GIT Department of Computer Engineering

CSE 222/505 - Spring 2021
Homework 8 # Report

Okan Torun

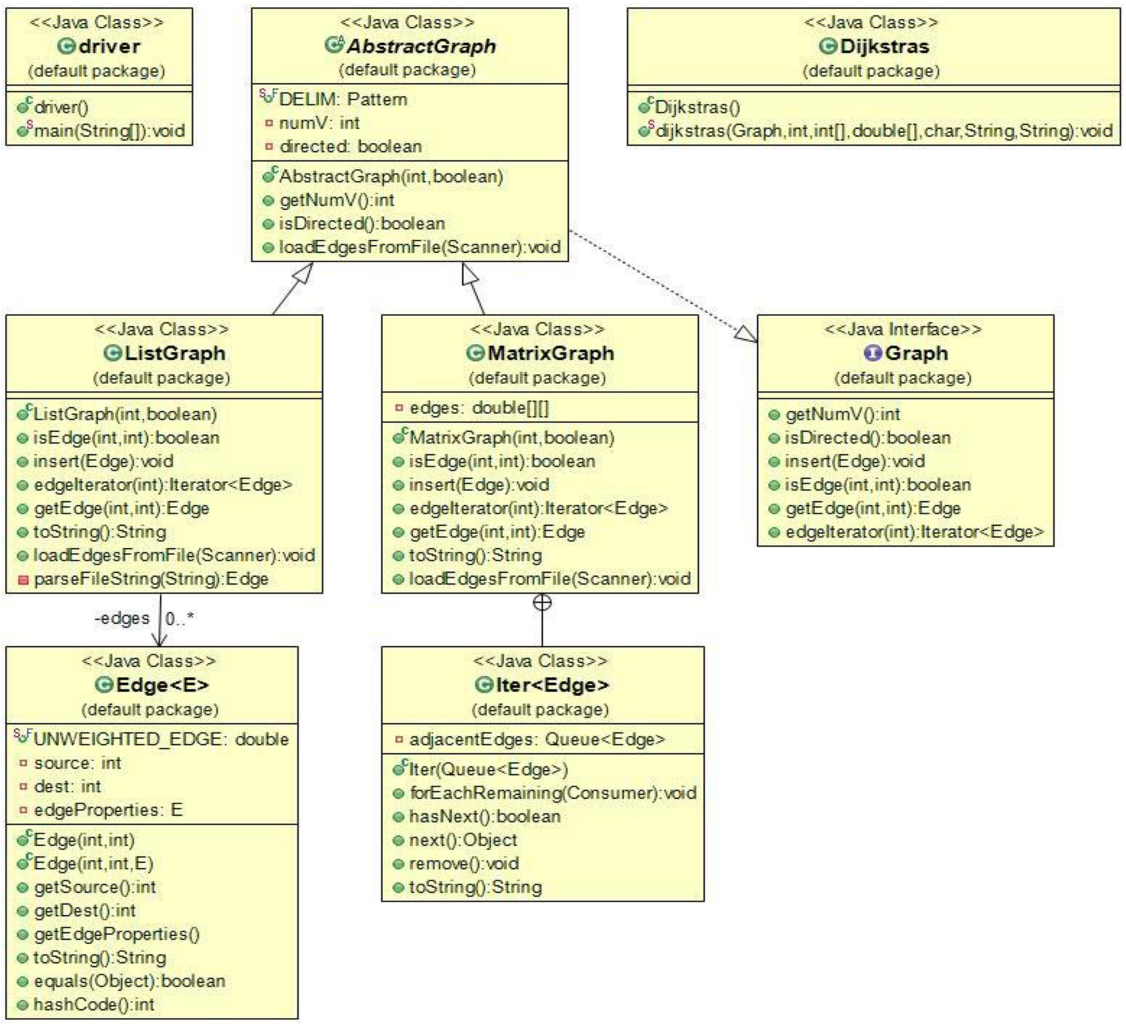
1801042662

PART 1

1)SYSTEM REQUIREMENTS

The system finds the shortest path of a entered graph with Dijkstra's algorithm. The user can determine the Graph type, path properties and the operators you want to use when calculating the path. The graph type is Matrix graph and List Graph. There are time, distance and quality properties when looking at the path properties. Calculation operators are Addition, multiplacation and '*' operators. The shortest path can be found with Dijkstra's algorithm by using these properties in the generated graph.

2) CLASS DIAGRAMS

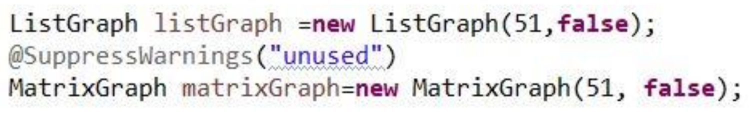


3)PROBLEM SOLUTION APPROACH

I used the Dijkstra implementation from the book to find the shortest path to the given graph with Dijkstra's algorithm. I have created a parameter that the user enters to determine the chart type. By specifying the type desired by the user, the path can be calculated with a list graph or matrix graphic application. Again, the desired calculation type is applied with the operator determined by the user. .The most important issue was the use of different features of the road. Since I cannot addition with the generic types I have created, the shortest route is created with the features requested by the user under different conditions.

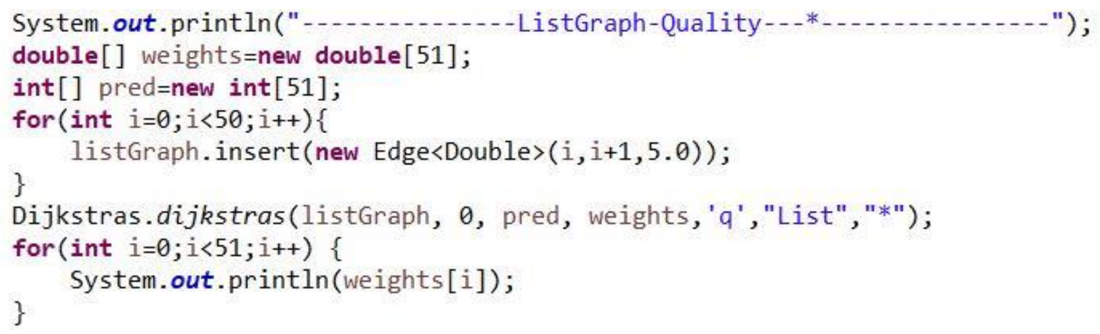
4)TEST CASES

I created list graph and matrix graph with 51 vertices.



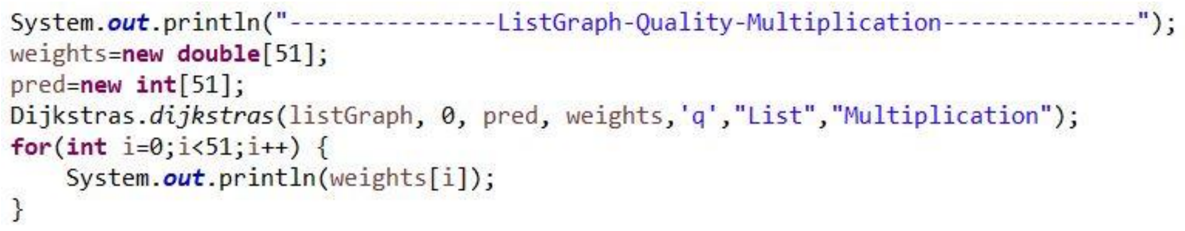
```
ListGraph listGraph =new ListGraph(51,false);
@SuppressWarnings("unused")
MatrixGraph matrixGraph=new MatrixGraph(51, false);
```

First I created edges on the List graph and then I calculated with Dijkstra's algorithm with Quality,List and '*' properties.



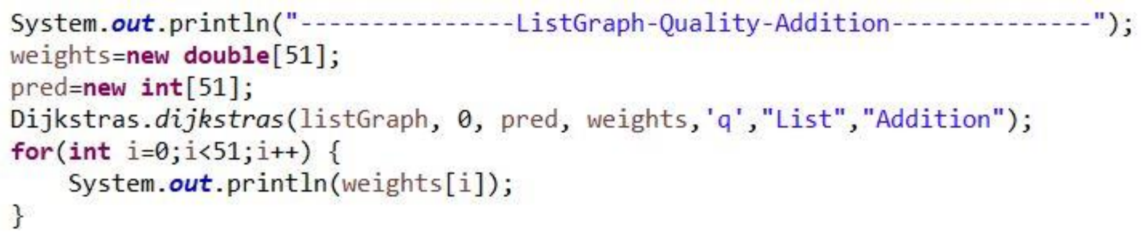
```
System.out.println("---------------ListGraph-Quality---*----------------");
double[] weights=new double[51];
int[] pred=new int[51];
for(int i=0;i<50;i++){
    listGraph.insert(new Edge<Double>(i,i+1,5.0));
}
Dijkstras.dijkstras(listGraph, 0, pred, weights,'q',"List","*");
for(int i=0;i<51;i++) {
    System.out.println(weights[i]);
}
```

Dijkstra's algorithm with Quality,List and Multiplication



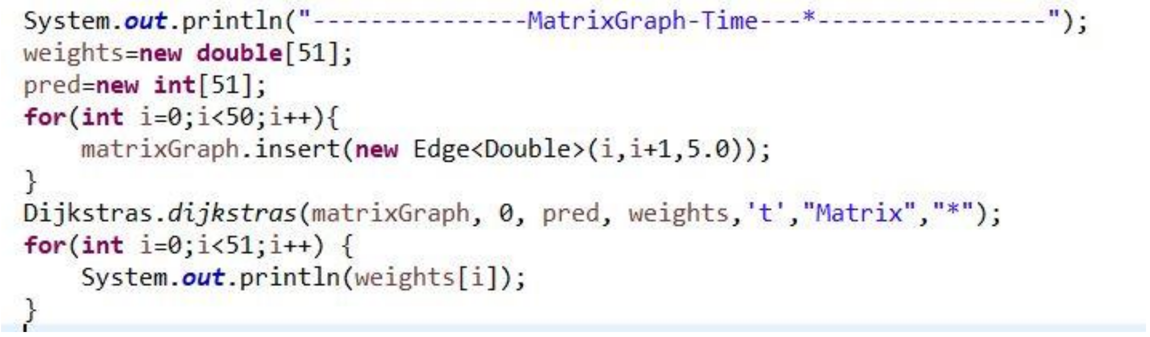
```
System.out.println("---------------ListGraph-Quality-Multiplication-------------");
weights=new double[51];
pred=new int[51];
Dijkstras.dijkstras(listGraph, 0, pred, weights,'q',"List","Multiplication");
for(int i=0;i<51;i++) {
    System.out.println(weights[i]);
}
```

Dijkstra's algorithm with Quality,List and Addition



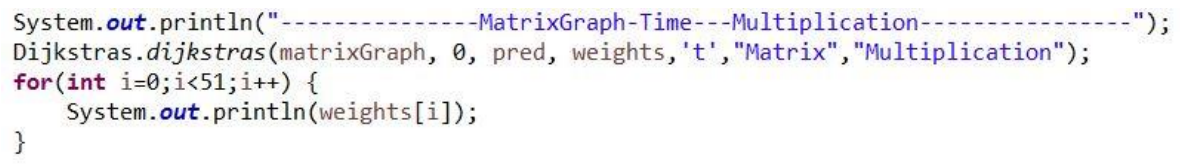
```
System.out.println("---------------ListGraph-Quality-Addition--------------");
weights=new double[51];
pred=new int[51];
Dijkstras.dijkstras(listGraph, 0, pred, weights,'q',"List","Addition");
for(int i=0;i<51;i++) {
    System.out.println(weights[i]);
}
```

I created edges on the matrix graph and then I calculated with Dijkstra's algorithm with Time,Matrix and '*' properties.



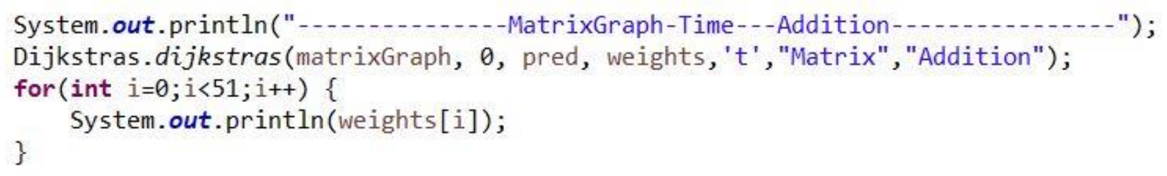
```
System.out.println("---------------MatrixGraph-Time---*----------------");
weights=new double[51];
pred=new int[51];
for(int i=0;i<50;i++){
    matrixGraph.insert(new Edge<Double>(i,i+1,5.0));
}
Dijkstras.dijkstras(matrixGraph, 0, pred, weights,'t',"Matrix","*");
for(int i=0;i<51;i++) {
    System.out.println(weights[i]);
}
```

Dijkstra's algorithm with Matrix,Time and Multiplication



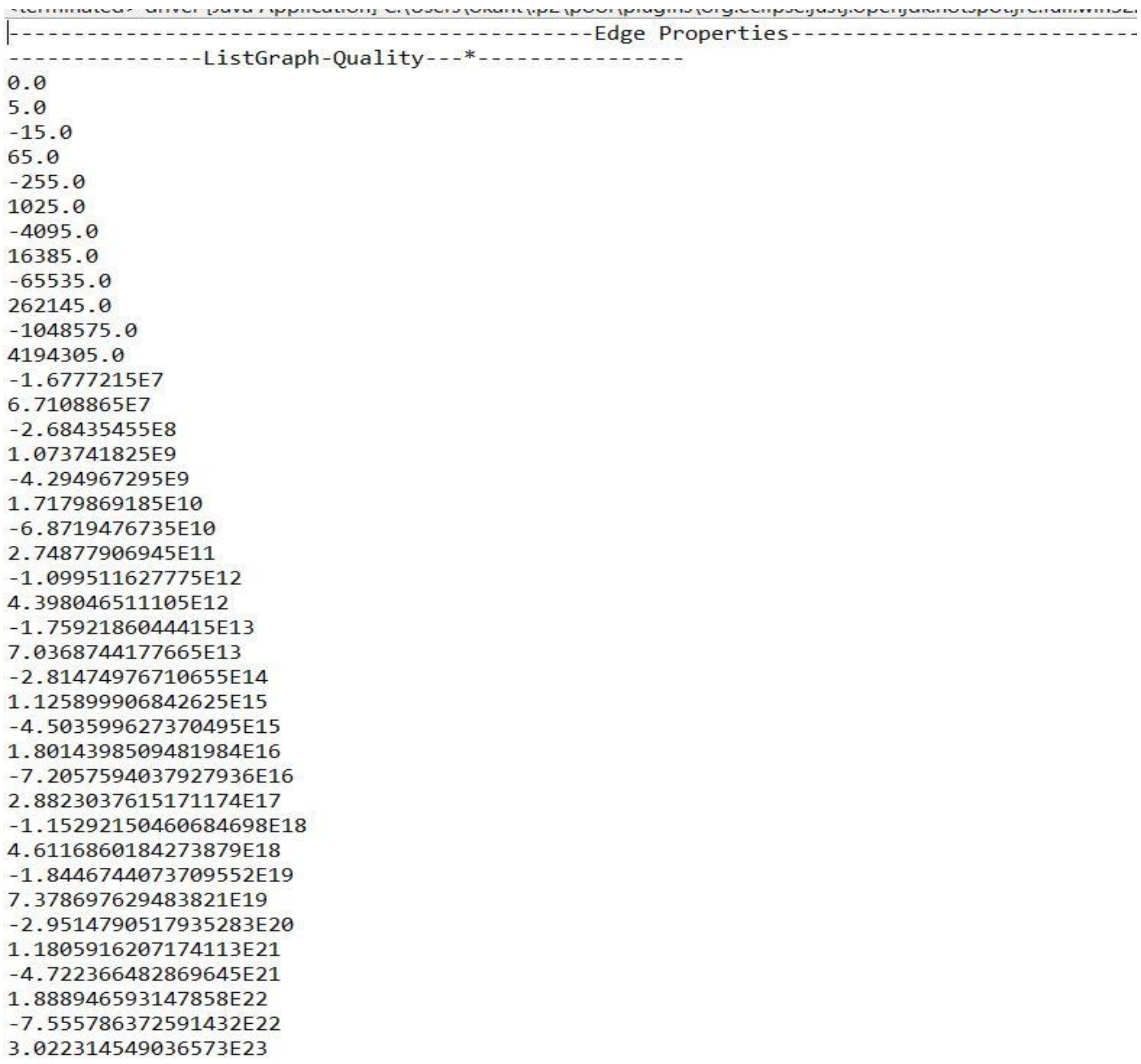
```
System.out.println("--------------MatrixGraph-Time---Multiplication---------------");
Dijkstras.dijkstras(matrixGraph, 0, pred, weights,'t',"Matrix","Multiplication");
for(int i=0;i<51;i++) {
    System.out.println(weights[i]);
}
```

Dijkstra's algorithm with Matrix,Time and Addition



```
System.out.println("---------------MatrixGraph-Time---Addition----------------");
Dijkstras.dijkstras(matrixGraph, 0, pred, weights,'t',"Matrix","Addition");
for(int i=0;i<51;i++) {
    System.out.println(weights[i]);
}
```

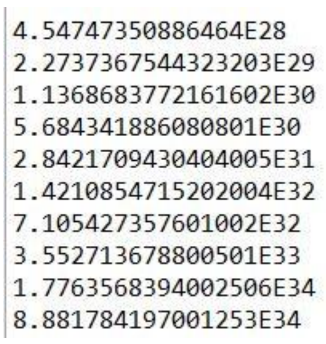
5)RUNNİNG AND RESULTS



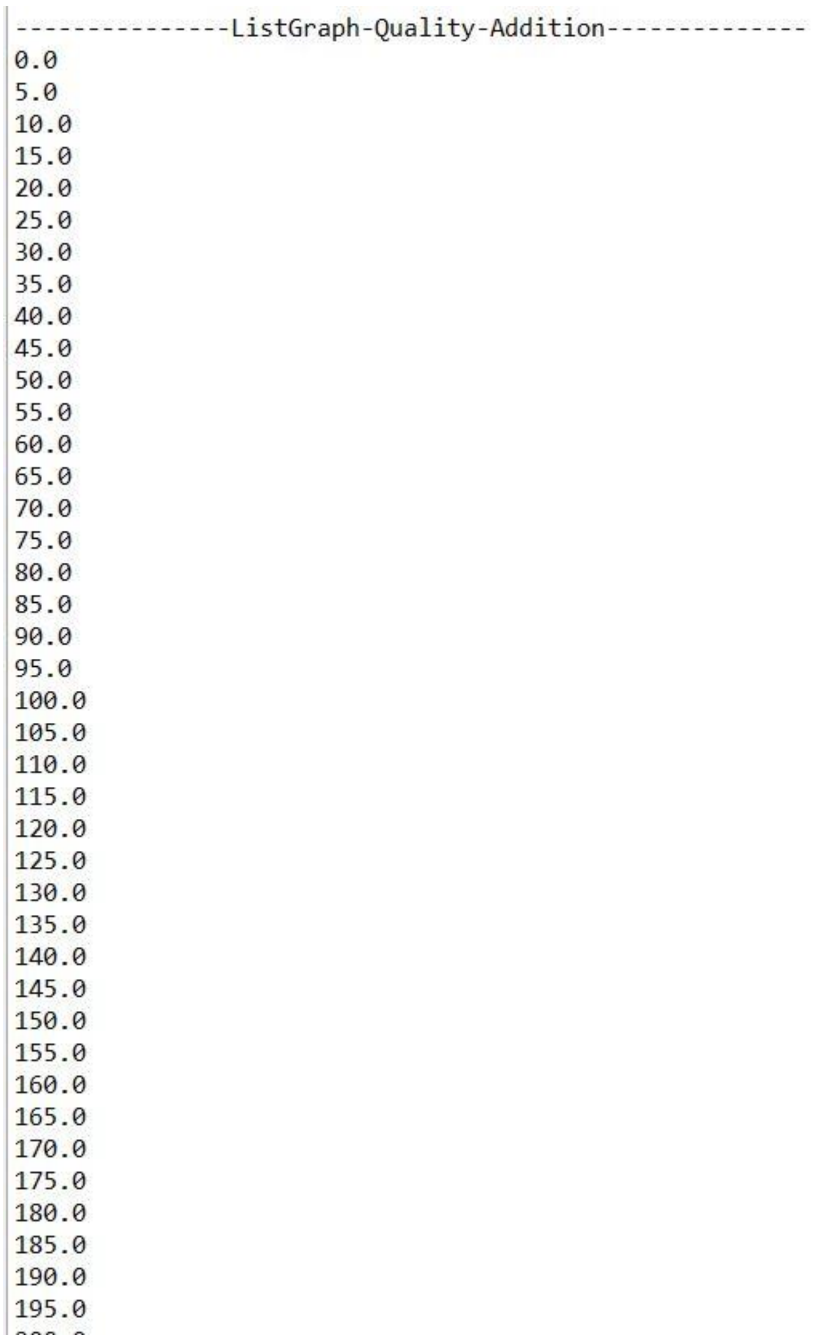
```
|---------------------------------------------Edge Properties---------------------------
---------------ListGraph-Quality---*----------------
0.0
5.0
-15.0
65.0
-255.0
1025.0
-4095.0
16385.0
-65535.0
262145.0
-1048575.0
4194305.0
-1.6777215E7
6.7108865E7
-2.68435455E8
1.073741825E9
-4.294967295E9
1.7179869185E10
-6.8719476735E10
2.74877906945E11
-1.099511627775E12
4.398046511105E12
-1.7592186044415E13
7.0368744177665E13
-2.81474976710655E14
1.125899906842625E15
-4.503599627370495E15
1.8014398509481984E16
-7.2057594037927936E16
2.8823037615171174E17
-1.15292150460684698E18
4.6116860184273879E18
-1.8446744073709552E19
7.378697629483821E19
-2.9514790517935283E20
1.1805916207174113E21
-4.722366482869645E21
1.888946593147858E22
-7.555786372591432E22
3.022314549036573E23
```

```
-1.2089258196146292E24
4.8357032784585167E24
-1.9342813113834067E25
7.737125245533627E25
-3.0948500982134507E26
1.2379400392853803E27
-4.951760157141521E27
1.9807040628566084E28
-7.922816251426434E28
3.1691265005705735E29
-1.2676506002282294E30
```

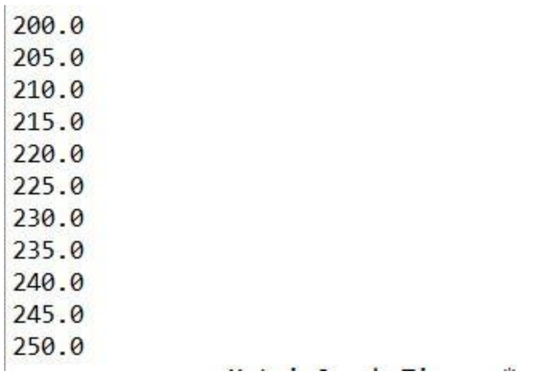
```
---------------ListGraph-Quality-Multiplication-------------
0.0
5.0
25.0
125.0
625.0
3125.0
15625.0
78125.0
390625.0
1953125.0
9765625.0
4.8828125E7
2.44140625E8
1.220703125E9
6.103515625E9
3.0517578125E10
1.52587890625E11
7.62939453125E11
3.814697265625E12
1.9073486328125E13
9.5367431640625E13
4.76837158203125E14
2.384185791015625E15
1.1920928955078124E16
5.960464477539062E16
2.9802322387695309E17
1.4901161193847544E18
7.4505805969238272E18
3.725290298461914E19
1.862645149230957E20
9.313225746154784E20
4.656612873077392E21
2.328306436538696E22
1.164153218269348E23
5.8207660913467404E23
2.9103830456733703E24
1.455191522836685E25
7.275957614183425E25
3.637978807091712E26
1.818989403545856E27
9.09494701772928E27
```



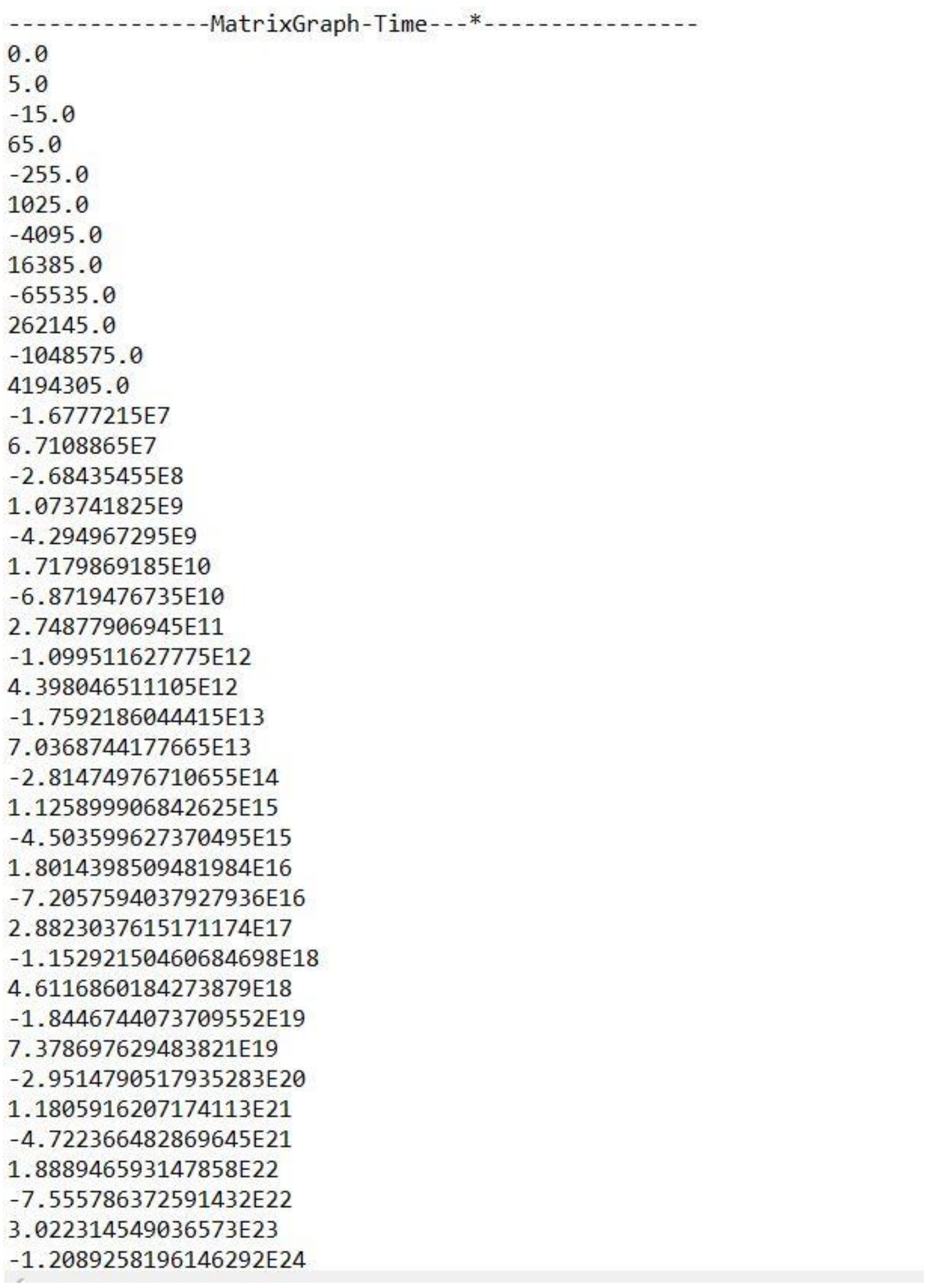
```
4.54747350886464E28
2.2737367544323203E29
1.1368683772161602E30
5.684341886080801E30
2.8421709430404005E31
1.4210854715202004E32
7.105427357601002E32
3.552713678800501E33
1.7763568394002506E34
8.881784197001253E34
```



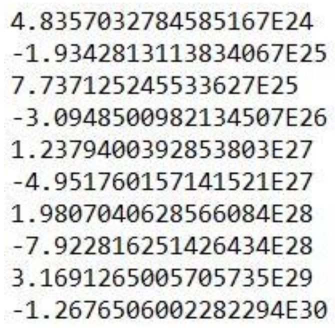
```
---------------ListGraph-Quality-Addition-------------
0.0
5.0
10.0
15.0
20.0
25.0
30.0
35.0
40.0
45.0
50.0
55.0
60.0
65.0
70.0
75.0
80.0
85.0
90.0
95.0
100.0
105.0
110.0
115.0
120.0
125.0
130.0
135.0
140.0
145.0
150.0
155.0
160.0
165.0
170.0
175.0
180.0
185.0
190.0
195.0
```



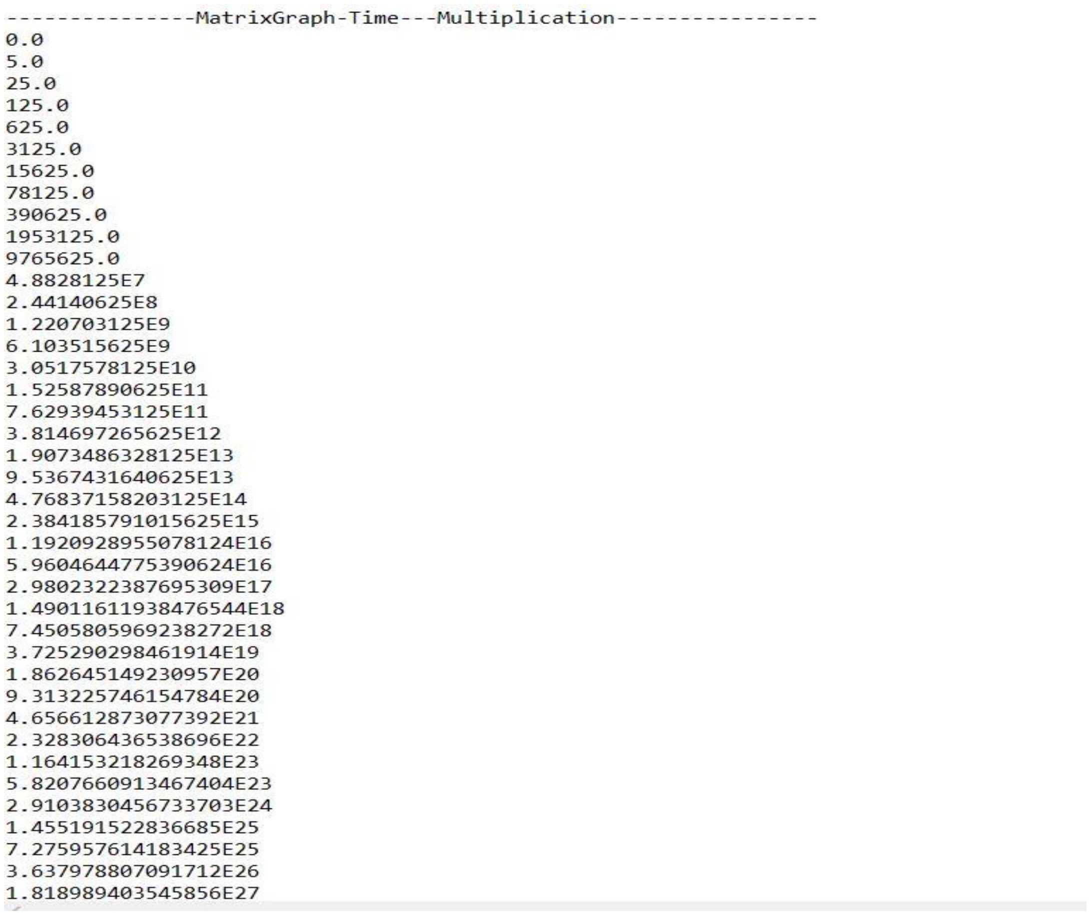
```
200.0
205.0
210.0
215.0
220.0
225.0
230.0
235.0
240.0
245.0
250.0
```



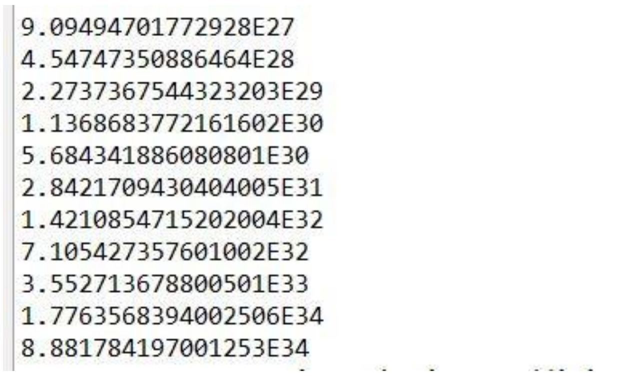
```
---------------MatrixGraph-Time---*----------------
0.0
5.0
-15.0
65.0
-255.0
1025.0
-4095.0
16385.0
-65535.0
262145.0
-1048575.0
4194305.0
-1.6777215E7
6.7108865E7
-2.68435455E8
1.073741825E9
-4.294967295E9
1.7179869185E10
-6.8719476735E10
2.74877906945E11
-1.099511627775E12
4.398046511105E12
-1.7592186044415E13
7.0368744177665E13
-2.81474976710655E14
1.125899906842625E15
-4.503599627370495E15
1.8014398509481984E16
-7.2057594037927936E16
2.8823037615171174E17
-1.15292150460684698E18
4.6116860184273879E18
-1.8446744073709552E19
7.378697629483821E19
-2.9514790517935283E20
1.1805916207174113E21
-4.722366482869645E21
1.888946593147858E22
-7.555786372591432E22
3.022314549036573E23
-1.2089258196146292E24
```



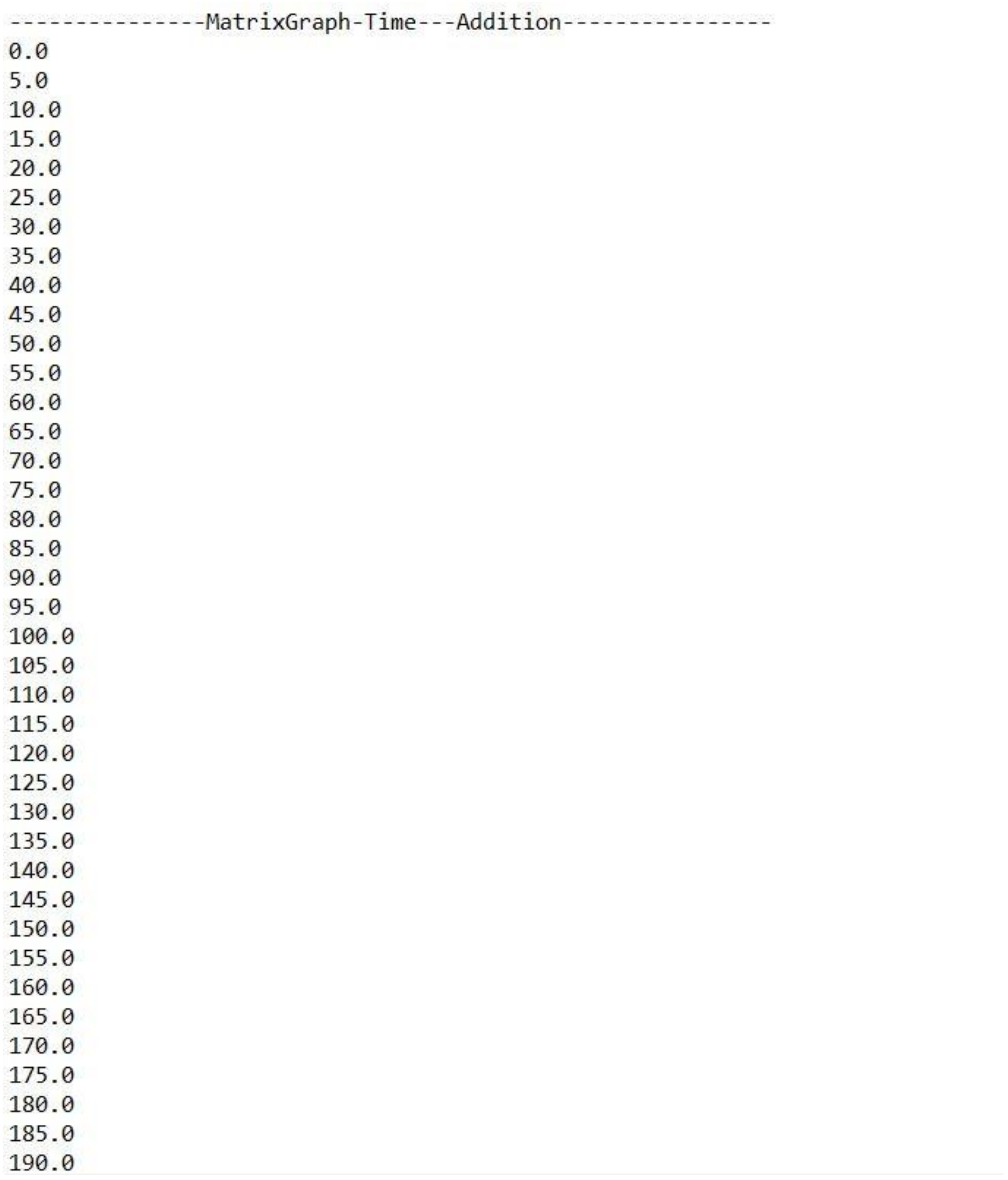
```
4.8357032784585167E24
-1.9342813113834067E25
7.737125245533627E25
-3.0948500982134507E26
1.2379400392853803E27
-4.951760157141521E27
1.9807040628566084E28
-7.922816251426434E28
3.1691265005705735E29
-1.2676506002282294E30
```



```
---------------MatrixGraph-Time---Multiplication----------------
0.0
5.0
25.0
125.0
625.0
3125.0
15625.0
78125.0
390625.0
1953125.0
9765625.0
4.8828125E7
2.44140625E8
1.220703125E9
6.103515625E9
3.0517578125E10
1.52587890625E11
7.62939453125E11
3.814697265625E12
1.9073486328125E13
9.5367431640625E13
4.76837158203125E14
2.384185791015625E15
1.1920928955078124E16
5.9604644775390624E16
2.9802322387695309E17
1.4901161193847654E18
7.4505805969238272E18
3.725290298461914E19
1.862645149230957E20
9.313225746154784E20
4.656612873077392E21
2.328306436538696E22
1.164153218269348E23
5.8207660913467404E23
2.9103830456733703E24
1.455191522836685E25
7.275957614183425E25
3.637978807091712E26
1.818989403545856E27
```



```
9.09494701772928E27
4.54747350886464E28
2.2737367544323203E29
1.1368683772161602E30
5.684341886080801E30
2.8421709430404005E31
1.4210854715202004E32
7.105427357601002E32
3.552713678800501E33
1.7763568394002506E34
8.881784197001253E34
```



```
---------------MatrixGraph-Time---Addition---------------
0.0
5.0
10.0
15.0
20.0
25.0
30.0
35.0
40.0
45.0
50.0
55.0
60.0
65.0
70.0
75.0
80.0
85.0
90.0
95.0
100.0
105.0
110.0
115.0
120.0
125.0
130.0
135.0
140.0
145.0
150.0
155.0
160.0
165.0
170.0
175.0
180.0
185.0
190.0
```

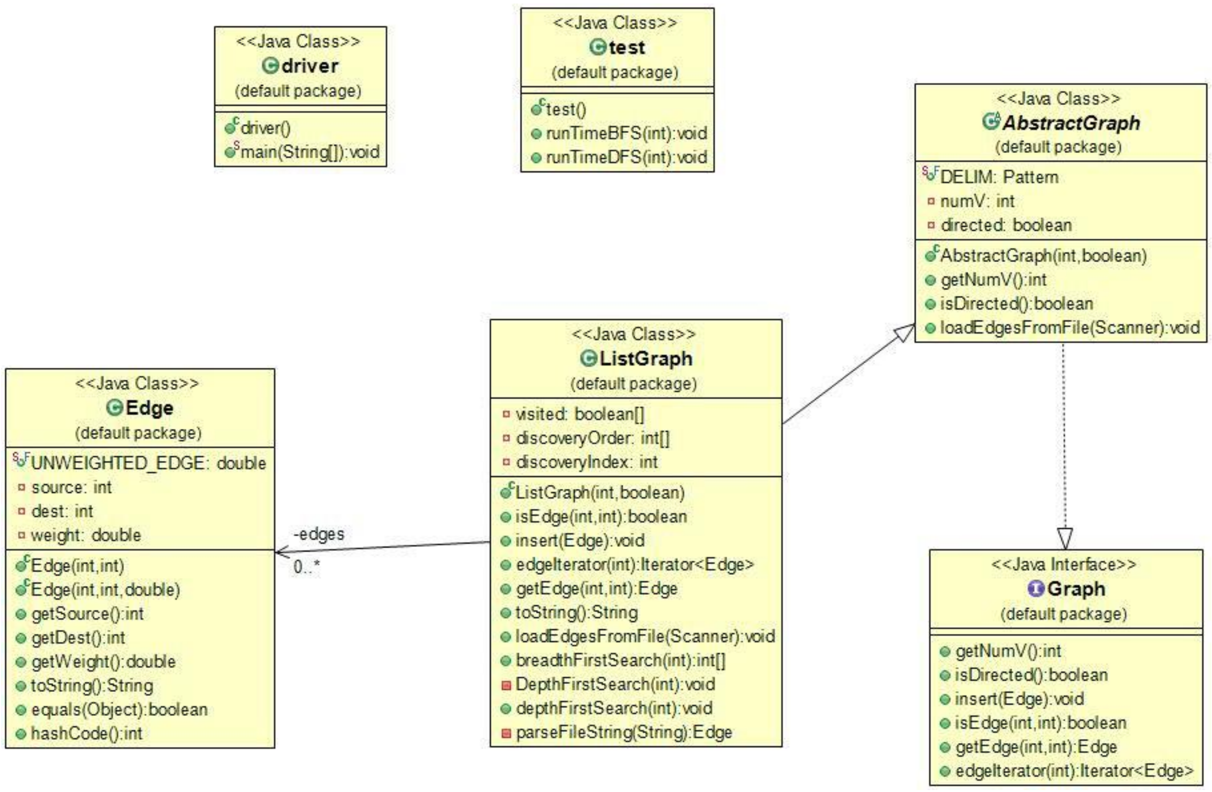
```
195.0
200.0
205.0
210.0
215.0
220.0
225.0
230.0
235.0
240.0
245.0
250.0
```

PART 2

1)SYSTEM REQUIREMENTS

Different sizes of graphs were created for the system. And the average time was calculated with the Breath first search and Depth first search searches of the graphs created in different sizes 10 times.And their results are shown on the graph.

2) CLASS DIAGRAMS



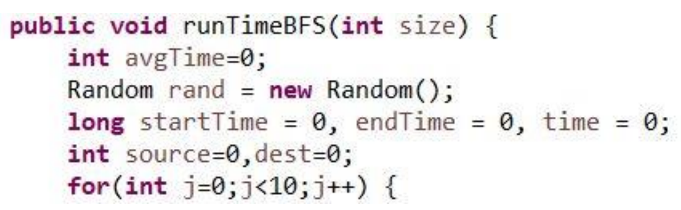
3)PROBLEM SOLUTION APPROACH

I first used the graph implementations in the book to navigate through the created graphs with breadth first search and depth first search. And I did this through list graph. Then listGraph

I implemented the DFS vs BFS methods in the class. I tested the running times of BFS and DFS on graphs of different sizes that I created in the driver code and showed the average times on the graph.

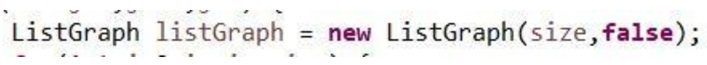
4)TEST CASES

I start the start and end points of the edges I will create. And I start the timer.



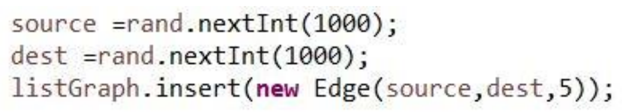
```java
public void runTimeBFS(int size) {
    int avgTime=0;
    Random rand = new Random();
    long startTime = 0, endTime = 0, time = 0;
    int source=0,dest=0;
    for(int j=0;j<10;j++) {
```

I am creating a list graph with as many vertices as the size I have determined.



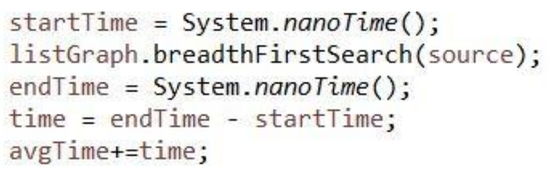
```java
ListGraph listGraph = new ListGraph(size,false);
```

I create an edge by specifying the source and destination points.



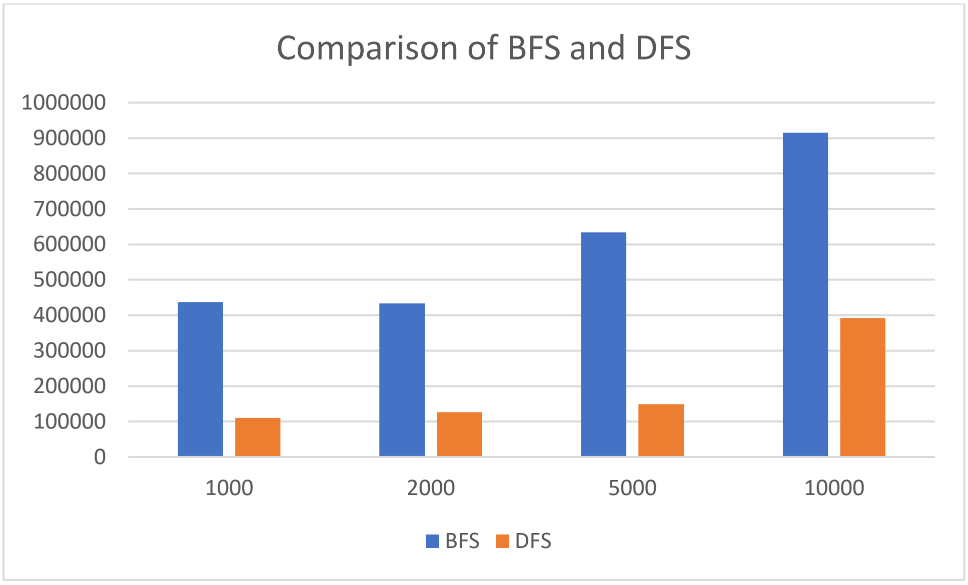
```java
source =rand.nextInt(1000);
dest =rand.nextInt(1000);
listGraph.insert(new Edge(source,dest,5));
```

I calculate the elapsed time while doing BFS.

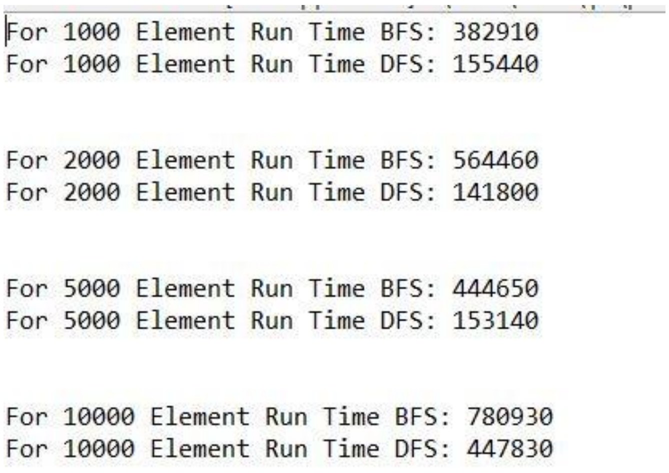


```java
startTime = System.nanoTime();
listGraph.breadthFirstSearch(source);
endTime = System.nanoTime();
time = endTime - startTime;
avgTime+=time;
```

5)RUNNİNG AND RESULTS



Compiling and running



```
For 1000 Element Run Time BFS: 382910
For 1000 Element Run Time DFS: 155440


For 2000 Element Run Time BFS: 564460
For 2000 Element Run Time DFS: 141800


For 5000 Element Run Time BFS: 444650
For 5000 Element Run Time DFS: 153140


For 10000 Element Run Time BFS: 780930
For 10000 Element Run Time DFS: 447830
```