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

BURAK KOCAUSTA
1901042605

1. SYSTEM REQUIREMENTS

There are 3 different packages. One of them is for DynamicGraph implementations, inside this package there are definitions of DynamicGraph interface, Graph interface, MyGraph implementation, and MatrixGraph implementation. Other package is GraphTraverseGTU, it has TraverseGraph class. Last one is for Dijkstra Algorithm. Those classes must be imported for testing.

They must be called like this:

```
MyGraph graph1 = new MyGraph(false);
```

MyGraph class requires graph is directed or not in its constructor.

```
TraverseGraph.differenceOfBFSAndDFS(graph1, 0)
```

Graph traversal method requires a starting index and, graph.

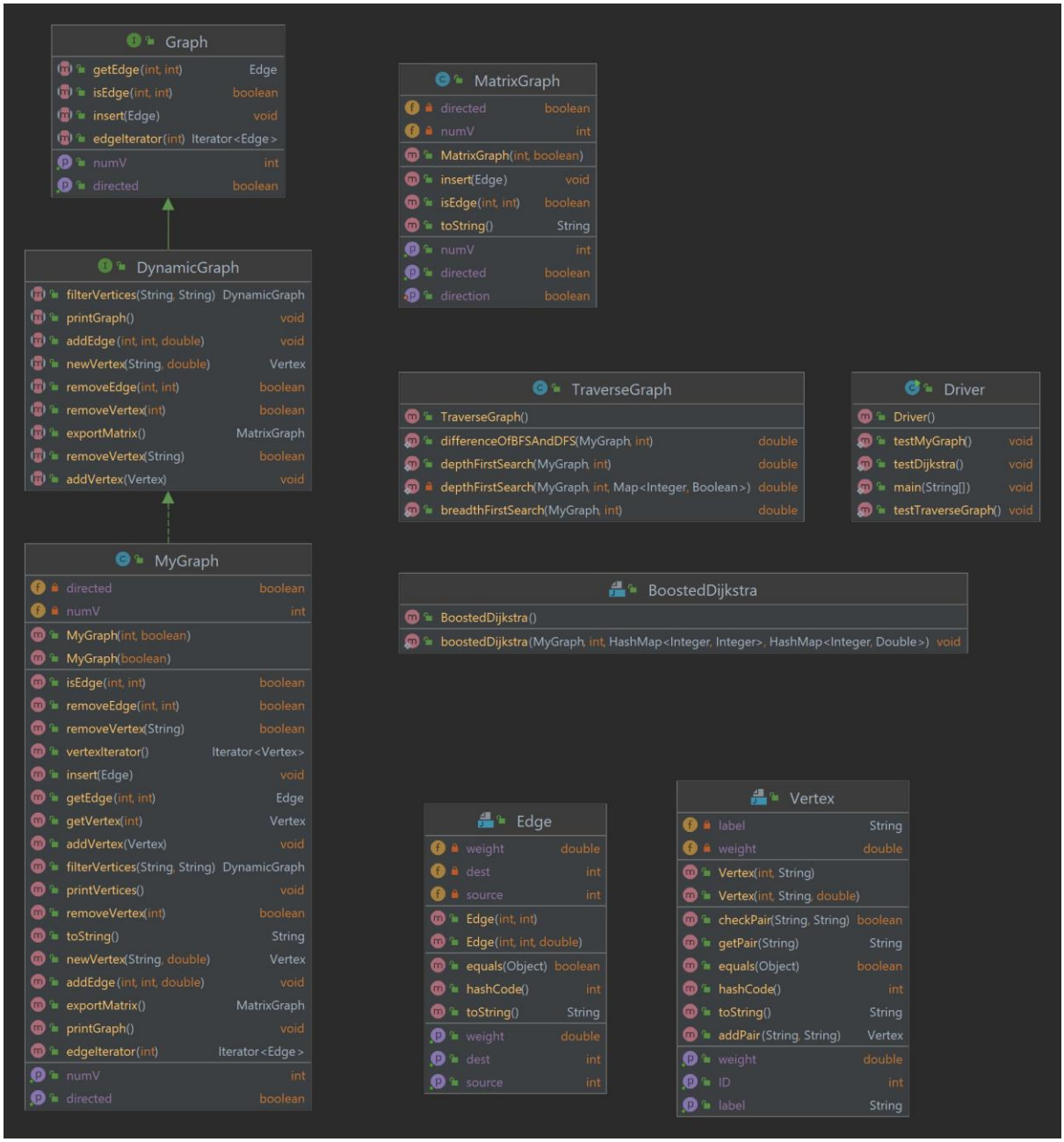
```
BoostedDijkstra.boostedDijkstra(graph2, 0, pred2, dist2);
```

BoostedDijkstra requires also graph, and starting index. Additionally, it requires predecessors, and distance maps to indicate the result.

All these implementations are inside these packages, therefore this import statement is required.

```
import DynamicGraphGTU.*;
import DijkstraAlgorithmGTU.*;
import GraphTraverseGTU.*;
```

2. CLASS DIAGRAM



3. PROBLEM SOLUTION APPROACH

Q1

In question 1, it is wanted to create a MyGraph class, which implements Graph, and DynamicGraph interfaces. First problem is, existence of Edge class. In lectures, we made an adjacency list implementation using Edges. In this homework it is also adjacency list, but it is dynamic, and has Vertex class. In lectures arrays are used to hold edges for each vertex, but I thought using

arrays are not wise choose, because graph is dynamic, therefore there are arbitrary removal. ID's are consecutive for regular graph implementation, but in this implementation they might not. ID's must be unique. It is ambiguous in the pdf that ID's can be added arbitrarily. My implementation also handles this case. There is no obligation to insert vertexes which starts from 0. I added this property, because Vertex class is independent, and it is a general class. Eventually, I used HashMap to access edges and vertices instead of regular array, because of the reason that I explained. It has constant time insertion, removal, and access in average, therefore using hash map have more advantages than its drawbacks. I used Edge classes source, and destination id's for to access vertices with these ids. In vertex class there are id, weight, label, and pairs holded. Equality check for vertices only done according to their id's. Hashing is also made with id of vertex. Pairs in Vertex class is hold in HashMap. Vertices also have weight, however it is said that, for MyGraph class weight of the vertices are not important. For exporting matrix, I used MatrixGraph class, and return that type of matrix. I added a matrix format to print the matrix version of graph. In print format First source vertex properties are printed, and if it is adjacent to another vertex, they are printed below with ids and edge weight. I also implement a printVertices method which print all vertices with detailed information.

Q2

I created a TraverseGraph class for differenceOfBFSandDFS() method. In this class there are two public method which calculates total distances of BFS and DFS traversal. difference method calls both of them and returns the result. For dfs, and bfs I holded the visited(identified) information on hash map. dfs method handles the unconnected graph case. In dfs graph it choses the smallest path while deciding where to go. I used two iterator in nested loops. One of them is used for finding shortest path. For bfs, it is similar in nested loop, two iterator handles the case. One of them founds the shortest distance, and next vertex is determined with that result. I confused about bfs part, Firstly I thought Dijkstra Algorithm is wanted or not. Later I decided that it is not Dijkstra Algorithm. Normally we learned the BFS and DFS traversals for unweighted graphs. Our graphs are weighted also they have vertex weighted. However, handling vertex weights are not wanted. But edge weights are still

important. In BFS and DFS traversals it founds the shortest path without weights. If weights are considered, Dijkstra is the best. These implementation of BFS and DFS still cannot work as good as Dijkstra, but they give better results according to their primal implementation.

Q3

In question 3, It wanted to implement a Dijkstra Algorithm which uses boosting property. Inside MyGraph class Vertices holds pairs, and inside these pairs they might be boosting property. This Dijkstra Algorithm calculates the shortest distance with that property. I made dist and pred HashMaps. I holded the vMinusS information on HashSet. While finding minimum distance, I used boosting information of that vertex, and I also used boost information while recording the distances to dist. Distance might be below 0, if vertex has boost, I handled this situation with setting it to 1. 1 means unweighted edge for my implementation. This method handles, unconnected graph situation. Indicates Infinity for unconnected edges in dist map. I used vertex iterator to save the id's in vMinusS set. This class assumes boosting keyword for pairs. Simply it works like, finding the minimum distance on vMinusS set. Process that minimum and remove from that set. While processing use boosting information unlike regular Dijkstra Algorithm. Then continue doing this till vMinusS set becomes empty. It means there aren't any vertex left to process.

4. Complexity Analysis

Q1

Vertex newVertex (String label, double weight)

```
@Override
public Vertex newVertex ( String label, double weight ) {
    Vertex vertex = new Vertex(++maxID, label, weight);
    vertexMap.put(vertex.getID(), vertex);
    edgeMap.put(vertex.getID(), new LinkedList<Edge>());
    maxID = numV;
    numV++;
    return vertex;
}
```

It is averagely constant time because inserting map is averagely constant. **Θ(1)**

Void addVertex (Vertex newVertex)

```
@Override
public void addVertex ( Vertex newVertex ) {
    if( vertexMap.put(newVertex.getID(), newVertex) == null )
        numV++;
    edgeMap.put(newVertex.getID(), new LinkedList<Edge>());
    maxID = (newVertex.getID() > maxID) ? newVertex.getID() : maxID;
}
```

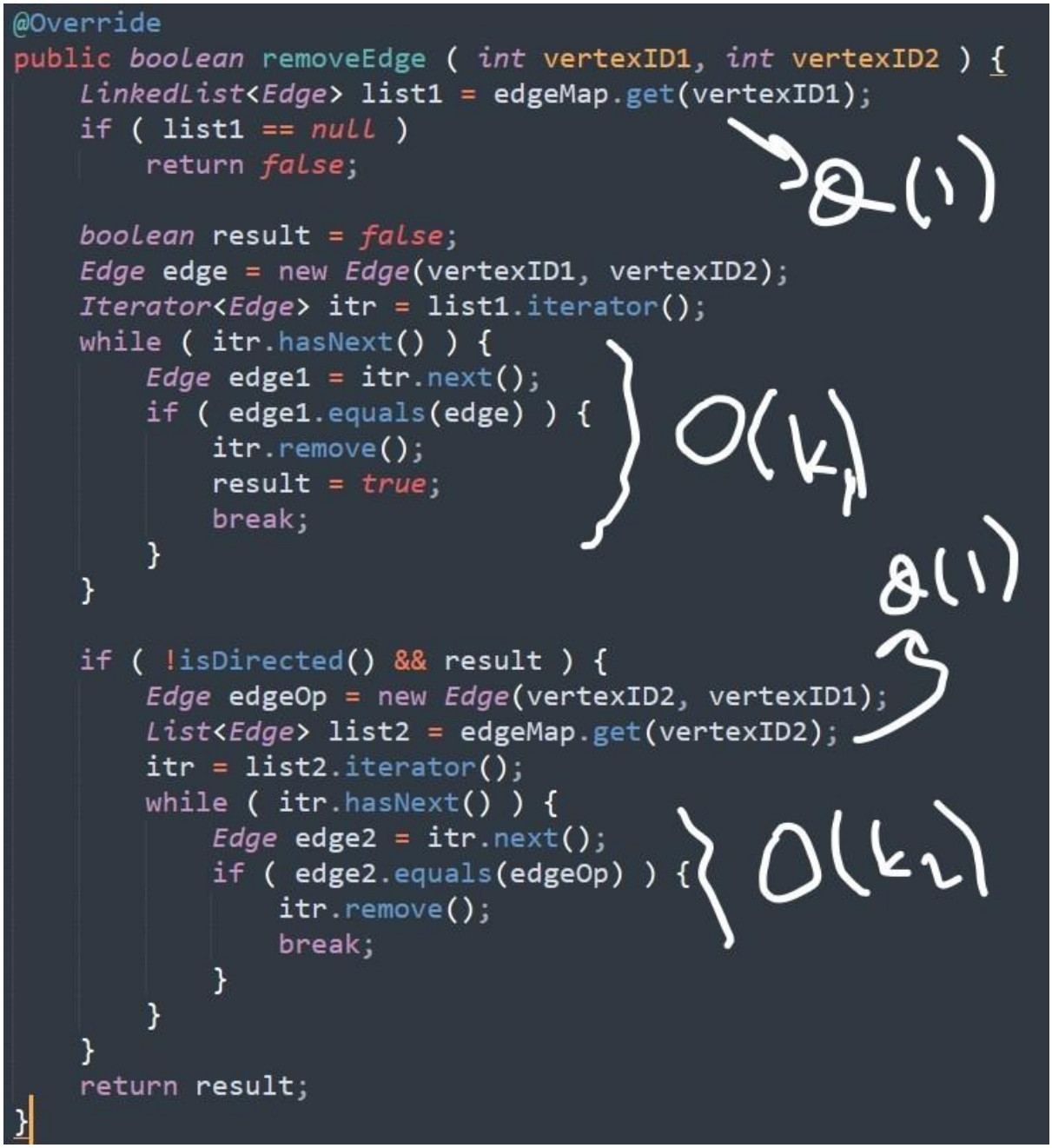
It is averagely constant time because inserting map is averagely constant. **Θ(1)**

void addEdge (int vertexID1, int vertexID2, double weight)

```
@Override
public void addEdge ( int vertexID1, int vertexID2, double weight ) {
    this.insert(new Edge(vertexID1, vertexID2, weight));
}
@Override
public void insert ( Edge edge ) {
    if ( vertexMap.get(edge.getSource()) == null || vertexMap.get(edge.getDest()) == null )
        return;

    edgeMap.get(edge.getSource()).add(edge);
    if( !isDirected() ) {
        edgeMap.get(edge.getDest()).add(new Edge(edge.getDest(), edge.getSource(), edge.getWeight()));
    }
}
```

It is averagely constant because, accessing map is averagely constant time, also insertion to linked list at tail is constant time. **Θ(1)**

boolean removeEdge (int vertexID1, int vertexID2)



Accessing lists are constant time, iterating vertex's edges depends on the number of edges done with vertexID1, and vertexID2, if graph is directed method works faster. k_1, and k_2 are number of edges vertex1 and vertex 2. **$O(k_1 + k_2)$. if graph is sparse O(1)**

boolean removeVertex (int vertexID)

```java
@Override
public boolean removeVertex ( int vertexID ) {

    // remove vertex
    if ( vertexMap.remove(vertexID) != null )
        numV--;
    else
        return false;

    // remove edges of this vertex
    edgeMap.remove(vertexID);
    for ( Map.Entry<Integer, LinkedList<Edge>> entry: edgeMap.entrySet() ) {
        List<Edge> list = entry.getValue();

        Iterator<Edge> itr = list.iterator();
        while ( itr.hasNext() ) {
            Edge val = itr.next();
            if ( val.getDest() == vertexID )
                itr.remove();
        }
    }
    return true;
}
```

Removing vertex requires to iterate all edges of graph, so time complexity is number of edges.**Θ(m)** (m is number of edges) if graph is **sparse** graph **o(V²)**. If it is **dense Θ(V²).**

boolean removeVertex (String label)

```java
@Override
public boolean removeVertex ( String label ) {
    int oldNum = numV;
    // hold deleted keys in tree set
    TreeSet<Integer> deletedKeys = new TreeSet<Integer>();

    Iterator<Map.Entry<Integer, Vertex>> itr1 = vertexMap.entrySet().iterator();
    while( itr1.hasNext() ) {
        Map.Entry<Integer, Vertex> entryVertex = itr1.next();
        if ( entryVertex.getValue().getLabel().equals(label) ) {
            deletedKeys.add(entryVertex.getKey());
            edgeMap.remove(entryVertex.getKey());
            itr1.remove();
            numV--;
        }
    }

    // check if any vertex is deleted
    if ( oldNum == numV )
        return false;

    // remove edges of this vertex
    for ( Map.Entry<Integer, LinkedList<Edge>> entryEdge: edgeMap.entrySet() ) {
        List<Edge> list = entryEdge.getValue();

        Iterator<Edge> itr = list.iterator();
        while ( itr.hasNext() ) {
            Edge val = itr.next();
            if ( deletedKeys.contains(Integer.valueOf(val.getDest())) )
                itr.remove();
        }
    }
    return true;
}
```

Iterating through vertices takes **Θ(V)** time. If any vertex is deleted. It is required to traverse all edges. It takes **Θ(E)** times. Eventually time complexity is **Θ(V+E)**. If it is **sparse** graph It can be said time complexity is **Θ(V)**, If it is **dense** complexity becomes **Θ(V^2)**.

DynamicGraph filterVertices (String key, String filter)

```java
@Override
public DynamicGraph filterVertices ( String key, String filter ) {
    MyGraph graph = new MyGraph(true);
    // hold filtered edge hash set to prevent duplication
    HashSet<Edge> edgeSet = new HashSet<Edge>();

    for ( Map.Entry<Integer, Vertex> entryVertex: vertexMap.entrySet() ) {
        if ( entryVertex.getValue().checkPair(key, filter) ) {
            graph.addVertex(entryVertex.getValue());
            LinkedList<Edge> list = edgeMap.get(entryVertex.getKey());
            for ( Edge edge: list )
                edgeSet.add(edge);
        }
    }

    for ( Edge edge: edgeSet )
        graph.insert(edge);
    // set current graphs direction
    graph.directed = isDirected();
    return graph;
}
```

Iterating through edges takes takes **Θ(E)** time. Iterating through edgeSet takes **Θ(k)** times and k is smaller than E. So time complexity is **Θ(E)**. If it is **sparse** graph It can be said time complexity is **Θ(V)**, If it is **dense** complexity becomes **Θ(V^2)**.

MatrixGraph exportMatrix ( )

```java
@Override
public MatrixGraph exportMatrix ( ) {
    MatrixGraph graph = new MatrixGraph(numV, true);
    HashSet<Edge> edgeSet = new HashSet<Edge>();
    LinkedHashMap<Integer, Integer> keyMap = new LinkedHashMap<Integer, Integer>();

    // set indexes for matrix graph
    int i = 0;
    for ( Integer key: vertexMap.keySet() ) {
        keyMap.put(key, i);
        ++i;
    }

    for ( LinkedList<Edge> list: edgeMap.values() )
        for ( Edge edge: list )
            edgeSet.add(edge);

    for ( Edge edge: edgeSet )
        graph.insert(new Edge(keyMap.get(edge.getSource()), keyMap.get(edge.getDest()), edge.getWeight()));

    graph.setDirection(isDirected());
    return graph;
}
```

void printGraph ( )

```java
@Override
public void printGraph ( ) {
    System.out.println(this.toString());
}
```

```java
@Override
public String toString ( ) {
    StringBuilder sb = new StringBuilder();

    sb.append("Adjacency List\n");
    if ( numV == 0 ) {
        sb.append("empty graph\n");
        return sb.toString();
    }

    for ( Map.Entry<Integer, LinkedList<Edge>> entryEdge: edgeMap.entrySet() ) {
        sb.append("vertex (id = " + entryEdge.getKey() + ", label = " + vertexMap.get(entryEdge.getKey()).getLabel() +
            ", weight(vertex) = " + vertexMap.get(entryEdge.getKey()).getWeight() + ") -->\n");
        for( Edge e: entryEdge.getValue() ) {

            sb.append("\t" + e + "\n");
        }
    }
    return sb.toString();
}
```

It iterates through all edges so time complexity is **Θ(E).** If it is **sparse** graph It can be said time complexity is **Θ(V)**, If it is **dense** complexity becomes **Θ(V^2)**.

5. TEST CASES

MyGraph Class

```java
        System.out.println("____Testing MyGraph Class____\n");

        System.out.println("Create empty undirected graph\n");

        MyGraph graph1 = new MyGraph(false);
        System.out.println(graph1 + "------------------size = " + graph1.getNumV() + "\n");

        System.out.println("Insert a vertex to that graph\n");
        graph1.addVertex(new Vertex(0, "v0", 52.3));
        System.out.println(graph1 + "------------------size = " + graph1.getNumV() + "\n");

        System.out.println("Insert more vertex to that graph");
        graph1.addVertex(new Vertex(1, "v1", 12.9));
        graph1.newVertex("v2", 0.9);
        graph1.addVertex(new Vertex(3, "v3", 67.0));
        graph1.addVertex(new Vertex(4, "v4", 11.31));
        graph1.addVertex(new Vertex(5, "v5", 22.34));
        graph1.addVertex(new Vertex(6, "v6", 12.13));
        graph1.addVertex(new Vertex(7, "v7", 71.3));
        graph1.addVertex(new Vertex(8, "v8", 62.3));
        graph1.newVertex("v9", 15.9);
        graph1.newVertex("v10", 22.39);
        graph1.printGraph();

        System.out.println("\nPrint vertices of graph\n");
        graph1.printVertices();

        System.out.println("\nAdd an edge to graph\n");
        graph1.addEdge(4,6, 3.1);

        graph1.printGraph();
```

```
System.out.println("\nAdd more edges to the graph\n");
graph1.addEdge(0,10, 22.1);
graph1.addEdge(2,6, 3.6);
graph1.addEdge(8,7, 15.9);
graph1.addEdge(3,8, 31.52);
graph1.addEdge(1,3, 15.9);
graph1.addEdge(3,2, 35.20);
graph1.insert(new Edge(1,5));
graph1.insert(new Edge(0,9));
graph1.printGraph();

System.out.println("\nExport matrix of current graph\n");
System.out.println(graph1.exportMatrix());

System.out.println("\nRemove a vertex from graph with id 6\n");
graph1.removeVertex(6);
graph1.printGraph();
graph1.printVertices();

System.out.println("\nRemove a vertex from graph with label v10\n");
graph1.removeVertex("v10");
graph1.printGraph();

System.out.println("\nTry to remove unexisted vertex\n");
graph1.removeVertex(30);
graph1.printGraph();

System.out.println("\nExport matrix of current graph\n");
System.out.println(graph1.exportMatrix());

System.out.println("\nRemove a vertex from graph with id 8\n");
graph1.removeVertex(8);
System.out.println(graph1 + "-----------------size = " + graph1.getNumV() + "\n");
System.out.println("\nRemove an edge between 1 and 5\n");
graph1.removeEdge(1, 5);
System.out.println(graph1 + "-----------------size = " + graph1.getNumV() + "\n");

System.out.println("\nRemove an edge between 1 and 3\n");
graph1.removeEdge(1, 3);
graph1.printGraph();

System.out.println("\nRemove an edge between 2 and 3\n");
graph1.removeEdge(2, 3);
graph1.printGraph();

System.out.println("\nRemove an edge between 0 and 9\n");
graph1.removeEdge(0, 9);
graph1.printGraph();

System.out.println("\nTry to remove unexisted edge\n");
graph1.removeEdge(1, 5);
graph1.printGraph();


System.out.println("\nInsert arbitrary id vertices\n");

graph1.addVertex(new Vertex(19, "v13", 12.9));
graph1.newVertex("v12", 0.9);
graph1.addVertex(new Vertex(16, "v14", 67.0));
graph1.addVertex(new Vertex(31, "v15", 11.31));
graph1.addVertex(new Vertex(24, "v16", 22.34));
graph1.addVertex(new Vertex(22, "v17", 12.13));
graph1.addVertex(new Vertex(21, "v18", 71.3));
graph1.addVertex(new Vertex(37, "v19", 62.3));
graph1.newVertex("v19", 15.9);
graph1.newVertex("v19", 22.39);
System.out.println(graph1 + "-----------------size = " + graph1.getNumV() + "\n");
graph1.printVertices();
```

```
System.out.println("\nAdd edges to the graph\n");

graph1.addEdge(37, 22, 10.6);
graph1.addEdge(39, 37, 24.12);
graph1.addEdge(16, 37, 41.3);
graph1.addEdge(0, 16, 6.5);
graph1.addEdge(3, 9, 7.31);
graph1.addEdge(20, 24, 8.16);
graph1.addEdge(14, 5, 5.42);
graph1.addEdge(3, 14, 12.31);

graph1.printGraph();


System.out.println("\nRemove vertices which have label v19\n");
graph1.removeVertex("v19");
graph1.printGraph();
graph1.printVertices();

System.out.println("\nAdd more edges to the graph\n");
graph1.addEdge(9, 19, 4.21);
graph1.addEdge(16, 24, 2.11);
graph1.addEdge(22, 31, 11.5);
graph1.addEdge(1, 21, 70.81);
graph1.addEdge(2, 1, 131.62);
graph1.printGraph();
```

```
System.out.println("\nAdd Pairs to the vertices using iterator\n");
Iterator<Vertex> itr = graph1.vertexIterator();
while ( itr.hasNext() ) {
    Vertex val = itr.next();
    if (val.getID() % 2 == 0)
        val.addPair("length", "5");
    else
        val.addPair("length", Integer.valueOf(val.getID()*2).toString());

    val.addPair("boosting", Integer.valueOf(val.getID()*3).toString());

    if ( val.getID() % 4 == 0)
        val.addPair("color", "blue");
    else if ( val.getID() % 3 == 0 )
        val.addPair("color", "red");
    else if ( val.getID() % 5 == 0 )
        val.addPair("color", "green");
    else
        val.addPair("color", "pink");
}

graph1.printVertices();
System.out.print("\n");
graph1.printGraph();

System.out.println("\nTesting filter method");
System.out.println("\nFiltered Graph(color, blue):");
DynamicGraph filtered1 = graph1.filterVertices("color", "blue");
filtered1.printGraph();
```

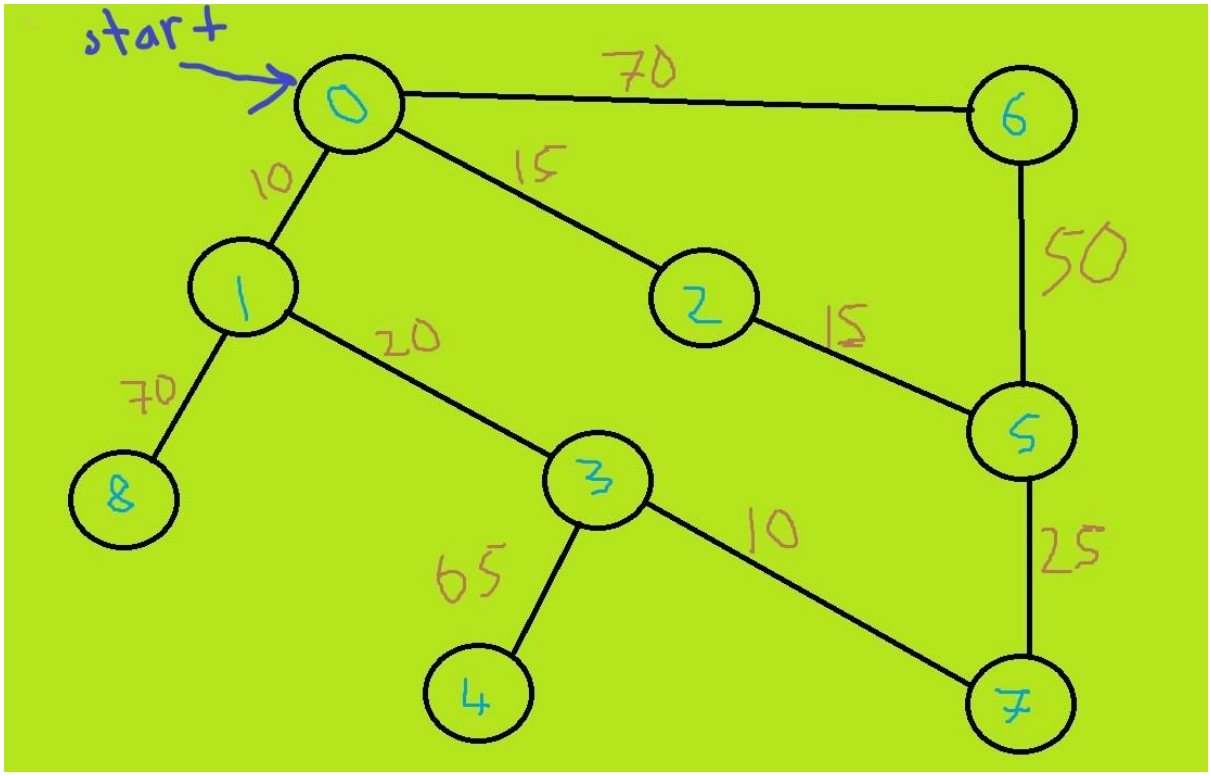
```
System.out.println("\nFiltered Graph(length, 5)");
DynamicGraph filtered2 = graph1.filterVertices("length", "5");
filtered2.printGraph();

System.out.println("\nTesting other methods\n");

graph1.printGraph();
System.out.println("graph1.getVertex(9) returns: " + graph1.getVertex(9));
System.out.println("\ngraph1.getVertex(30)(not exists) returns:\n" + graph1.getVertex(30));
System.out.println("\ngraph1.isEdge(9,19) = " + graph1.isEdge(9, 19));
System.out.println("\ngraph1.isEdge(5,16)(not exists) = " + graph1.isEdge(5, 16));
System.out.println("\ngraph1.getEdge(24,16) = " + graph1.getEdge(24, 16));
System.out.println("\ngraph1.getEdge(4,7)(not exists) = " + graph1.getEdge(4, 7));


System.out.println("\nTest a directed graph\n");

MyGraph graph2 = new MyGraph(true);
graph2.addVertex(new Vertex(0, "v0", 6.0).addPair("color", "purple"));
graph2.addVertex(new Vertex(1, "v1", 13.31).addPair("color", "purple"));
graph2.addVertex(new Vertex(2, "v2", 21.34).addPair("color", "purple"));
graph2.addVertex(new Vertex(3, "v3", 12.13).addPair("color", "purple"));
graph2.addVertex(new Vertex(4, "v4", 76.3).addPair("color", "purple"));
graph2.addVertex(new Vertex(5, "v5", 7.3).addPair("color", "purple"));
graph2.addVertex(new Vertex(6, "v6", 7.0).addPair("boosting", "6"));
graph2.addVertex(new Vertex(7, "v7", 1.31).addPair("boosting", "6"));
graph2.addVertex(new Vertex(8, "v8", 25.34));
graph2.addVertex(new Vertex(9, "v9", 11.43).addPair("color", "green"));
graph2.addVertex(new Vertex(10, "v10", 72.4).addPair("boosting", "6"));
graph2.addVertex(new Vertex(11, "v11", 65.3).addPair("boosting", "9"));

System.out.println("Inserting vertices to directed graph\n");
graph2.printVertices();
System.out.println("\ngraph2.isDirected() = " + graph2.isDirected());
```

```
System.out.println("\nInserting edges to directed graph\n");
graph2.addEdge(0, 11, 12.6);
graph2.addEdge(1, 5, 25.12);
graph2.addEdge(2, 9, 46.3);
graph2.addEdge(5, 1, 16.5);
graph2.addEdge(3, 1, 12.31);
graph2.addEdge(4, 9, 51.16);
graph2.addEdge(9, 6, 56.42);
graph2.addEdge(3, 9, 11.31);
graph2.addEdge(8, 6, 10.6);
graph2.addEdge(6, 4, 24.12);
graph2.addEdge(7, 3, 41.3);
graph2.addEdge(0, 2, 6.5);
graph2.addEdge(2, 10, 7.31);
graph2.insert(new Edge(10,1, 5.3));
graph2.insert(new Edge(11,6));

graph2.printGraph();

System.out.println("\nExport matrix of current graph\n");
System.out.println(graph2.exportMatrix());

graph2.printVertices();
System.out.println("\nFiltered Graph(color, purple):");
graph2.filterVertices("color", "purple").printGraph();

System.out.println("\nAdjancecy Matrix of Filtered Graph(color, purple):");
System.out.println(graph2.filterVertices("color", "purple").exportMatrix());

graph2.printGraph();
```

```
System.out.println("\nRemove some vertices from graph\n");

graph2.removeVertex(4);
graph2.removeVertex(1);
graph2.removeVertex("v10");

graph2.printGraph();

System.out.println("\nRemove some edges from graph\n");

graph2.removeEdge(0, 2);
graph2.removeEdge(11, 6);
graph2.removeEdge(9, 6);
graph2.printGraph();

System.out.println("\n__Test Results for MyGraph and DynamicGraph__");
System.out.println("1- Insertion of edges and vertices are tested.");
System.out.println("2- Removal of edges and vertices are tested.");
System.out.println("3- Export Matrix and Filter methods are tested.");
System.out.println("4- printGraph() and toString are tested.");
System.out.println("5- Graph is tested after deletion and insertion.");
System.out.println("6- Graph is tested for nonconsecutive ID's.");
System.out.println("7- Iterator's are tested, and both directed, undirected graphs are tested.");
System.out.println("8- Other DynamicGraph, and MyGraph methods are tested.\n");
```

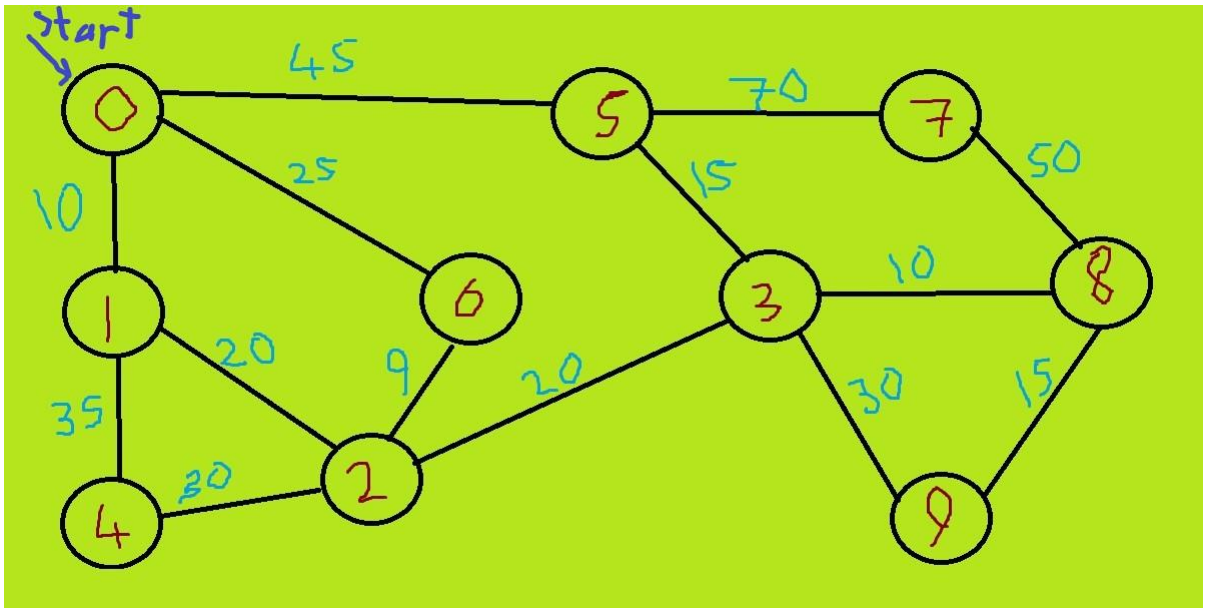
TraverseGraph

1



```java
public static void testTraverseGraph ( ) {
    System.out.println("____Testing TraverseGraph Class(calculating difference)____\n");

    MyGraph graph1 = new MyGraph(false);
    graph1.addVertex(new Vertex(0, "v0", 5.0));
    graph1.addVertex(new Vertex(1, "v1", 15.0));
    graph1.addVertex(new Vertex(2, "v2", 2.5));
    graph1.addVertex(new Vertex(3, "v3", 3.0));
    graph1.addVertex(new Vertex(4, "v4", 4.0));
    graph1.addVertex(new Vertex(5, "v5", 10.3));
    graph1.addVertex(new Vertex(6, "v6", 63.0));
    graph1.addVertex(new Vertex(7, "v7", 40.0));
    graph1.addVertex(new Vertex(8, "v8", 15.9));

    graph1.addEdge(0, 1, 10.0);
    graph1.addEdge(0, 2, 15.0);
    graph1.addEdge(0, 6, 70.0);
    graph1.addEdge(1, 3, 20.0);
    graph1.addEdge(1, 8, 70.0);
    graph1.addEdge(2, 3, 30.0);
    graph1.addEdge(2, 5, 15.0);
    graph1.addEdge(3, 4, 65.0);
    graph1.addEdge(3, 7, 10.0);
    graph1.addEdge(5, 6, 50.0);
    graph1.addEdge(5, 7, 25.0);

    graph1.printVertices();
    System.out.print("\n");
    graph1.printGraph();

    System.out.println("Start id is 0 for traversals\n");
    System.out.println( "Distance(BFS) - Distance(DFS) = " + TraverseGraph.differenceOfBFSAndDFS(graph1, 0) + "\n");
```

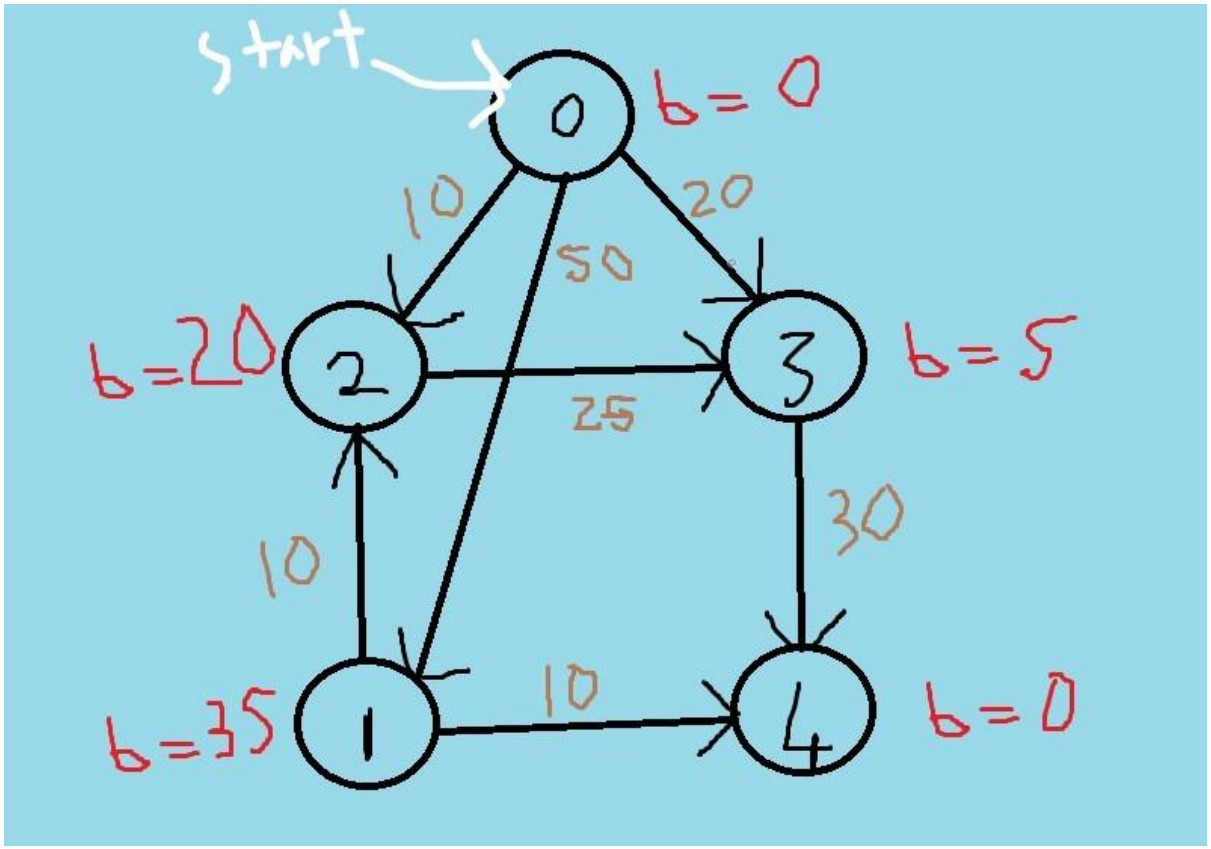
2



```
System.out.println("\nTest for another graph\n");
MyGraph graph2 = new MyGraph(false);
graph2.addVertex(new Vertex(0, "v0", 15.0));
graph2.addVertex(new Vertex(1, "v1", 25.0));
graph2.addVertex(new Vertex(2, "v2", 2.3));
graph2.addVertex(new Vertex(3, "v3", 4.0));
graph2.addVertex(new Vertex(4, "v4", 9.0));
graph2.addVertex(new Vertex(5, "v5", 10.13));
graph2.addVertex(new Vertex(6, "v6", 23.0));
graph2.addVertex(new Vertex(7, "v7", 50.0));
graph2.addVertex(new Vertex(8, "v8", 25.9));
graph2.addVertex(new Vertex(9, "v9", 135.9));

graph2.addEdge(0, 1, 10.0);
graph2.addEdge(0, 5, 45.0);
graph2.addEdge(0, 6, 25.0);
graph2.addEdge(1, 2, 20.0);
graph2.addEdge(1, 4, 35.0);
graph2.addEdge(2, 4, 30.0);
graph2.addEdge(2, 6, 9.0);
graph2.addEdge(3, 4, 20.0);
graph2.addEdge(3, 9, 30.0);
graph2.addEdge(3, 8, 10.0);
graph2.addEdge(3, 5, 15.0);
graph2.addEdge(5, 7, 70.0);
graph2.addEdge(7, 8, 50.0);
graph2.addEdge(8, 9, 15.0);

graph2.printVertices();
System.out.print("\n");
graph2.printGraph();

System.out.println("Start id is 4 for traversals\n");
System.out.println( "Distance(BFS) - Distance(DFS) = " + TraverseGraph.differenceOfBFSAndDFS(graph2, 4) + "\n");
```

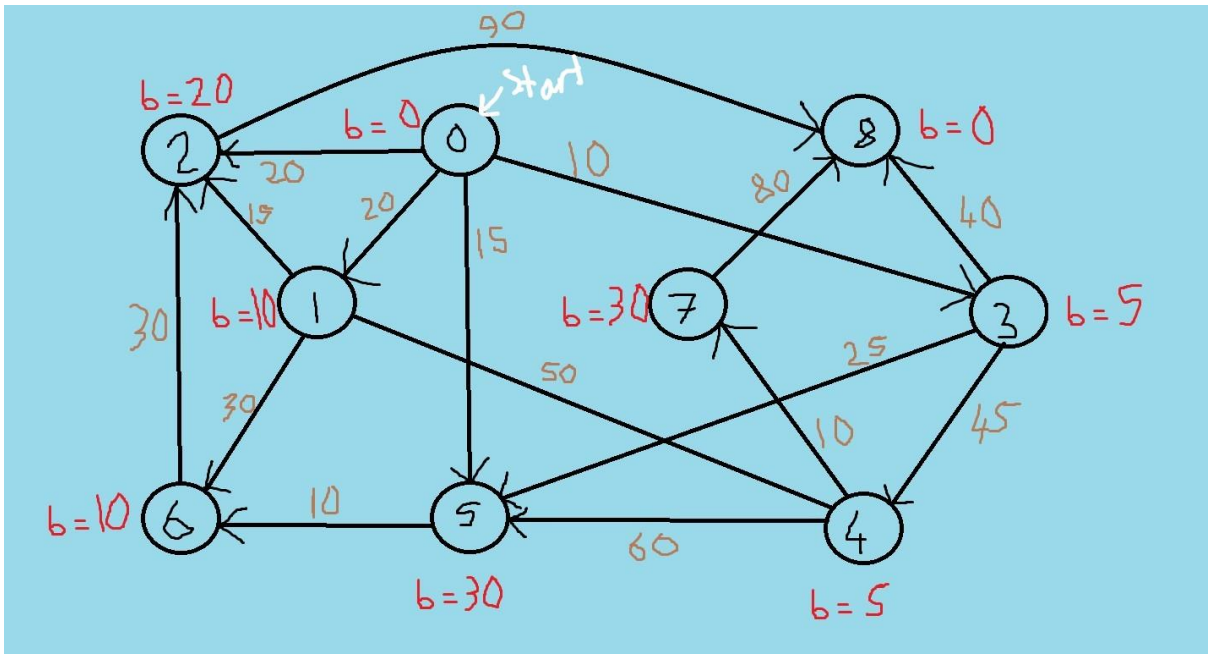
BoostedDijkstra

1



```java
public static void testDijkstra ( ) {
    System.out.println("\n____Testing BoostedDijkstra Class____\n");
    MyGraph graph1 = new MyGraph(true);
    graph1.addVertex(new Vertex(0, "v0", 17.0));
    graph1.addVertex(new Vertex(1, "v1", 11.21).addPair("boosting", "35.0"));
    graph1.addVertex(new Vertex(2, "v2", 42.34).addPair("boosting", "20.0"));
    graph1.addVertex(new Vertex(3, "v3", 13.33).addPair("boosting", "5.0"));
    graph1.addVertex(new Vertex(4, "v4", 16.5));


    graph1.addEdge(0, 1, 50.0);
    graph1.addEdge(0, 2, 10.0);
    graph1.addEdge(0, 3, 20.0);
    graph1.addEdge(1, 4, 10.0);
    graph1.addEdge(1, 2, 10.0);
    graph1.addEdge(2, 3, 25.0);
    graph1.addEdge(3, 4, 30.0);
    graph1.printVertices();
    System.out.print("\n");
    graph1.printGraph();

    System.out.println("\nStarting id is 0\n");

    HashMap<Integer, Integer> pred1 = new HashMap<Integer, Integer>();
    HashMap<Integer, Double> dist1 = new HashMap<Integer, Double>();

    BoostedDijkstra.boostedDijkstra(graph1, 0, pred1, dist1);

    System.out.println("Predecessors for each id = " + pred1);
    System.out.println("Distances to each id = " + dist1);
```

2



```
System.out.println("\nTest for another graph\n");

MyGraph graph2 = new MyGraph(true);
graph2.addVertex(new Vertex(0, "v0", 7.0));
graph2.addVertex(new Vertex(1, "v1", 1.31).addPair("boosting", "10.0"));
graph2.addVertex(new Vertex(2, "v2", 2.34).addPair("boosting", "20.0"));
graph2.addVertex(new Vertex(3, "v3", 12.13).addPair("boosting", "5.0"));
graph2.addVertex(new Vertex(4, "v4", 7.3).addPair("boosting", "5.0"));
graph2.addVertex(new Vertex(5, "v5", 11.1).addPair("boosting", "30.0"));
graph2.addVertex(new Vertex(6, "v6", 27.34).addPair("boosting", "10.0"));
graph2.addVertex(new Vertex(7, "v7", 2.13).addPair("boosting", "30.0"));
graph2.addVertex(new Vertex(8, "v8", 72.3));


graph2.addEdge(0, 1, 20.0);
graph2.addEdge(0, 2, 20.0);
graph2.addEdge(0, 3, 10.0);
graph2.addEdge(0, 5, 15.0);
graph2.addEdge(1, 2, 15.0);
graph2.addEdge(1, 6, 30.0);
graph2.addEdge(1, 4, 50.0);
graph2.addEdge(2, 8, 90.0);
graph2.addEdge(3, 8, 40.0);
graph2.addEdge(3, 5, 25.0);
graph2.addEdge(3, 4, 45.0);
graph2.addEdge(4, 5, 60.0);
graph2.addEdge(4, 7, 10.0);
graph2.addEdge(5, 6, 10.0);
graph2.addEdge(6, 2, 30.0);
graph2.addEdge(7, 8, 80.0);
```

```
graph2.printVertices();
System.out.print("\n");
graph2.printGraph();

System.out.println("\nStarting id is 0\n");

HashMap<Integer, Integer> pred2 = new HashMap<Integer, Integer>();
HashMap<Integer, Double> dist2 = new HashMap<Integer, Double>();

BoostedDijkstra.boostedDijkstra(graph2, 0, pred2, dist2);

System.out.println("Predecessors for each id = " + pred2);
System.out.println("Distances to each id = " + dist2);
```

6. RUNNING AND RESULTS

MyGraph Results

```
___Testing MyGraph Class___

Create empty undirected graph

Adjacency List
empty graph
----------------size = 0

Insert a vertex to that graph

Adjacency List
vertex (id = 0, label = v0, weight(vertex) = 52.3) -->
----------------size = 1

Insert more vertex to that graph
Adjacency List
vertex (id = 0, label = v0, weight(vertex) = 52.3) -->
vertex (id = 1, label = v1, weight(vertex) = 12.9) -->
vertex (id = 2, label = v2, weight(vertex) = 0.9) -->
vertex (id = 3, label = v3, weight(vertex) = 67.0) -->
vertex (id = 4, label = v4, weight(vertex) = 11.31) -->
vertex (id = 5, label = v5, weight(vertex) = 22.34) -->
vertex (id = 6, label = v6, weight(vertex) = 12.13) -->
vertex (id = 7, label = v7, weight(vertex) = 71.3) -->
vertex (id = 8, label = v8, weight(vertex) = 62.3) -->
vertex (id = 9, label = v9, weight(vertex) = 15.9) -->
vertex (id = 10, label = v10, weight(vertex) = 22.39) -->
```

```
Print vertices of graph

Vertices:

id = 0, label = v0, weight = 52.3
pairs: {}
id = 1, label = v1, weight = 12.9
pairs: {}
id = 2, label = v2, weight = 0.9
pairs: {}
id = 3, label = v3, weight = 67.0
pairs: {}
id = 4, label = v4, weight = 11.31
pairs: {}
id = 5, label = v5, weight = 22.34
pairs: {}
id = 6, label = v6, weight = 12.13
pairs: {}
id = 7, label = v7, weight = 71.3
pairs: {}
id = 8, label = v8, weight = 62.3
pairs: {}
id = 9, label = v9, weight = 15.9
pairs: {}
id = 10, label = v10, weight = 22.39
pairs: {}

Add an edge to graph

Adjacency List
vertex (id = 0, label = v0, weight(vertex) = 52.3) -->
vertex (id = 1, label = v1, weight(vertex) = 12.9) -->
vertex (id = 2, label = v2, weight(vertex) = 0.9) -->
vertex (id = 3, label = v3, weight(vertex) = 67.0) -->
vertex (id = 4, label = v4, weight(vertex) = 11.31) -->
       [(4, 6): 3.1]
vertex (id = 5, label = v5, weight(vertex) = 22.34) -->
vertex (id = 6, label = v6, weight(vertex) = 12.13) -->
       [(6, 4): 3.1]
vertex (id = 7, label = v7, weight(vertex) = 71.3) -->
vertex (id = 8, label = v8, weight(vertex) = 62.3) -->
vertex (id = 9, label = v9, weight(vertex) = 15.9) -->
vertex (id = 10, label = v10, weight(vertex) = 22.39) -->
```

```
Add more edges to the graph

Adjacency List
vertex (id = 0, label = v0, weight(vertex) = 52.3) -->
        [(0, 10): 22.1]
        [(0, 9): 1.0]
vertex (id = 1, label = v1, weight(vertex) = 12.9) -->
        [(1, 3): 15.9]
        [(1, 5): 1.0]
vertex (id = 2, label = v2, weight(vertex) = 0.9) -->
        [(2, 6): 3.6]
        [(2, 3): 35.2]
vertex (id = 3, label = v3, weight(vertex) = 67.0) -->
        [(3, 8): 31.52]
        [(3, 1): 15.9]
        [(3, 2): 35.2]
vertex (id = 4, label = v4, weight(vertex) = 11.31) -->
        [(4, 6): 3.1]
vertex (id = 5, label = v5, weight(vertex) = 22.34) -->
        [(5, 1): 1.0]
vertex (id = 6, label = v6, weight(vertex) = 12.13) -->
        [(6, 4): 3.1]
        [(6, 2): 3.6]
vertex (id = 7, label = v7, weight(vertex) = 71.3) -->
        [(7, 8): 15.9]
vertex (id = 8, label = v8, weight(vertex) = 62.3) -->
        [(8, 7): 15.9]
        [(8, 3): 31.52]
vertex (id = 9, label = v9, weight(vertex) = 15.9) -->
        [(9, 0): 1.0]
vertex (id = 10, label = v10, weight(vertex) = 22.39) -->
        [(10, 0): 22.1]
```

```
Export matrix of current graph

     0      1      2      3      4      5      6      7      8      9     10

0    x      x      x      x      x      x      x      x      x      1.00  22.10

1    x      x      x      15.90  x      1.00   x      x      x      x      x

2    x      x      x      35.20  x      x      3.60   x      x      x      x

3    x      15.90  35.20  x      x      x      x      x      31.52  x      x

4    x      x      x      x      x      x      3.10   x      x      x      x

5    x      1.00   x      x      x      x      x      x      x      x      x

6    x      x      3.60   x      3.10   x      x      x      x      x      x

7    x      x      x      x      x      x      x      x      15.90  x      x

8    x      x      x      31.52  x      x      x      15.90  x      x      x

9    1.00   x      x      x      x      x      x      x      x      x      x

10   22.10  x      x      x      x      x      x      x      x      x      x
```

```
Remove a vertex from graph with id 6

Adjacency List
vertex (id = 0, label = v0, weight(vertex) = 52.3) -->
       [(0, 10): 22.1]
       [(0, 9): 1.0]
vertex (id = 1, label = v1, weight(vertex) = 12.9) -->
       [(1, 3): 15.9]
       [(1, 5): 1.0]
vertex (id = 2, label = v2, weight(vertex) = 0.9) -->
       [(2, 3): 35.2]
vertex (id = 3, label = v3, weight(vertex) = 67.0) -->
       [(3, 8): 31.52]
       [(3, 1): 15.9]
       [(3, 2): 35.2]
vertex (id = 4, label = v4, weight(vertex) = 11.31) -->
vertex (id = 5, label = v5, weight(vertex) = 22.34) -->
       [(5, 1): 1.0]
vertex (id = 7, label = v7, weight(vertex) = 71.3) -->
       [(7, 8): 15.9]
vertex (id = 8, label = v8, weight(vertex) = 62.3) -->
       [(8, 7): 15.9]
       [(8, 3): 31.52]
vertex (id = 9, label = v9, weight(vertex) = 15.9) -->
       [(9, 0): 1.0]
vertex (id = 10, label = v10, weight(vertex) = 22.39) -->
       [(10, 0): 22.1]

Vertices:

id = 0, label = v0, weight = 52.3
pairs: {}
id = 1, label = v1, weight = 12.9
pairs: {}
id = 2, label = v2, weight = 0.9
pairs: {}
id = 3, label = v3, weight = 67.0
pairs: {}
id = 4, label = v4, weight = 11.31
pairs: {}
id = 5, label = v5, weight = 22.34
pairs: {}
id = 7, label = v7, weight = 71.3
pairs: {}
id = 8, label = v8, weight = 62.3
pairs: {}
id = 9, label = v9, weight = 15.9
pairs: {}
id = 10, label = v10, weight = 22.39
pairs: {}
```

```
Remove a vertex from graph with label v10

Adjacency List
vertex (id = 0, label = v0, weight(vertex) = 52.3) -->
       [(0, 9): 1.0]
vertex (id = 1, label = v1, weight(vertex) = 12.9) -->
       [(1, 3): 15.9]
       [(1, 5): 1.0]
vertex (id = 2, label = v2, weight(vertex) = 0.9) -->
       [(2, 3): 35.2]
vertex (id = 3, label = v3, weight(vertex) = 67.0) -->
       [(3, 8): 31.52]
       [(3, 1): 15.9]
       [(3, 2): 35.2]
vertex (id = 4, label = v4, weight(vertex) = 11.31) -->
vertex (id = 5, label = v5, weight(vertex) = 22.34) -->
       [(5, 1): 1.0]
vertex (id = 7, label = v7, weight(vertex) = 71.3) -->
       [(7, 8): 15.9]
vertex (id = 8, label = v8, weight(vertex) = 62.3) -->
       [(8, 7): 15.9]
       [(8, 3): 31.52]
vertex (id = 9, label = v9, weight(vertex) = 15.9) -->
       [(9, 0): 1.0]


Try to remove unexisted vertex

Adjacency List
vertex (id = 0, label = v0, weight(vertex) = 52.3) -->
       [(0, 9): 1.0]
vertex (id = 1, label = v1, weight(vertex) = 12.9) -->
       [(1, 3): 15.9]
       [(1, 5): 1.0]
vertex (id = 2, label = v2, weight(vertex) = 0.9) -->
       [(2, 3): 35.2]
vertex (id = 3, label = v3, weight(vertex) = 67.0) -->
       [(3, 8): 31.52]
       [(3, 1): 15.9]
       [(3, 2): 35.2]
vertex (id = 4, label = v4, weight(vertex) = 11.31) -->
vertex (id = 5, label = v5, weight(vertex) = 22.34) -->
       [(5, 1): 1.0]
vertex (id = 7, label = v7, weight(vertex) = 71.3) -->
       [(7, 8): 15.9]
vertex (id = 8, label = v8, weight(vertex) = 62.3) -->
       [(8, 7): 15.9]
       [(8, 3): 31.52]
vertex (id = 9, label = v9, weight(vertex) = 15.9) -->
       [(9, 0): 1.0]
```

```
Export matrix of current graph

    0      1      2      3      4      5      6      7      8

0   x      x      x      x      x      x      x      x      1.00

1   x      x      x      15.90  x      1.00   x      x      x

2   x      x      x      35.20  x      x      x      x      x

3   x      15.90  35.20  x      x      x      x      31.52  x

4   x      x      x      x      x      x      x      x      x

5   x      1.00   x      x      x      x      x      x      x

6   x      x      x      x      x      x      x      15.90  x

7   x      x      x      31.52  x      x      15.90  x      x

8   1.00   x      x      x      x      x      x      x      x



Remove a vertex from graph with id 8

Adjacency List
vertex (id = 0, label = v0, weight(vertex) = 52.3) -->
       [(0, 9): 1.0]
vertex (id = 1, label = v1, weight(vertex) = 12.9) -->
       [(1, 3): 15.9]
       [(1, 5): 1.0]
vertex (id = 2, label = v2, weight(vertex) = 0.9) -->
       [(2, 3): 35.2]
vertex (id = 3, label = v3, weight(vertex) = 67.0) -->
       [(3, 1): 15.9]
       [(3, 2): 35.2]
vertex (id = 4, label = v4, weight(vertex) = 11.31) -->
vertex (id = 5, label = v5, weight(vertex) = 22.34) -->
       [(5, 1): 1.0]
vertex (id = 7, label = v7, weight(vertex) = 71.3) -->
vertex (id = 9, label = v9, weight(vertex) = 15.9) -->
       [(9, 0): 1.0]
-----------------size = 8
```

```
Remove an edge between 1 and 5

Adjacency List
vertex (id = 0, label = v0, weight(vertex) = 52.3) -->
       [(0, 9): 1.0]
vertex (id = 1, label = v1, weight(vertex) = 12.9) -->
       [(1, 3): 15.9]
vertex (id = 2, label = v2, weight(vertex) = 0.9) -->
       [(2, 3): 35.2]
vertex (id = 3, label = v3, weight(vertex) = 67.0) -->
       [(3, 1): 15.9]
       [(3, 2): 35.2]
vertex (id = 4, label = v4, weight(vertex) = 11.31) -->
vertex (id = 5, label = v5, weight(vertex) = 22.34) -->
vertex (id = 7, label = v7, weight(vertex) = 71.3) -->
vertex (id = 9, label = v9, weight(vertex) = 15.9) -->
       [(9, 0): 1.0]
----------------size = 8


Remove an edge between 1 and 3

Adjacency List
vertex (id = 0, label = v0, weight(vertex) = 52.3) -->
       [(0, 9): 1.0]
vertex (id = 1, label = v1, weight(vertex) = 12.9) -->
vertex (id = 2, label = v2, weight(vertex) = 0.9) -->
       [(2, 3): 35.2]
vertex (id = 3, label = v3, weight(vertex) = 67.0) -->
       [(3, 2): 35.2]
vertex (id = 4, label = v4, weight(vertex) = 11.31) -->
vertex (id = 5, label = v5, weight(vertex) = 22.34) -->
vertex (id = 7, label = v7, weight(vertex) = 71.3) -->
vertex (id = 9, label = v9, weight(vertex) = 15.9) -->
       [(9, 0): 1.0]
```

```
Remove an edge between 2 and 3

Adjacency List
vertex (id = 0, label = v0, weight(vertex) = 52.3) -->
       [(0, 9): 1.0]
vertex (id = 1, label = v1, weight(vertex) = 12.9) -->
vertex (id = 2, label = v2, weight(vertex) = 0.9) -->
vertex (id = 3, label = v3, weight(vertex) = 67.0) -->
vertex (id = 4, label = v4, weight(vertex) = 11.31) -->
vertex (id = 5, label = v5, weight(vertex) = 22.34) -->
vertex (id = 7, label = v7, weight(vertex) = 71.3) -->
vertex (id = 9, label = v9, weight(vertex) = 15.9) -->
       [(9, 0): 1.0]


Remove an edge between 0 and 9

Adjacency List
vertex (id = 0, label = v0, weight(vertex) = 52.3) -->
vertex (id = 1, label = v1, weight(vertex) = 12.9) -->
vertex (id = 2, label = v2, weight(vertex) = 0.9) -->
vertex (id = 3, label = v3, weight(vertex) = 67.0) -->
vertex (id = 4, label = v4, weight(vertex) = 11.31) -->
vertex (id = 5, label = v5, weight(vertex) = 22.34) -->
vertex (id = 7, label = v7, weight(vertex) = 71.3) -->
vertex (id = 9, label = v9, weight(vertex) = 15.9) -->


Try to remove unexisted edge

Adjacency List
vertex (id = 0, label = v0, weight(vertex) = 52.3) -->
vertex (id = 1, label = v1, weight(vertex) = 12.9) -->
vertex (id = 2, label = v2, weight(vertex) = 0.9) -->
vertex (id = 3, label = v3, weight(vertex) = 67.0) -->
vertex (id = 4, label = v4, weight(vertex) = 11.31) -->
vertex (id = 5, label = v5, weight(vertex) = 22.34) -->
vertex (id = 7, label = v7, weight(vertex) = 71.3) -->
vertex (id = 9, label = v9, weight(vertex) = 15.9) -->
```

```
Insert arbitrary id vertices

Adjacency List
vertex (id = 0, label = v0, weight(vertex) = 52.3) -->
vertex (id = 1, label = v1, weight(vertex) = 12.9) -->
vertex (id = 2, label = v2, weight(vertex) = 0.9) -->
vertex (id = 3, label = v3, weight(vertex) = 67.0) -->
vertex (id = 4, label = v4, weight(vertex) = 11.31) -->
vertex (id = 5, label = v5, weight(vertex) = 22.34) -->
vertex (id = 7, label = v7, weight(vertex) = 71.3) -->
vertex (id = 9, label = v9, weight(vertex) = 15.9) -->
vertex (id = 19, label = v13, weight(vertex) = 12.9) -->
vertex (id = 20, label = v12, weight(vertex) = 0.9) -->
vertex (id = 16, label = v14, weight(vertex) = 67.0) -->
vertex (id = 31, label = v15, weight(vertex) = 11.31) -->
vertex (id = 24, label = v16, weight(vertex) = 22.34) -->
vertex (id = 22, label = v17, weight(vertex) = 12.13) -->
vertex (id = 21, label = v18, weight(vertex) = 71.3) -->
vertex (id = 37, label = v19, weight(vertex) = 62.3) -->
vertex (id = 38, label = v19, weight(vertex) = 15.9) -->
vertex (id = 17, label = v19, weight(vertex) = 22.39) -->
----------------size = 18
```

```
Vertices:

id = 0, label = v0, weight = 52.3
pairs: {}
id = 1, label = v1, weight = 12.9
pairs: {}
id = 2, label = v2, weight = 0.9
pairs: {}
id = 3, label = v3, weight = 67.0
pairs: {}
id = 4, label = v4, weight = 11.31
pairs: {}
id = 5, label = v5, weight = 22.34
pairs: {}
id = 7, label = v7, weight = 71.3
pairs: {}
id = 9, label = v9, weight = 15.9
pairs: {}
id = 19, label = v13, weight = 12.9
pairs: {}
id = 20, label = v12, weight = 0.9
pairs: {}
id = 16, label = v14, weight = 67.0
pairs: {}
id = 31, label = v15, weight = 11.31
pairs: {}
id = 24, label = v16, weight = 22.34
pairs: {}
id = 22, label = v17, weight = 12.13
pairs: {}
id = 21, label = v18, weight = 71.3
pairs: {}
id = 37, label = v19, weight = 62.3
pairs: {}
id = 38, label = v19, weight = 15.9
pairs: {}
id = 17, label = v19, weight = 22.39
pairs: {}
```

```
Add edges to the graph

Adjacency List
vertex (id = 0, label = v0, weight(vertex) = 52.3) -->
        [(0, 16): 6.5]
vertex (id = 1, label = v1, weight(vertex) = 12.9) -->
vertex (id = 2, label = v2, weight(vertex) = 0.9) -->
vertex (id = 3, label = v3, weight(vertex) = 67.0) -->
        [(3, 9): 7.31]
vertex (id = 4, label = v4, weight(vertex) = 11.31) -->
vertex (id = 5, label = v5, weight(vertex) = 22.34) -->
vertex (id = 7, label = v7, weight(vertex) = 71.3) -->
vertex (id = 9, label = v9, weight(vertex) = 15.9) -->
        [(9, 3): 7.31]
vertex (id = 19, label = v13, weight(vertex) = 12.9) -->
vertex (id = 20, label = v12, weight(vertex) = 0.9) -->
        [(20, 24): 8.16]
vertex (id = 16, label = v14, weight(vertex) = 67.0) -->
        [(16, 37): 41.3]
        [(16, 0): 6.5]
vertex (id = 31, label = v15, weight(vertex) = 11.31) -->
vertex (id = 24, label = v16, weight(vertex) = 22.34) -->
        [(24, 20): 8.16]
vertex (id = 22, label = v17, weight(vertex) = 12.13) -->
        [(22, 37): 10.6]
vertex (id = 21, label = v18, weight(vertex) = 71.3) -->
vertex (id = 37, label = v19, weight(vertex) = 62.3) -->
        [(37, 22): 10.6]
        [(37, 16): 41.3]
vertex (id = 38, label = v19, weight(vertex) = 15.9) -->
vertex (id = 17, label = v19, weight(vertex) = 22.39) -->
Remove vertices which have label v19

Adjacency List
vertex (id = 0, label = v0, weight(vertex) = 52.3) -->
        [(0, 16): 6.5]
vertex (id = 1, label = v1, weight(vertex) = 12.9) -->
vertex (id = 2, label = v2, weight(vertex) = 0.9) -->
vertex (id = 3, label = v3, weight(vertex) = 67.0) -->
        [(3, 9): 7.31]
vertex (id = 4, label = v4, weight(vertex) = 11.31) -->
vertex (id = 5, label = v5, weight(vertex) = 22.34) -->
vertex (id = 7, label = v7, weight(vertex) = 71.3) -->
vertex (id = 9, label = v9, weight(vertex) = 15.9) -->
        [(9, 3): 7.31]
vertex (id = 19, label = v13, weight(vertex) = 12.9) -->
vertex (id = 20, label = v12, weight(vertex) = 0.9) -->
        [(20, 24): 8.16]
vertex (id = 16, label = v14, weight(vertex) = 67.0) -->
        [(16, 0): 6.5]
vertex (id = 31, label = v15, weight(vertex) = 11.31) -->
vertex (id = 24, label = v16, weight(vertex) = 22.34) -->
        [(24, 20): 8.16]
vertex (id = 22, label = v17, weight(vertex) = 12.13) -->
vertex (id = 21, label = v18, weight(vertex) = 71.3) -->
```

```
Vertices:

id = 0, label = v0, weight = 52.3
pairs: {}
id = 1, label = v1, weight = 12.9
pairs: {}
id = 2, label = v2, weight = 0.9
pairs: {}
id = 3, label = v3, weight = 67.0
pairs: {}
id = 4, label = v4, weight = 11.31
pairs: {}
id = 5, label = v5, weight = 22.34
pairs: {}
id = 7, label = v7, weight = 71.3
pairs: {}
id = 9, label = v9, weight = 15.9
pairs: {}
id = 19, label = v13, weight = 12.9
pairs: {}
id = 20, label = v12, weight = 0.9
pairs: {}
id = 16, label = v14, weight = 67.0
pairs: {}
id = 31, label = v15, weight = 11.31
pairs: {}
id = 24, label = v16, weight = 22.34
pairs: {}
id = 22, label = v17, weight = 12.13
pairs: {}
id = 21, label = v18, weight = 71.3
pairs: {}
```

```
Add more edges to the graph

Adjacency List
vertex (id = 0, label = v0, weight(vertex) = 52.3) -->
       [(0, 16): 6.5]
vertex (id = 1, label = v1, weight(vertex) = 12.9) -->
       [(1, 21): 70.81]
       [(1, 2): 131.62]
vertex (id = 2, label = v2, weight(vertex) = 0.9) -->
       [(2, 1): 131.62]
vertex (id = 3, label = v3, weight(vertex) = 67.0) -->
       [(3, 9): 7.31]
vertex (id = 4, label = v4, weight(vertex) = 11.31) -->
vertex (id = 5, label = v5, weight(vertex) = 22.34) -->
vertex (id = 7, label = v7, weight(vertex) = 71.3) -->
vertex (id = 9, label = v9, weight(vertex) = 15.9) -->
       [(9, 3): 7.31]
       [(9, 19): 4.21]
vertex (id = 19, label = v13, weight(vertex) = 12.9) -->
       [(19, 9): 4.21]
vertex (id = 20, label = v12, weight(vertex) = 0.9) -->
       [(20, 24): 8.16]
vertex (id = 16, label = v14, weight(vertex) = 67.0) -->
       [(16, 0): 6.5]
       [(16, 24): 2.11]
vertex (id = 31, label = v15, weight(vertex) = 11.31) -->
       [(31, 22): 11.5]
vertex (id = 24, label = v16, weight(vertex) = 22.34) -->
       [(24, 20): 8.16]
       [(24, 16): 2.11]
vertex (id = 22, label = v17, weight(vertex) = 12.13) -->
       [(22, 31): 11.5]
vertex (id = 21, label = v18, weight(vertex) = 71.3) -->
       [(21, 1): 70.81]
```

```
Add Pairs to the vertices using iterator

Vertices:

id = 0, label = v0, weight = 52.3
pairs: {boosting=0, color=blue, length=5}
id = 1, label = v1, weight = 12.9
pairs: {boosting=3, color=pink, length=2}
id = 2, label = v2, weight = 0.9
pairs: {boosting=6, color=pink, length=5}
id = 3, label = v3, weight = 67.0
pairs: {boosting=9, color=red, length=6}
id = 4, label = v4, weight = 11.31
pairs: {boosting=12, color=blue, length=5}
id = 5, label = v5, weight = 22.34
pairs: {boosting=15, color=green, length=10}
id = 7, label = v7, weight = 71.3
pairs: {boosting=21, color=pink, length=14}
id = 9, label = v9, weight = 15.9
pairs: {boosting=27, color=red, length=18}
id = 19, label = v13, weight = 12.9
pairs: {boosting=57, color=pink, length=38}
id = 20, label = v12, weight = 0.9
pairs: {boosting=60, color=blue, length=5}
id = 16, label = v14, weight = 67.0
pairs: {boosting=48, color=blue, length=5}
id = 31, label = v15, weight = 11.31
pairs: {boosting=93, color=pink, length=62}
id = 24, label = v16, weight = 22.34
pairs: {boosting=72, color=blue, length=5}
id = 22, label = v17, weight = 12.13
pairs: {boosting=66, color=pink, length=5}
id = 21, label = v18, weight = 71.3
pairs: {boosting=63, color=red, length=42}
```

```
Adjacency List
vertex (id = 0, label = v0, weight(vertex) = 52.3) -->
       [(0, 16): 6.5]
vertex (id = 1, label = v1, weight(vertex) = 12.9) -->
       [(1, 21): 70.81]
       [(1, 2): 131.62]
vertex (id = 2, label = v2, weight(vertex) = 0.9) -->
       [(2, 1): 131.62]
vertex (id = 3, label = v3, weight(vertex) = 67.0) -->
       [(3, 9): 7.31]
vertex (id = 4, label = v4, weight(vertex) = 11.31) -->
vertex (id = 5, label = v5, weight(vertex) = 22.34) -->
vertex (id = 7, label = v7, weight(vertex) = 71.3) -->
vertex (id = 9, label = v9, weight(vertex) = 15.9) -->
       [(9, 3): 7.31]
       [(9, 19): 4.21]
vertex (id = 19, label = v13, weight(vertex) = 12.9) -->
       [(19, 9): 4.21]
vertex (id = 20, label = v12, weight(vertex) = 0.9) -->
       [(20, 24): 8.16]
vertex (id = 16, label = v14, weight(vertex) = 67.0) -->
       [(16, 0): 6.5]
       [(16, 24): 2.11]
vertex (id = 31, label = v15, weight(vertex) = 11.31) -->
       [(31, 22): 11.5]
vertex (id = 24, label = v16, weight(vertex) = 22.34) -->
       [(24, 20): 8.16]
       [(24, 16): 2.11]
vertex (id = 22, label = v17, weight(vertex) = 12.13) -->
       [(22, 31): 11.5]
vertex (id = 21, label = v18, weight(vertex) = 71.3) -->
       [(21, 1): 70.81]
```

```
Testing filter method

Filtered Graph(color, blue):
Adjacency List
vertex (id = 0, label = v0, weight(vertex) = 52.3) -->
       [(0, 16): 6.5]
vertex (id = 4, label = v4, weight(vertex) = 11.31) -->
vertex (id = 20, label = v12, weight(vertex) = 0.9) -->
       [(20, 24): 8.16]
vertex (id = 16, label = v14, weight(vertex) = 67.0) -->
       [(16, 0): 6.5]
       [(16, 24): 2.11]
vertex (id = 24, label = v16, weight(vertex) = 22.34) -->
       [(24, 16): 2.11]
       [(24, 20): 8.16]


Filtered Graph(length, 5)
Adjacency List
vertex (id = 0, label = v0, weight(vertex) = 52.3) -->
       [(0, 16): 6.5]
vertex (id = 2, label = v2, weight(vertex) = 0.9) -->
vertex (id = 4, label = v4, weight(vertex) = 11.31) -->
vertex (id = 20, label = v12, weight(vertex) = 0.9) -->
       [(20, 24): 8.16]
vertex (id = 16, label = v14, weight(vertex) = 67.0) -->
       [(16, 0): 6.5]
       [(16, 24): 2.11]
vertex (id = 24, label = v16, weight(vertex) = 22.34) -->
       [(24, 16): 2.11]
       [(24, 20): 8.16]
vertex (id = 22, label = v17, weight(vertex) = 12.13) -->
```

```
Testing other methods

Adjacency List
vertex (id = 0, label = v0, weight(vertex) = 52.3) -->
       [(0, 16): 6.5]
vertex (id = 1, label = v1, weight(vertex) = 12.9) -->
       [(1, 21): 70.81]
       [(1, 2): 131.62]
vertex (id = 2, label = v2, weight(vertex) = 0.9) -->
       [(2, 1): 131.62]
vertex (id = 3, label = v3, weight(vertex) = 67.0) -->
       [(3, 9): 7.31]
vertex (id = 4, label = v4, weight(vertex) = 11.31) -->
vertex (id = 5, label = v5, weight(vertex) = 22.34) -->
vertex (id = 7, label = v7, weight(vertex) = 71.3) -->
vertex (id = 9, label = v9, weight(vertex) = 15.9) -->
       [(9, 3): 7.31]
       [(9, 19): 4.21]
vertex (id = 19, label = v13, weight(vertex) = 12.9) -->
       [(19, 9): 4.21]
vertex (id = 20, label = v12, weight(vertex) = 0.9) -->
       [(20, 24): 8.16]
vertex (id = 16, label = v14, weight(vertex) = 67.0) -->
       [(16, 0): 6.5]
       [(16, 24): 2.11]
vertex (id = 31, label = v15, weight(vertex) = 11.31) -->
       [(31, 22): 11.5]
vertex (id = 24, label = v16, weight(vertex) = 22.34) -->
       [(24, 20): 8.16]
       [(24, 16): 2.11]
vertex (id = 22, label = v17, weight(vertex) = 12.13) -->
       [(22, 31): 11.5]
vertex (id = 21, label = v18, weight(vertex) = 71.3) -->
       [(21, 1): 70.81]

graph1.getVertex(9) returns: id = 9, label = v9, weight = 15.9
pairs: {boosting=27, color=red, length=18}

graph1.getVertex(30)(not exists) returns:
null

graph1.isEdge(9,19) = true

graph1.isEdge(5,16)(not exists) = false

graph1.getEdge(24,16) = [(24, 16): 2.11]

graph1.getEdge(4,7)(not exists) = null
```

```
Test a directed graph

Inserting vertices to directed graph

Vertices:

id = 0, label = v0, weight = 6.0
pairs: {color=purple}
id = 1, label = v1, weight = 13.31
pairs: {color=purple}
id = 2, label = v2, weight = 21.34
pairs: {color=purple}
id = 3, label = v3, weight = 12.13
pairs: {color=purple}
id = 4, label = v4, weight = 76.3
pairs: {color=purple}
id = 5, label = v5, weight = 7.3
pairs: {color=purple}
id = 6, label = v6, weight = 7.0
pairs: {boosting=6}
id = 7, label = v7, weight = 1.31
pairs: {boosting=6}
id = 8, label = v8, weight = 25.34
pairs: {}
id = 9, label = v9, weight = 11.43
pairs: {color=green}
id = 10, label = v10, weight = 72.4
pairs: {boosting=6}
id = 11, label = v11, weight = 65.3
pairs: {boosting=9}

graph2.isDirected() = true
```

```
Inserting edges to directed graph

Adjacency List
vertex (id = 0, label = v0, weight(vertex) = 6.0) -->
       [(0, 11): 12.6]
       [(0, 2): 6.5]
vertex (id = 1, label = v1, weight(vertex) = 13.31) -->
       [(1, 5): 25.12]
vertex (id = 2, label = v2, weight(vertex) = 21.34) -->
       [(2, 9): 46.3]
       [(2, 10): 7.31]
vertex (id = 3, label = v3, weight(vertex) = 12.13) -->
       [(3, 1): 12.31]
       [(3, 9): 11.31]
vertex (id = 4, label = v4, weight(vertex) = 76.3) -->
       [(4, 9): 51.16]
vertex (id = 5, label = v5, weight(vertex) = 7.3) -->
       [(5, 1): 16.5]
vertex (id = 6, label = v6, weight(vertex) = 7.0) -->
       [(6, 4): 24.12]
vertex (id = 7, label = v7, weight(vertex) = 1.31) -->
       [(7, 3): 41.3]
vertex (id = 8, label = v8, weight(vertex) = 25.34) -->
       [(8, 6): 10.6]
vertex (id = 9, label = v9, weight(vertex) = 11.43) -->
       [(9, 6): 56.42]
vertex (id = 10, label = v10, weight(vertex) = 72.4) -->
       [(10, 1): 5.3]
vertex (id = 11, label = v11, weight(vertex) = 65.3) -->
       [(11, 6): 1.0]
```

Export matrix of current graph

	0	1	2	3	4	5	6	7	8	9	10	11
0	x	x	6.50	x	x	x	x	x	x	x	x	12.60
1	x	x	x	x	x	25.12	x	x	x	x	x	x
2	x	x	x	x	x	x	x	x	x	46.30	7.31	x
3	x	12.31	x	x	x	x	x	x	x	11.31	x	x
4	x	x	x	x	x	x	x	x	x	51.16	x	x
5	x	16.50	x	x	x	x	x	x	x	x	x	x
6	x	x	x	x	24.12	x	x	x	x	x	x	x
7	x	x	x	41.30	x	x	x	x	x	x	x	x
8	x	x	x	x	x	x	10.60	x	x	x	x	x
9	x	x	x	x	x	x	56.42	x	x	x	x	x
10	x	5.30	x	x	x	x	x	x	x	x	x	x
11	x	x	x	x	x	x	1.00	x	x	x	x	x

```
Vertices:

id = 0, label = v0, weight = 6.0
pairs: {color=purple}
id = 1, label = v1, weight = 13.31
pairs: {color=purple}
id = 2, label = v2, weight = 21.34
pairs: {color=purple}
id = 3, label = v3, weight = 12.13
pairs: {color=purple}
id = 4, label = v4, weight = 76.3
pairs: {color=purple}
id = 5, label = v5, weight = 7.3
pairs: {color=purple}
id = 6, label = v6, weight = 7.0
pairs: {boosting=6}
id = 7, label = v7, weight = 1.31
pairs: {boosting=6}
id = 8, label = v8, weight = 25.34
pairs: {}
id = 9, label = v9, weight = 11.43
pairs: {color=green}
id = 10, label = v10, weight = 72.4
pairs: {boosting=6}
id = 11, label = v11, weight = 65.3
pairs: {boosting=9}

Filtered Graph(color, purple):
Adjacency List
vertex (id = 0, label = v0, weight(vertex) = 6.0) -->
        [(0, 2): 6.5]
vertex (id = 1, label = v1, weight(vertex) = 13.31) -->
        [(1, 5): 25.12]
vertex (id = 2, label = v2, weight(vertex) = 21.34) -->
vertex (id = 3, label = v3, weight(vertex) = 12.13) -->
        [(3, 1): 12.31]
vertex (id = 4, label = v4, weight(vertex) = 76.3) -->
vertex (id = 5, label = v5, weight(vertex) = 7.3) -->
        [(5, 1): 16.5]
```

```
Adjancecy Matrix of Filtered Graph(color, purple):
     0      1      2      3      4      5

0    x      x      6.50   x      x      x

1    x      x      x      x      x      25.12

2    x      x      x      x      x      x

3    x      12.31  x      x      x      x

4    x      x      x      x      x      x

5    x      16.50  x      x      x      x


Adjacency List
vertex (id = 0, label = v0, weight(vertex) = 6.0) -->
        [(0, 11): 12.6]
        [(0, 2): 6.5]
vertex (id = 1, label = v1, weight(vertex) = 13.31) -->
        [(1, 5): 25.12]
vertex (id = 2, label = v2, weight(vertex) = 21.34) -->
        [(2, 9): 46.3]
        [(2, 10): 7.31]
vertex (id = 3, label = v3, weight(vertex) = 12.13) -->
        [(3, 1): 12.31]
        [(3, 9): 11.31]
vertex (id = 4, label = v4, weight(vertex) = 76.3) -->
        [(4, 9): 51.16]
vertex (id = 5, label = v5, weight(vertex) = 7.3) -->
        [(5, 1): 16.5]
vertex (id = 6, label = v6, weight(vertex) = 7.0) -->
        [(6, 4): 24.12]
vertex (id = 7, label = v7, weight(vertex) = 1.31) -->
        [(7, 3): 41.3]
vertex (id = 8, label = v8, weight(vertex) = 25.34) -->
        [(8, 6): 10.6]
vertex (id = 9, label = v9, weight(vertex) = 11.43) -->
        [(9, 6): 56.42]
vertex (id = 10, label = v10, weight(vertex) = 72.4) -->
        [(10, 1): 5.3]
vertex (id = 11, label = v11, weight(vertex) = 65.3) -->
        [(11, 6): 1.0]
```

```
Remove some vertices from graph

Adjacency List
vertex (id = 0, label = v0, weight(vertex) = 6.0) -->
        [(0, 11): 12.6]
        [(0, 2): 6.5]
vertex (id = 2, label = v2, weight(vertex) = 21.34) -->
        [(2, 9): 46.3]
vertex (id = 3, label = v3, weight(vertex) = 12.13) -->
        [(3, 9): 11.31]
vertex (id = 5, label = v5, weight(vertex) = 7.3) -->
vertex (id = 6, label = v6, weight(vertex) = 7.0) -->
vertex (id = 7, label = v7, weight(vertex) = 1.31) -->
        [(7, 3): 41.3]
vertex (id = 8, label = v8, weight(vertex) = 25.34) -->
        [(8, 6): 10.6]
vertex (id = 9, label = v9, weight(vertex) = 11.43) -->
        [(9, 6): 56.42]
vertex (id = 11, label = v11, weight(vertex) = 65.3) -->
        [(11, 6): 1.0]


Remove some edges from graph

Adjacency List
vertex (id = 0, label = v0, weight(vertex) = 6.0) -->
        [(0, 11): 12.6]
vertex (id = 2, label = v2, weight(vertex) = 21.34) -->
        [(2, 9): 46.3]
vertex (id = 3, label = v3, weight(vertex) = 12.13) -->
        [(3, 9): 11.31]
vertex (id = 5, label = v5, weight(vertex) = 7.3) -->
vertex (id = 6, label = v6, weight(vertex) = 7.0) -->
vertex (id = 7, label = v7, weight(vertex) = 1.31) -->
        [(7, 3): 41.3]
vertex (id = 8, label = v8, weight(vertex) = 25.34) -->
        [(8, 6): 10.6]
vertex (id = 9, label = v9, weight(vertex) = 11.43) -->
vertex (id = 11, label = v11, weight(vertex) = 65.3) -->

__Test Results for MyGraph and DynamicGraph__
1- Insertion of edges and vertices are tested.
2- Removal of edges and vertices are tested.
3- Export Matrix and Filter methods are tested.
4- printGraph() and toString are tested.
5- Graph is tested after deletion and insertion.
6- Graph is tested for nonconsecutive ID's.
7- Iterator's are tested, and both directed, undirected graphs are tested.
8- Other DynamicGraph, and MyGraph methods are tested.
```

Traverse Graph Results

```
___Testing TraverseGraph Class(calculating difference)___

Vertices:

id = 0, label = v0, weight = 5.0
pairs: {}
id = 1, label = v1, weight = 15.0
pairs: {}
id = 2, label = v2, weight = 2.5
pairs: {}
id = 3, label = v3, weight = 3.0
pairs: {}
id = 4, label = v4, weight = 4.0
pairs: {}
id = 5, label = v5, weight = 10.3
pairs: {}
id = 6, label = v6, weight = 63.0
pairs: {}
id = 7, label = v7, weight = 40.0
pairs: {}
id = 8, label = v8, weight = 15.9
pairs: {}
```

```
Adjacency List
vertex (id = 0, label = v0, weight(vertex) = 5.0) -->
       [(0, 1): 10.0]
       [(0, 2): 15.0]
       [(0, 6): 70.0]
vertex (id = 1, label = v1, weight(vertex) = 15.0) -->
       [(1, 0): 10.0]
       [(1, 3): 20.0]
       [(1, 8): 70.0]
vertex (id = 2, label = v2, weight(vertex) = 2.5) -->
       [(2, 0): 15.0]
       [(2, 3): 30.0]
       [(2, 5): 15.0]
vertex (id = 3, label = v3, weight(vertex) = 3.0) -->
       [(3, 1): 20.0]
       [(3, 2): 30.0]
       [(3, 4): 65.0]
       [(3, 7): 10.0]
vertex (id = 4, label = v4, weight(vertex) = 4.0) -->
       [(4, 3): 65.0]
vertex (id = 5, label = v5, weight(vertex) = 10.3) -->
       [(5, 2): 15.0]
       [(5, 6): 50.0]
       [(5, 7): 25.0]
vertex (id = 6, label = v6, weight(vertex) = 63.0) -->
       [(6, 0): 70.0]
       [(6, 5): 50.0]
vertex (id = 7, label = v7, weight(vertex) = 40.0) -->
       [(7, 3): 10.0]
       [(7, 5): 25.0]
vertex (id = 8, label = v8, weight(vertex) = 15.9) -->
       [(8, 1): 70.0]

Start id is 0 for traversals

Distance(BFS) - Distance(DFS) = 10.0
```

```
Test for another graph

Vertices:

id = 0, label = v0, weight = 15.0
pairs: {}
id = 1, label = v1, weight = 25.0
pairs: {}
id = 2, label = v2, weight = 2.3
pairs: {}
id = 3, label = v3, weight = 4.0
pairs: {}
id = 4, label = v4, weight = 9.0
pairs: {}
id = 5, label = v5, weight = 10.13
pairs: {}
id = 6, label = v6, weight = 23.0
pairs: {}
id = 7, label = v7, weight = 50.0
pairs: {}
id = 8, label = v8, weight = 25.9
pairs: {}
id = 9, label = v9, weight = 135.9
pairs: {}
```

```
Adjacency List
vertex (id = 0, label = v0, weight(vertex) = 15.0) -->
       [(0, 1): 10.0]
       [(0, 5): 45.0]
       [(0, 6): 25.0]
vertex (id = 1, label = v1, weight(vertex) = 25.0) -->
       [(1, 0): 10.0]
       [(1, 2): 20.0]
       [(1, 4): 35.0]
vertex (id = 2, label = v2, weight(vertex) = 2.3) -->
       [(2, 1): 20.0]
       [(2, 4): 30.0]
       [(2, 6): 9.0]
vertex (id = 3, label = v3, weight(vertex) = 4.0) -->
       [(3, 4): 20.0]
       [(3, 9): 30.0]
       [(3, 8): 10.0]
       [(3, 5): 15.0]
vertex (id = 4, label = v4, weight(vertex) = 9.0) -->
       [(4, 1): 35.0]
       [(4, 2): 30.0]
       [(4, 3): 20.0]
vertex (id = 5, label = v5, weight(vertex) = 10.13) -->
       [(5, 0): 45.0]
       [(5, 3): 15.0]
       [(5, 7): 70.0]
vertex (id = 6, label = v6, weight(vertex) = 23.0) -->
       [(6, 0): 25.0]
       [(6, 2): 9.0]
vertex (id = 7, label = v7, weight(vertex) = 50.0) -->
       [(7, 5): 70.0]
       [(7, 8): 50.0]
vertex (id = 8, label = v8, weight(vertex) = 25.9) -->
       [(8, 3): 10.0]
       [(8, 7): 50.0]
       [(8, 9): 15.0]
vertex (id = 9, label = v9, weight(vertex) = 135.9) -->
       [(9, 3): 30.0]
       [(9, 8): 15.0]

Start id is 4 for traversals

Distance(BFS) - Distance(DFS) = -40.0
```

BoostedDijkstra Class Results

```
___Testing BoostedDijkstra Class___

Vertices:

id = 0, label = v0, weight = 17.0
pairs: {}
id = 1, label = v1, weight = 11.21
pairs: {boosting=35.0}
id = 2, label = v2, weight = 42.34
pairs: {boosting=20.0}
id = 3, label = v3, weight = 13.33
pairs: {boosting=5.0}
id = 4, label = v4, weight = 16.5
pairs: {}

Adjacency List
vertex (id = 0, label = v0, weight(vertex) = 17.0) -->
       [(0, 1): 50.0]
       [(0, 2): 10.0]
       [(0, 3): 20.0]
vertex (id = 1, label = v1, weight(vertex) = 11.21) -->
       [(1, 4): 10.0]
       [(1, 2): 10.0]
vertex (id = 2, label = v2, weight(vertex) = 42.34) -->
       [(2, 3): 25.0]
vertex (id = 3, label = v3, weight(vertex) = 13.33) -->
       [(3, 4): 30.0]
vertex (id = 4, label = v4, weight(vertex) = 16.5) -->


Starting id is 0

Predecessors for each id = {1=0, 2=0, 3=2, 4=1}
Distances to each id = {1=50.0, 2=10.0, 3=15.0, 4=25.0}
```

```
Test for another graph

Vertices:

id = 0, label = v0, weight = 7.0
pairs: {}
id = 1, label = v1, weight = 1.31
pairs: {boosting=10.0}
id = 2, label = v2, weight = 2.34
pairs: {boosting=20.0}
id = 3, label = v3, weight = 12.13
pairs: {boosting=5.0}
id = 4, label = v4, weight = 7.3
pairs: {boosting=5.0}
id = 5, label = v5, weight = 11.1
pairs: {boosting=30.0}
id = 6, label = v6, weight = 27.34
pairs: {boosting=10.0}
id = 7, label = v7, weight = 2.13
pairs: {boosting=30.0}
id = 8, label = v8, weight = 72.3
pairs: {}

Adjacency List
vertex (id = 0, label = v0, weight(vertex) = 7.0) -->
       [(0, 1): 20.0]
       [(0, 2): 20.0]
       [(0, 3): 10.0]
       [(0, 5): 15.0]
vertex (id = 1, label = v1, weight(vertex) = 1.31) -->
       [(1, 2): 15.0]
       [(1, 6): 30.0]
       [(1, 4): 50.0]
vertex (id = 2, label = v2, weight(vertex) = 2.34) -->
       [(2, 8): 90.0]
vertex (id = 3, label = v3, weight(vertex) = 12.13) -->
       [(3, 8): 40.0]
       [(3, 5): 25.0]
       [(3, 4): 45.0]
vertex (id = 4, label = v4, weight(vertex) = 7.3) -->
       [(4, 5): 60.0]
       [(4, 7): 10.0]
vertex (id = 5, label = v5, weight(vertex) = 11.1) -->
       [(5, 6): 10.0]
vertex (id = 6, label = v6, weight(vertex) = 27.34) -->
       [(6, 2): 30.0]
vertex (id = 7, label = v7, weight(vertex) = 2.13) -->
       [(7, 8): 80.0]
vertex (id = 8, label = v8, weight(vertex) = 72.3) -->
```

```
Starting id is 0

Predecessors for each id = {1=0, 2=0, 3=0, 4=3, 5=0, 6=5, 7=4, 8=3}
Distances to each id = {1=20.0, 2=20.0, 3=10.0, 4=50.0, 5=15.0, 6=1.0, 7=55.0, 8=45.0}
```

“make” command compiles, and “make run” command runs the program.