13): false
containsVertex (3 ): true
containsVertex (34): false
containsVertex (5 ): true
```

T6

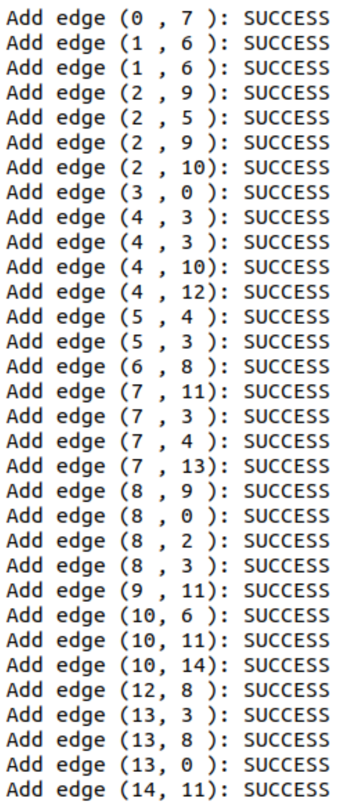


```
src > test > inputs > s1.txt
0 1 12.0
0 3 15.0
1 2 19.0
1 4 1.0
1 6 32.0
1 7 5.0
3 2 7.0
2 8 3.0
2 9 21.0
4 5 3.0
4 6 17.0
7 6 21.0
4 7 42.0
```

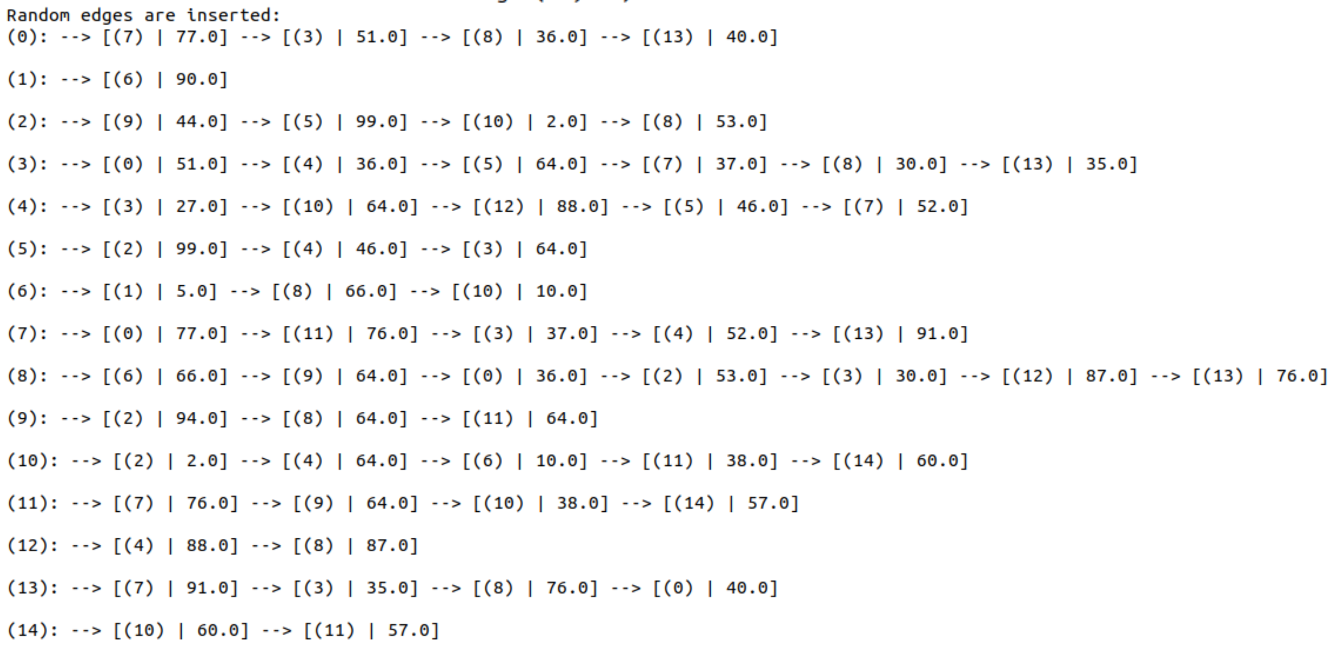
```
Edges are loaded

(0): --> [(1) | 21.0] --> [(2) | 12.0] --> [(3) | 8.0] --> [(4) | 31.0]

(1): --> [(3) | 15.0] --> [(4) | 34.0]

(2): --> [(5) | 25.0] --> [(6) | 12.0]

(3): --> [(4) | 19.0]

(4):

(5): --> [(6) | 50.0]

(6):
```

(Input file s1.txt and created graph)

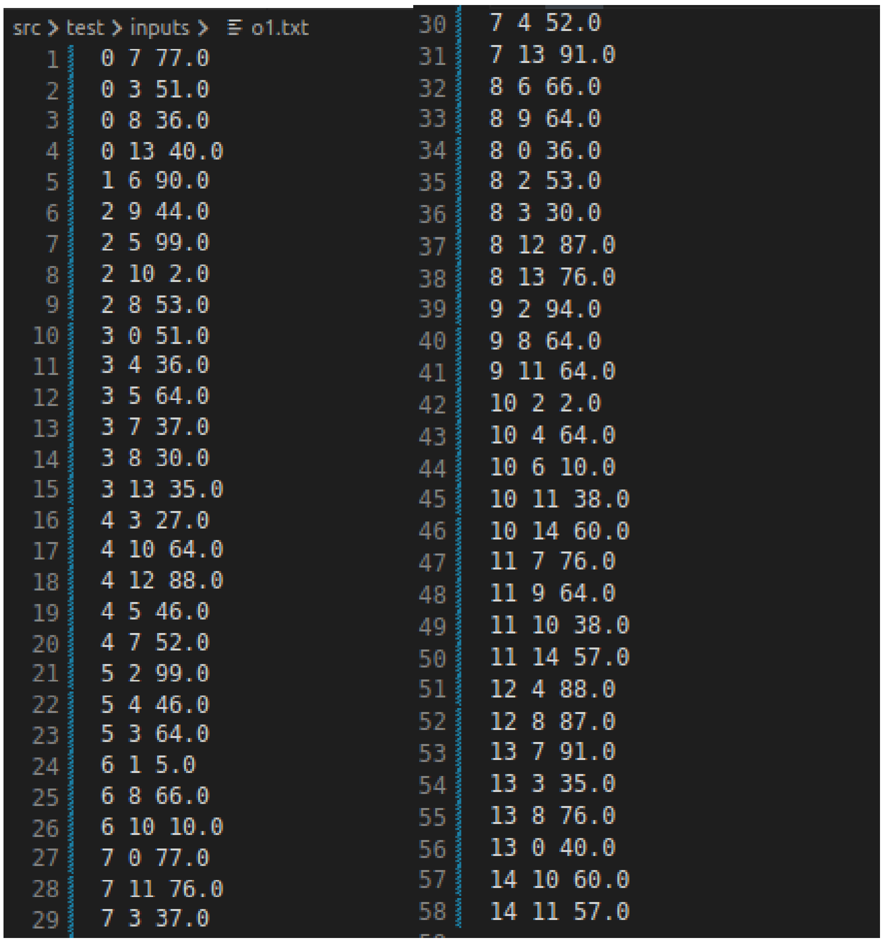
T7



```
Add edge (0 , 7 ): SUCCESS
Add edge (1 , 6 ): SUCCESS
Add edge (1 , 6 ): SUCCESS
Add edge (2 , 9 ): SUCCESS
Add edge (2 , 5 ): SUCCESS
Add edge (2 , 9 ): SUCCESS
Add edge (2 , 10): SUCCESS
Add edge (3 , 0 ): SUCCESS
Add edge (4 , 3 ): SUCCESS
Add edge (4 , 3 ): SUCCESS
Add edge (4 , 10): SUCCESS
Add edge (4 , 12): SUCCESS
Add edge (5 , 4 ): SUCCESS
Add edge (5 , 3 ): SUCCESS
Add edge (6 , 8 ): SUCCESS
Add edge (7 , 11): SUCCESS
Add edge (7 , 3 ): SUCCESS
Add edge (7 , 4 ): SUCCESS
Add edge (7 , 13): SUCCESS
Add edge (8 , 9 ): SUCCESS
Add edge (8 , 0 ): SUCCESS
Add edge (8 , 2 ): SUCCESS
Add edge (8 , 3 ): SUCCESS
Add edge (9 , 11): SUCCESS
Add edge (10, 6 ): SUCCESS
Add edge (10, 11): SUCCESS
Add edge (10, 14): SUCCESS
Add edge (12, 8 ): SUCCESS
Add edge (13, 3 ): SUCCESS
Add edge (13, 8 ): SUCCESS
Add edge (13, 0 ): SUCCESS
Add edge (14, 11): SUCCESS
```



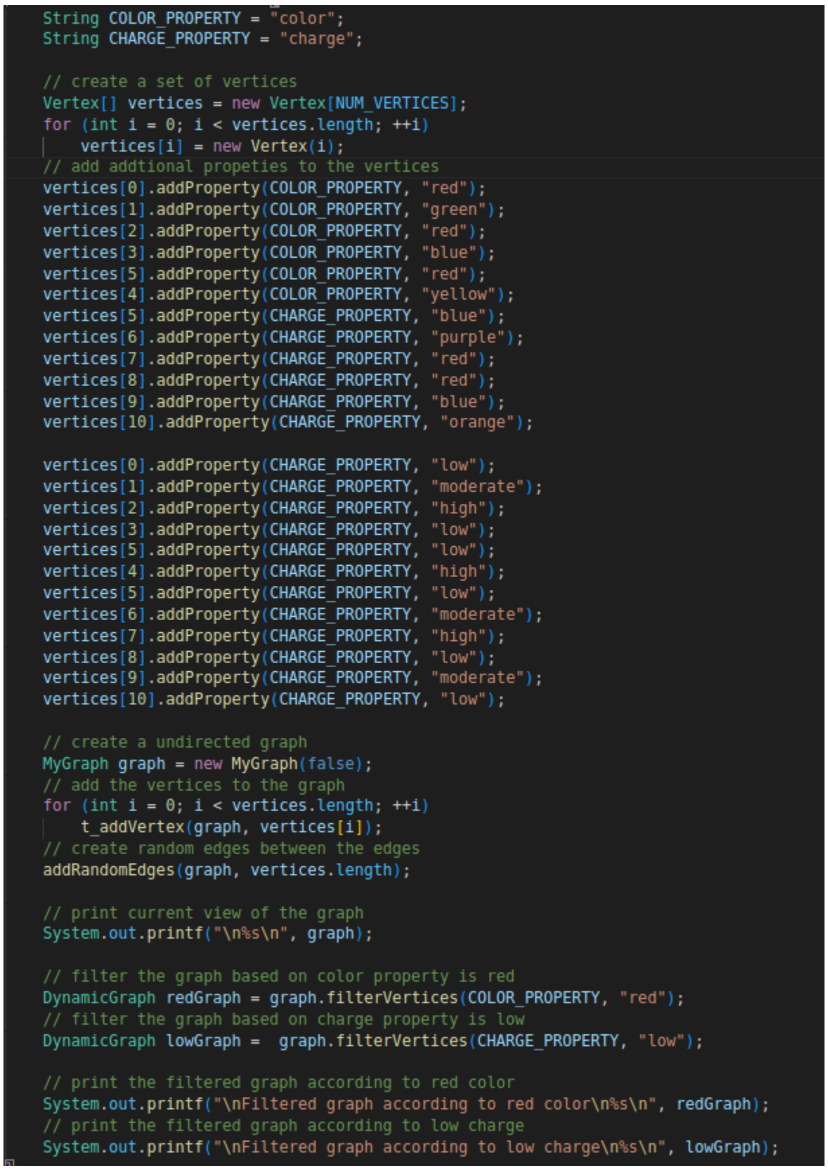
```
Random edges are inserted:
(0): --> [(7) | 77.0] --> [(3) | 51.0] --> [(8) | 36.0] --> [(13) | 40.0]

(1): --> [(6) | 90.0]

(2): --> [(9) | 44.0] --> [(5) | 99.0] --> [(10) | 2.0] --> [(8) | 53.0]

(3): --> [(0) | 51.0] --> [(4) | 36.0] --> [(5) | 64.0] --> [(7) | 37.0] --> [(8) | 30.0] --> [(13) | 35.0]

(4): --> [(3) | 27.0] --> [(10) | 64.0] --> [(12) | 88.0] --> [(5) | 46.0] --> [(7) | 52.0]

(5): --> [(2) | 99.0] --> [(4) | 46.0] --> [(3) | 64.0]

(6): --> [(1) | 5.0] --> [(8) | 66.0] --> [(10) | 10.0]

(7): --> [(0) | 77.0] --> [(11) | 76.0] --> [(3) | 37.0] --> [(4) | 52.0] --> [(13) | 91.0]

(8): --> [(6) | 66.0] --> [(9) | 64.0] --> [(0) | 36.0] --> [(2) | 53.0] --> [(3) | 30.0] --> [(12) | 87.0] --> [(13) | 76.0]

(9): --> [(2) | 94.0] --> [(8) | 64.0] --> [(11) | 64.0]

(10): --> [(2) | 2.0] --> [(4) | 64.0] --> [(6) | 10.0] --> [(11) | 38.0] --> [(14) | 60.0]

(11): --> [(7) | 76.0] --> [(9) | 64.0] --> [(10) | 38.0] --> [(14) | 57.0]

(12): --> [(4) | 88.0] --> [(8) | 87.0]

(13): --> [(7) | 91.0] --> [(3) | 35.0] --> [(8) | 76.0] --> [(0) | 40.0]

(14): --> [(10) | 60.0] --> [(11) | 57.0]
```


src > test > inputs > o1.txt

```
0 7 77.0
0 3 51.0
0 8 36.0
0 13 40.0
1 6 90.0
2 9 44.0
2 5 99.0
2 10 2.0
2 8 53.0
3 0 51.0
3 4 36.0
3 5 64.0
3 7 37.0
3 8 30.0
3 13 35.0
4 3 27.0
4 10 64.0
4 12 88.0
4 5 46.0
4 7 52.0
5 2 99.0
5 4 46.0
5 3 64.0
6 1 5.0
6 8 66.0
6 10 10.0
7 0 77.0
7 11 76.0
7 3 37.0
7 4 52.0
7 13 91.0
8 6 66.0
8 9 64.0
8 0 36.0
8 2 53.0
8 3 30.0
8 12 87.0
8 13 76.0
9 2 94.0
9 8 64.0
9 11 64.0
10 2 2.0
10 4 64.0
10 6 10.0
10 11 38.0
10 14 60.0
11 7 76.0
11 9 64.0
11 10 38.0
11 14 57.0
12 4 88.0
12 8 87.0
13 7 91.0
13 3 35.0
13 8 76.0
13 0 40.0
14 10 60.0
14 11 57.0
```

(printGraph output file: o1.txt)

T8

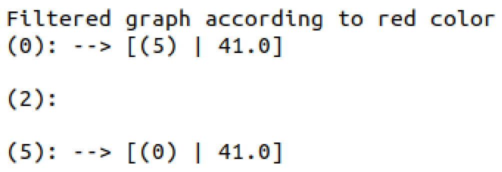


```java
String COLOR_PROPERTY = "color";
String CHARGE_PROPERTY = "charge";

// create a set of vertices
Vertex[] vertices = new Vertex[NUM_VERTICES];
for (int i = 0; i < vertices.length; ++i)
    vertices[i] = new Vertex(i);
// add addtional propeties to the vertices
vertices[0].addProperty(COLOR_PROPERTY, "red");
vertices[1].addProperty(COLOR_PROPERTY, "green");
vertices[2].addProperty(COLOR_PROPERTY, "red");
vertices[3].addProperty(COLOR_PROPERTY, "blue");
vertices[5].addProperty(COLOR_PROPERTY, "red");
vertices[4].addProperty(COLOR_PROPERTY, "yellow");
vertices[5].addProperty(CHARGE_PROPERTY, "blue");
vertices[6].addProperty(CHARGE_PROPERTY, "purple");
vertices[7].addProperty(CHARGE_PROPERTY, "red");
vertices[8].addProperty(CHARGE_PROPERTY, "red");
vertices[9].addProperty(CHARGE_PROPERTY, "blue");
vertices[10].addProperty(CHARGE_PROPERTY, "orange");

vertices[0].addProperty(CHARGE_PROPERTY, "low");
vertices[1].addProperty(CHARGE_PROPERTY, "moderate");
vertices[2].addProperty(CHARGE_PROPERTY, "high");
vertices[3].addProperty(CHARGE_PROPERTY, "low");
vertices[5].addProperty(CHARGE_PROPERTY, "low");
vertices[4].addProperty(CHARGE_PROPERTY, "high");
vertices[5].addProperty(CHARGE_PROPERTY, "low");
vertices[6].addProperty(CHARGE_PROPERTY, "moderate");
vertices[7].addProperty(CHARGE_PROPERTY, "high");
vertices[8].addProperty(CHARGE_PROPERTY, "low");
vertices[9].addProperty(CHARGE_PROPERTY, "moderate");
vertices[10].addProperty(CHARGE_PROPERTY, "low");

// create a undirected graph
MyGraph graph = new MyGraph(false);
// add the vertices to the graph
for (int i = 0; i < vertices.length; ++i)
    t_addVertex(graph, vertices[i]);
// create random edges between the edges
addRandomEdges(graph, vertices.length);

// print current view of the graph
System.out.printf("\n%s\n", graph);

// filter the graph based on color property is red
DynamicGraph redGraph = graph.filterVertices(COLOR_PROPERTY, "red");
// filter the graph based on charge property is low
DynamicGraph lowGraph =  graph.filterVertices(CHARGE_PROPERTY, "low");

// print the filtered graph according to red color
System.out.printf("\nFiltered graph according to red color\n%s\n", redGraph);
// print the filtered graph according to low charge
System.out.printf("\nFiltered graph according to low charge\n%s\n", lowGraph);
```

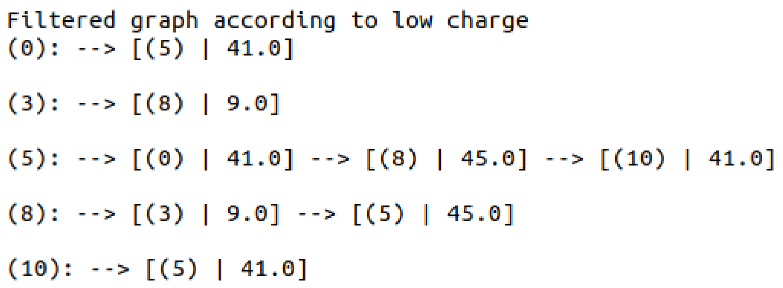
(Adding properties to vertices)

```
(0): --> [(6) | 93.0] --> [(5) | 41.0] --> [(7) | 55.0] --> [(9) | 95.0] --> [(12) | 9.0]

(1): --> [(4) | 85.0] --> [(5) | 84.0] --> [(7) | 6.0]

(2): --> [(8) | 1.0] --> [(3) | 99.0] --> [(6) | 85.0] --> [(10) | 32.0] --> [(11) | 65.0]

(3): --> [(11) | 82.0] --> [(2) | 99.0] --> [(14) | 4.0] --> [(8) | 9.0]

(4): --> [(1) | 85.0] --> [(7) | 52.0] --> [(9) | 18.0]

(5): --> [(1) | 84.0] --> [(14) | 3.0] --> [(0) | 41.0] --> [(8) | 45.0] --> [(10) | 41.0]

(6): --> [(0) | 93.0] --> [(13) | 45.0] --> [(2) | 85.0] --> [(9) | 12.0]

(7): --> [(4) | 52.0] --> [(0) | 55.0] --> [(13) | 38.0] --> [(1) | 6.0] --> [(10) | 46.0] --> [(11) | 79.0]

(8): --> [(2) | 1.0] --> [(3) | 9.0] --> [(5) | 45.0]

(9): --> [(6) | 12.0] --> [(4) | 18.0] --> [(0) | 95.0] --> [(10) | 65.0] --> [(11) | 27.0] --> [(13) | 68.0]

(10): --> [(5) | 41.0] --> [(9) | 65.0] --> [(2) | 32.0] --> [(7) | 46.0]

(11): --> [(3) | 82.0] --> [(2) | 65.0] --> [(9) | 27.0] --> [(7) | 79.0]

(12): --> [(0) | 9.0] --> [(13) | 46.0]

(13): --> [(6) | 45.0] --> [(7) | 8.0] --> [(12) | 46.0] --> [(9) | 14.0]

(14): --> [(3) | 4.0] --> [(5) | 3.0]
```

(Randomly created graph)



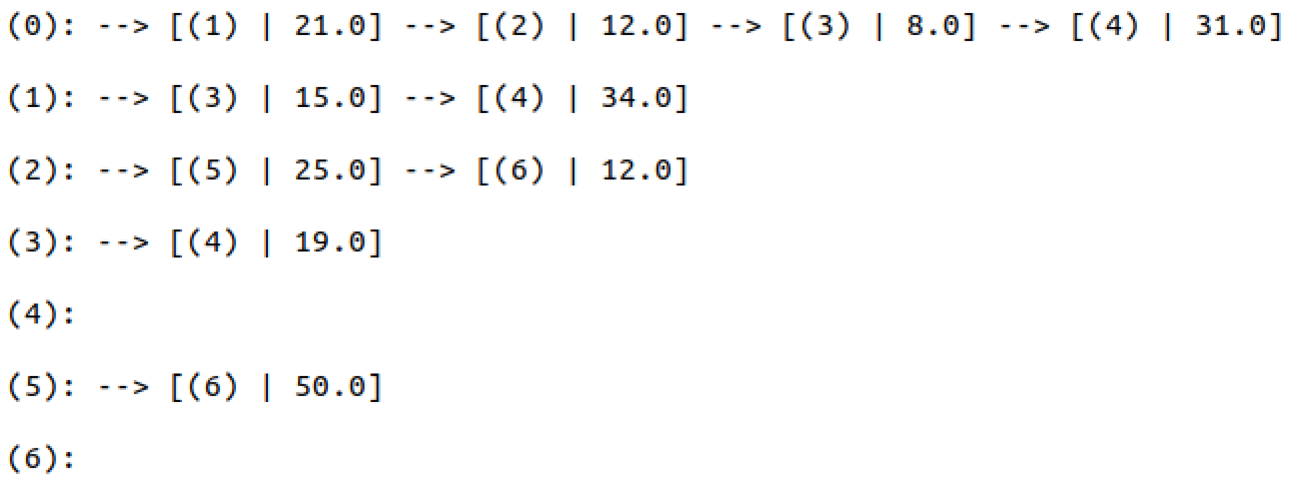
```
Filtered graph according to red color
(0): --> [(5) | 41.0]

(2):

(5): --> [(0) | 41.0]
```



```
Filtered graph according to low charge
(0): --> [(5) | 41.0]

(3): --> [(8) | 9.0]

(5): --> [(0) | 41.0] --> [(8) | 45.0] --> [(10) | 41.0]

(8): --> [(3) | 9.0] --> [(5) | 45.0]

(10): --> [(5) | 41.0]
```

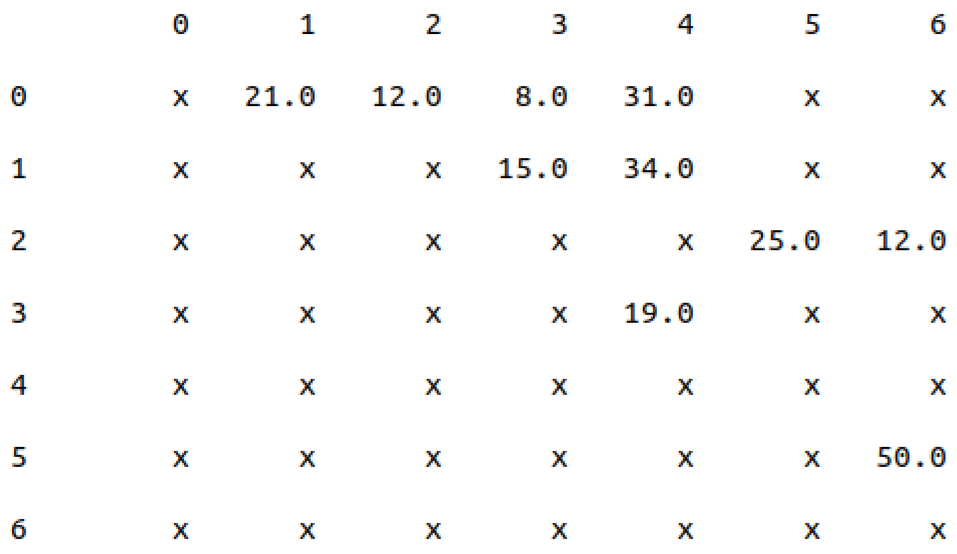
(Filtered subgraphs)

T9



```
(0): --> [(1) | 21.0] --> [(2) | 12.0] --> [(3) | 8.0] --> [(4) | 31.0]

(1): --> [(3) | 15.0] --> [(4) | 34.0]

(2): --> [(5) | 25.0] --> [(6) | 12.0]

(3): --> [(4) | 19.0]

(4):

(5): --> [(6) | 50.0]

(6):
```

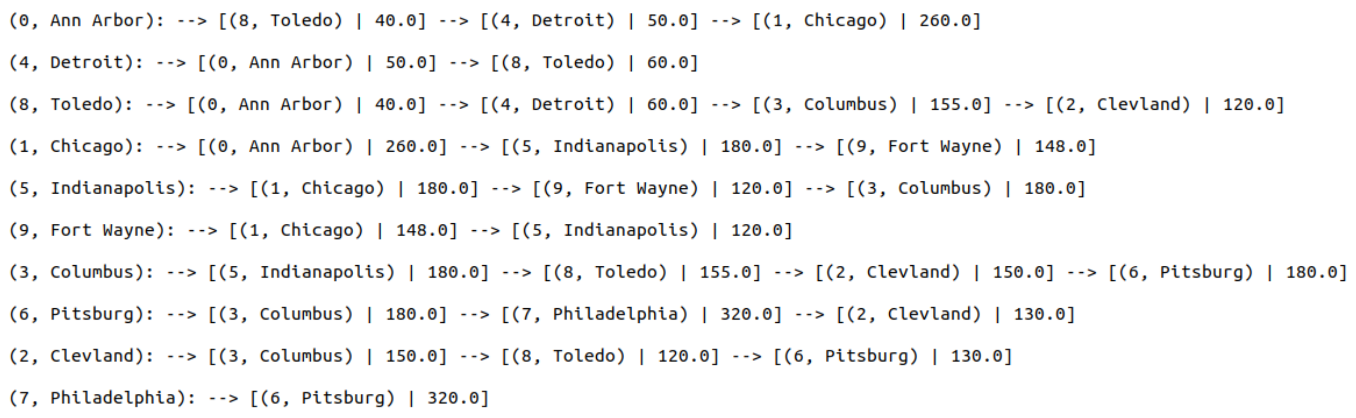
(Adjacency list representation of the graph)



```
              0       1       2       3       4       5       6

 0            x    21.0    12.0     8.0    31.0       x       x

 1            x       x       x    15.0    34.0       x       x

 2            x       x       x       x       x    25.0    12.0

 3            x       x       x       x    19.0       x       x

 4            x       x       x       x       x       x       x

 5            x       x       x       x       x       x    50.0

 6            x       x       x       x       x       x       x
```

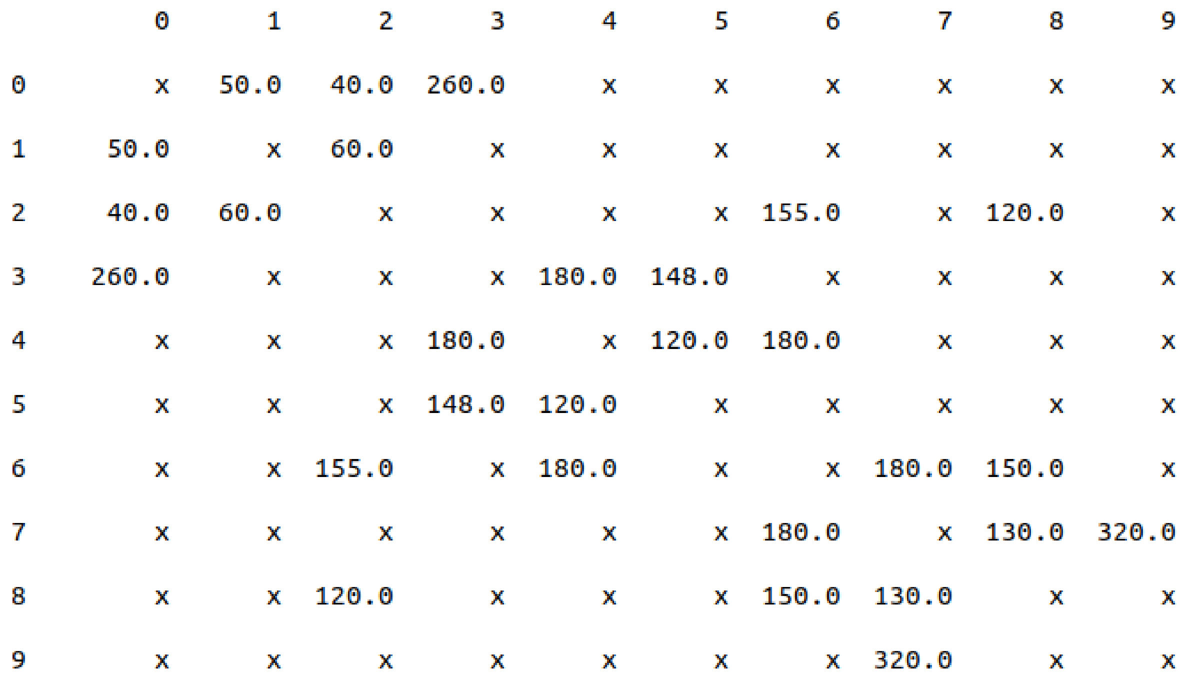
(Adjacency matrix representation of the graph)

T10



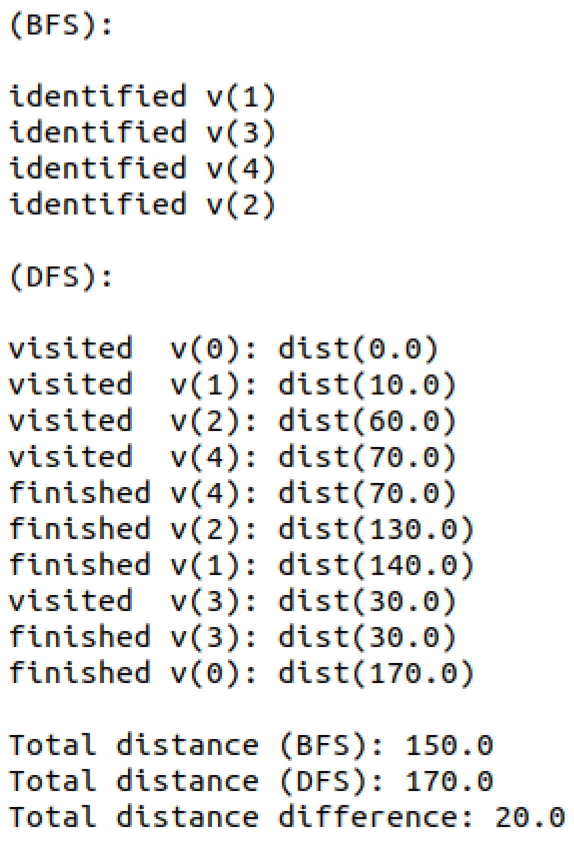
```
(0, Ann Arbor): --> [(8, Toledo) | 40.0] --> [(4, Detroit) | 50.0] --> [(1, Chicago) | 260.0]

(4, Detroit): --> [(0, Ann Arbor) | 50.0] --> [(8, Toledo) | 60.0]

(8, Toledo): --> [(0, Ann Arbor) | 40.0] --> [(4, Detroit) | 60.0] --> [(3, Columbus) | 155.0] --> [(2, Clevland) | 120.0]

(1, Chicago): --> [(0, Ann Arbor) | 260.0] --> [(5, Indianapolis) | 180.0] --> [(9, Fort Wayne) | 148.0]

(5, Indianapolis): --> [(1, Chicago) | 180.0] --> [(9, Fort Wayne) | 120.0] --> [(3, Columbus) | 180.0]

(9, Fort Wayne): --> [(1, Chicago) | 148.0] --> [(5, Indianapolis) | 120.0]

(3, Columbus): --> [(5, Indianapolis) | 180.0] --> [(8, Toledo) | 155.0] --> [(2, Clevland) | 150.0] --> [(6, Pitsburg) | 180.0]

(6, Pitsburg): --> [(3, Columbus) | 180.0] --> [(7, Philadelphia) | 320.0] --> [(2, Clevland) | 130.0]

(2, Clevland): --> [(3, Columbus) | 150.0] --> [(8, Toledo) | 120.0] --> [(6, Pitsburg) | 130.0]

(7, Philadelphia): --> [(6, Pitsburg) | 320.0]
```

(Cities and the distances between them shown in adjacency list representation)



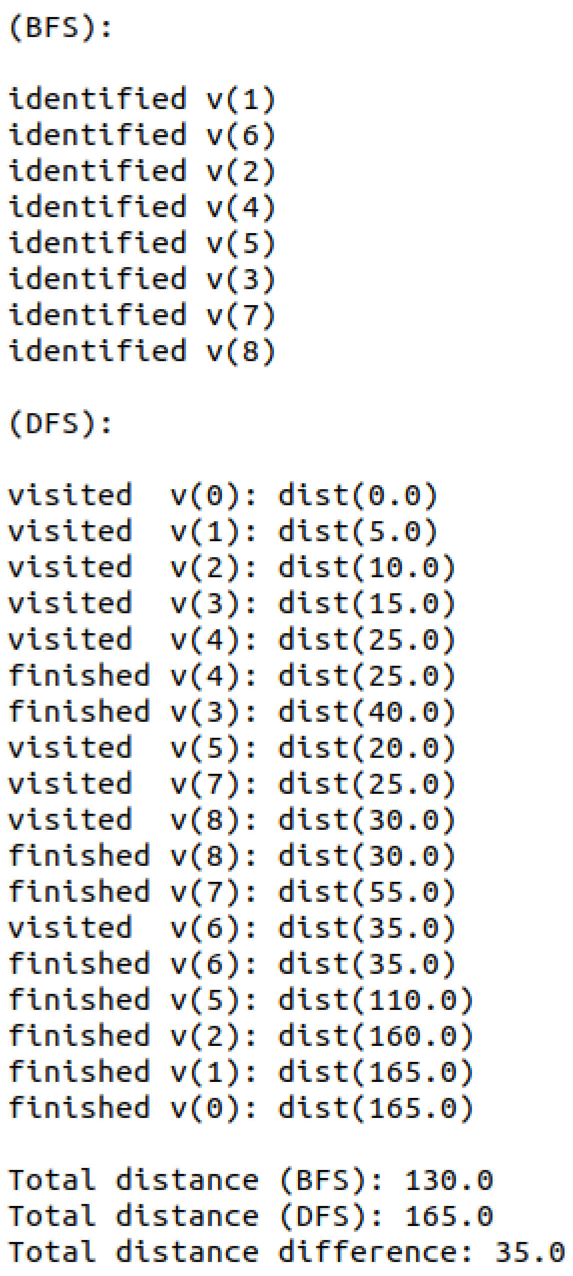
```
            0      1      2      3      4      5      6      7      8      9

   0        x   50.0   40.0  260.0      x      x      x      x      x      x

   1     50.0      x   60.0      x      x      x      x      x      x      x

   2     40.0   60.0      x      x      x      x  155.0      x  120.0      x

   3    260.0      x      x      x  180.0  148.0      x      x      x      x

   4        x      x      x  180.0      x  120.0  180.0      x      x      x

   5        x      x      x  148.0  120.0      x      x      x      x      x

   6        x      x  155.0      x  180.0      x      x  180.0  150.0      x

   7        x      x      x      x      x      x  180.0      x  130.0  320.0

   8        x      x  120.0      x      x      x  150.0  130.0      x      x

   9        x      x      x      x      x      x      x  320.0      x      x
```

(Cities with their hashed ID and the distances between them shown in adjacency matrix representation)

Q2 - Total Distance Difference	
Test ID	Test Result
T1	

```
(BFS):

identified v(1)
identified v(3)
identified v(4)
identified v(2)

(DFS):

visited  v(0): dist(0.0)
visited  v(1): dist(10.0)
visited  v(2): dist(60.0)
visited  v(4): dist(70.0)
finished v(4): dist(70.0)
finished v(2): dist(130.0)
finished v(1): dist(140.0)
visited  v(3): dist(30.0)
finished v(3): dist(30.0)
finished v(0): dist(170.0)

Total distance (BFS): 150.0
Total distance (DFS): 170.0
Total distance difference: 20.0
```

T2



```
(BFS):

identified v(1)
identified v(6)
identified v(2)
identified v(4)
identified v(5)
identified v(3)
identified v(7)
identified v(8)

(DFS):

visited  v(0): dist(0.0)
visited  v(1): dist(5.0)
visited  v(2): dist(10.0)
visited  v(3): dist(15.0)
visited  v(4): dist(25.0)
finished v(4): dist(25.0)
finished v(3): dist(40.0)
visited  v(5): dist(20.0)
visited  v(7): dist(25.0)
visited  v(8): dist(30.0)
finished v(8): dist(30.0)
finished v(7): dist(55.0)
visited  v(6): dist(35.0)
finished v(6): dist(35.0)
finished v(5): dist(110.0)
finished v(2): dist(160.0)
finished v(1): dist(165.0)
finished v(0): dist(165.0)

Total distance (BFS): 130.0
Total distance (DFS): 165.0
Total distance difference: 35.0
```

Q3 - Modified Dijkstra Algorithm

Test ID | Test Result

T1

```
Fix vertex 1 with shortest distance 10.0

        Check the neighbor vertex 2
        Remove the boosting value 2.0 from the distance 60.0
        Update the distance of vertex 2 from Infinity to 58.0

Fix vertex 3 with shortest distance 30.0

        Check the neighbor vertex 2
        Remove the boosting value 3.0 from the distance 50.0
        Update the distance of vertex 2 from 58.0 to 47.0

        Check the neighbor vertex 4
        Remove the boosting value 3.0 from the distance 90.0
        Update the distance of vertex 4 from 100.0 to 87.0

Fix vertex 2 with shortest distance 47.0

        Check the neighbor vertex 4
        Remove the boosting value 5.0 from the distance 57.0
        Update the distance of vertex 4 from 87.0 to 52.0

Fix vertex 4 with shortest distance 52.0

Results:
------------------------------
ID      pred    dist

1       0       10.0
3       0       30.0
4       2       52.0
2       3       47.0
```

T2

```
Fix vertex 8 with shortest distance 40.0

        Check the neighbor vertex 0
        Remove the boosting value 20.0 from the distance 80.0

        Check the neighbor vertex 4
        Remove the boosting value 20.0 from the distance 100.0

        Check the neighbor vertex 3
        Remove the boosting value 20.0 from the distance 195.0
        Update the distance of vertex 3 from Infinity to 175.0

        Check the neighbor vertex 2
        Remove the boosting value 20.0 from the distance 160.0
        Update the distance of vertex 2 from Infinity to 140.0

Fix vertex 4 with shortest distance 50.0

        Check the neighbor vertex 0
        Remove the boosting value 30.0 from the distance 100.0

        Check the neighbor vertex 8
        Remove the boosting value 30.0 from the distance 110.0

Fix vertex 2 with shortest distance 140.0

        Check the neighbor vertex 3
        Remove the boosting value 80.0 from the distance 290.0

        Check the neighbor vertex 8
        Remove the boosting value 80.0 from the distance 260.0

        Check the neighbor vertex 6
        Remove the boosting value 80.0 from the distance 270.0
        Update the distance of vertex 6 from Infinity to 190.0
```

```
Fix vertex 3 with shortest distance 175.0

        Check the neighbor vertex 5
        Remove the boosting value 70.0 from the distance 355.0
        Update the distance of vertex 5 from Infinity to 285.0

        Check the neighbor vertex 8
        Remove the boosting value 70.0 from the distance 330.0

        Check the neighbor vertex 2
        Remove the boosting value 70.0 from the distance 325.0

        Check the neighbor vertex 6
        Remove the boosting value 70.0 from the distance 355.0

Fix vertex 6 with shortest distance 190.0

        Check the neighbor vertex 3
        Remove the boosting value 90.0 from the distance 370.0

        Check the neighbor vertex 7
        Remove the boosting value 90.0 from the distance 510.0
        Update the distance of vertex 7 from Infinity to 420.0

        Check the neighbor vertex 2
        Remove the boosting value 90.0 from the distance 320.0

Fix vertex 1 with shortest distance 260.0

        Check the neighbor vertex 0
        Remove the boosting value 40.0 from the distance 520.0

        Check the neighbor vertex 5
        Remove the boosting value 40.0 from the distance 440.0

        Check the neighbor vertex 9
        Remove the boosting value 40.0 from the distance 408.0
        Update the distance of vertex 9 from Infinity to 368.0
```

```
Fix vertex 5 with shortest distance 285.0

        Check the neighbor vertex 1
        Remove the boosting value 50.0 from the distance 465.0

        Check the neighbor vertex 9
        Remove the boosting value 50.0 from the distance 405.0
        Update the distance of vertex 9 from 368.0 to 355.0

        Check the neighbor vertex 3
        Remove the boosting value 50.0 from the distance 465.0

Fix vertex 9 with shortest distance 355.0

        Check the neighbor vertex 1
        Remove the boosting value 60.0 from the distance 503.0

        Check the neighbor vertex 5
        Remove the boosting value 60.0 from the distance 475.0

Fix vertex 7 with shortest distance 420.0

        Check the neighbor vertex 6
        Remove the boosting value 100.0 from the distance 740.0

Results:
------------------------------
ID      pred    dist

8       0       40.0
4       0       50.0
1       0       260.0
5       3       285.0
9       5       355.0
3       8       175.0
2       8       140.0
6       2       190.0
7       6       420.0
```

T3

```
Fix vertex 8 with shortest distance 120.0

        Check the neighbor vertex 4
        Remove the boosting value 96.0 from the distance 280.0

        Check the neighbor vertex 0
        Remove the boosting value 96.0 from the distance 240.0

        Check the neighbor vertex 2
        Remove the boosting value 96.0 from the distance 340.0
        Update the distance of vertex 2 from Infinity to 244.0

Fix vertex 4 with shortest distance 150.0

        Check the neighbor vertex 0
        Remove the boosting value 96.0 from the distance 300.0

        Check the neighbor vertex 8
        Remove the boosting value 96.0 from the distance 310.0

Fix vertex 2 with shortest distance 244.0

        Check the neighbor vertex 8
        Remove the boosting value 95.0 from the distance 464.0

        Check the neighbor vertex 6
        Remove the boosting value 95.0 from the distance 374.0
        Update the distance of vertex 6 from Infinity to 279.0
```

```
Fix vertex 1 with shortest distance 260.0

        Check the neighbor vertex 0
        Remove the boosting value 91.0 from the distance 520.0

        Check the neighbor vertex 5
        Remove the boosting value 91.0 from the distance 540.0
        Update the distance of vertex 5 from Infinity to 449.0

        Check the neighbor vertex 9
        Remove the boosting value 91.0 from the distance 408.0
        Update the distance of vertex 9 from Infinity to 317.0

Fix vertex 6 with shortest distance 279.0

        Check the neighbor vertex 7
        Remove the boosting value 58.0 from the distance 599.0
        Update the distance of vertex 7 from Infinity to 541.0

        Check the neighbor vertex 2
        Remove the boosting value 58.0 from the distance 409.0

Fix vertex 9 with shortest distance 317.0

        Check the neighbor vertex 1
        Remove the boosting value 93.0 from the distance 465.0

        Check the neighbor vertex 3
        Remove the boosting value 93.0 from the distance 837.0
        Update the distance of vertex 3 from Infinity to 744.0
```

```
Fix vertex 5 with shortest distance 449.0

        Check the neighbor vertex 1
        Remove the boosting value 12.0 from the distance 729.0

        Check the neighbor vertex 3
        Remove the boosting value 12.0 from the distance 629.0
        Update the distance of vertex 3 from 744.0 to 617.0

Fix vertex 7 with shortest distance 541.0

        Check the neighbor vertex 6
        Remove the boosting value 64.0 from the distance 861.0

Fix vertex 3 with shortest distance 617.0

        Check the neighbor vertex 9
        Remove the boosting value 0.0 from the distance 1137.0

        Check the neighbor vertex 5
        Remove the boosting value 0.0 from the distance 797.0

Results:
------------------------------
ID      pred    dist

4       0       150.0
8       0       120.0
1       0       260.0
5       1       449.0
9       1       317.0
3       5       617.0
6       2       279.0
7       6       541.0
2       8       244.0
```