GIT Department of Computer Engineering
CSE 222/505 - Spring 2021
Homework 8 Report

Emre SEZER
1901042640

SYSTEM REQUIREMENTS:

I created this homework on Windows 10 using terminal (Java Development Kit).My java version is 11.0.8 .You need to compile “driver.java” in order to test my homework.There are java files, a folder named “Javadoc” that includes javadoc files and report file in my homework. It may take take like 30 seconds to execute PART2. I put results of part2 to report.

PROBLEM SOLUTION APPROACH:

PART1:

I added time, quality and distance properties other than weight (All integers) to Edge class. In order to perform requested tasks is modified the MatrixGraph classs. My Dijkstra algorithm takes 2 extra integer inputs. One of them is selecting property of the Edge and other one is for selecting associative operation.

I couldn’t understand “Your method should run for any specified associative operator.” So, implemented a method called func(). This method contains 5 associative operations.With selection input user can select which operation to operate.

PART2:

I wrote nofccBreathFirstSearch() method for counting the number of connected components in a graph using BFS and nofccDepthFirstSearch() method for counting the number of connected components in a graph using DFS.

PART3:

I wrote importanceUtil() method for finding all possible paths from start to end vertexes. Then, i wrote importance() method that returns ArrayList<Double>.

This main method iterates for each u and w vertexes in the connected component and V.This method finds the importance of the each V.

TEST CASES:

PART1:

- Using Dijkstra Algorithm with Distance as Edge Property and (a * a) + (b * b) as operation with ListGraph.
- Using Dijkstra Algorithm with Using Time as Edge Property and a / (a + b) as operation with ListGraph
- Using Dijkstra Algorithm with Quality as Edge Property and (a + b + (a * b)) as operation with ListGraph
- Using Dijkstra Algorithm with Distance as Edge Property and (a * a) + (b * b) as operation with MatrixGraph.
- Using Dijkstra Algorithm with Using Time as Edge Property and a / (a + b) as operation with MatrixGraph
- Using Dijkstra Algorithm with Quality as Edge Property and (a + b + (a * b)) as operation with MatrixGraph

PART2:

I followed instructions on the pdf file.

PART3:

- Using Graph With Size Equals to 10 to Calcute Importance of Each Vertex in the Graph
- Using Graph With Size Equals to 5 to Calcute Importance of Each Vertex in the Graph

PART2 RESULTS:

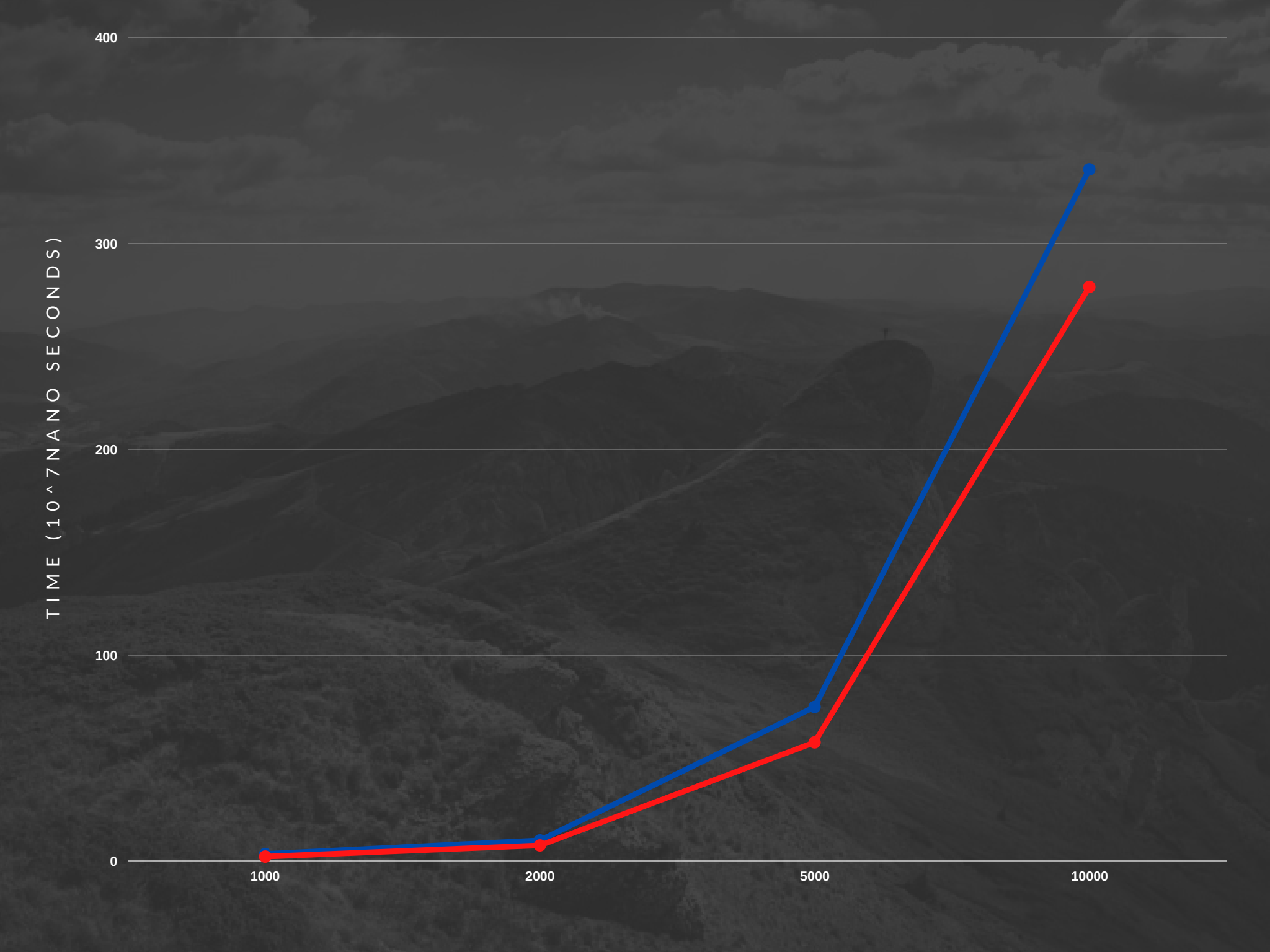
	1000	2000	5000	10000
BFS	3.49062199E7	1.0091344E8	7.491668399E8	3.3622540497E9
DFS	2.21273397E7	7.64059202E7	5.772049199E8	2.79068017E9

(Unit is nano seconds)

FOR NEXT GRAPH:

RED REPRESENTS: DFS

BLUE REPRESENTS: BFS


400
300
200
100
0
TIME (10^7NANO SECONDS)
1000
2000
5000
10000

Edge

- weight : int
- distance : int
- time : int
- quality : int

+ Edge(int source, int dest, int distance, int time, int quality)
+ Edge(int source, int dest)
+ Edge(int source, int dest, double w)
+ getSource() : int
+ getDest() : int
+ getWeight() : int
+ getDistance() : int
+ getTime() : int
+ getQuality() : int
+ toString() : String
+ equals(Object obj) : boolean
+ hashCode : int

Graph (Interface)

+ getNumV() : int
+ isDirected(): boolean
+ insert(Edge edge) : void
+ isEdge(int source, int dest) : boolean
+ getEdge(int source, int dest) : Edge
+ edgeIterator(int source) : Iterator < Edge >

DepthFirstSearch

- graph : Graph
- parent : int []
- visited : boolean[]
- discoveryOrder : int[]
- finishOrder : int[]
- discoverIndex : int
- finishIndex : int

+ DepthFirstSearch(Graph graph, int num)
+ depthFirstSearch(int current) : void
+ getFinishOrder() : int []

BreadthFirstSearch

+ breadthFirstSearch(Graph graph, int start) : int[]

AbstractGraph

- total : ArrayList<ArrayList<Integer>>
- numV : int
- directed : boolean

+ AbstractGraph(int numV, boolean directed)
+ nofccBreathFirstSearch() : int
+ nofccDepthFirstSearch() : int
+ importance() : ArrayList<Double>
- importanceUtil(Integer u, Integer d, boolean[] isVisited, ArrayList<Integer> localPathList) : void

DijkstrasAlgorithm

+ dijkstrasAlgorithm(Graph graph,int start,int[] pred,double[] dist, int selection1, int selection2) : void
+ func(double a, double b, int selection) : double

MatrixGraph

- edges : double[][]

+ MatrixGraph(int numV, boolean directed)
- setEdgeValue(int source, int dest, double wt) : void
- getEdgeValue(int source, int dest) : double

ListGraph

- edges : List < Edge > []

+ ListGraph(int numV, boolean directed)

Iter

- source : int
- index : int

+ Iter(int source)
+ hasNext() : boolean
+ next() : Edge
+ advanceIndex() : void

SCREENSHOTS:

~/java/cse222/hw8

```
emr3s@DESKTOP-POGAK7B ~/java/cse222/hw8
$ java driver
TESTING PART 1
Using Distance as Edge Property and (a * a) + (b * b) as operation With ListGraph:
PRED ARRAY:
0
0
1
0
0
2
3
4

DIST ARRAY:

0.0
2.0
5.0
4.0
4.0
146.0
20.0
116.0
Using Time as Edge Property and a / (a + b) as operation With ListGraph:
PRED ARRAY:
0
0
1
0
1
2
4
5

DIST ARRAY:

0.0
2.0
0.6666666666666666
4.0
0.6666666666666666
0.05714285714285714
0.14285714285714285
0.011299435028248588
Using Quality as Edge Property and (a + b + (a * b)) as operation With ListGraph:
PRED ARRAY:
0
0
1
0
0
2
3
4

DIST ARRAY:

0.0
2.0
5.0
4.0
4.0
71.0
14.0
54.0
Using Distance as Edge Property and (a * a) + (b * b) as operation With MatrixGraph:
PRED ARRAY:
0
0
1
```

~/java/cse222/hw8

```
0.0
2.0
5.0
4.0
4.0
71.0
14.0
54.0
Using Distance as Edge Property and (a * a) + (b * b) as operation With MatrixGraph:
PRED ARRAY:
0
0
1
0
0
2
3
4

DIST ARRAY:

0.0
2.0
5.0
4.0
4.0
146.0
20.0
116.0
Using Time as Edge Property and a / (a + b) as operation With MatrixGraph:
PRED ARRAY:
0
0
1
0
1
2
4
5

DIST ARRAY:

0.0
2.0
0.6666666666666666
4.0
0.6666666666666666
0.05714285714285714
0.25
0.011299435028248588
Using Quality as Edge Property and (a + b + (a * b)) as operation With MatrixGraph:
PRED ARRAY:
0
0
1
0
0
2
3
4

DIST ARRAY:

0.0
2.0
5.0
4.0
4.0
71.0
14.0
54.0
```

```
0.0
2.0
5.0
4.0
4.0
146.0
20.0
116.0
Using Time as Edge Property and a / (a + b) as operation With MatrixGraph:
PRED ARRAY:
0
0
1
0
1
2
4
5

DIST ARRAY:

0.0
2.0
0.6666666666666666
4.0
0.6666666666666666
0.05714285714285714
0.25
0.011299435028248588
Using Quality as Edge Property and (a + b + (a * b)) as operation With MatrixGraph:
PRED ARRAY:
0
0
1
0
0
2
3
4

DIST ARRAY:

0.0
2.0
5.0
4.0
4.0
71.0
14.0
54.0

TESTING PART 3
Using Graph With Size Equals to 10 to Calcute Importance of Each Vertex in the Graph
Vertex 0) 0.03
Vertex 1) 0.0
Vertex 2) 0.06
Vertex 3) 0.18
Vertex 4) 0.49
Vertex 5) 0.0
Vertex 6) 0.36
Vertex 7) 0.28
Vertex 8) 0.0
Vertex 9) 0.0

•       Using Graph With Size Equals to 5 to Calcute Importance of Each Vertex in the Graph

Vertex 0) 0.0
Vertex 1) 0.24
Vertex 2) 0.08
Vertex 3) 0.08
Vertex 4) 0.0
```

```
Using Time as Edge Property and a / (a + b) as operation With MatrixGraph:
PRED ARRAY:
0
0
1
0
1
2
4
5

DIST ARRAY:

0.0
2.0
0.6666666666666666
4.0
0.6666666666666666
0.05714285714285714
0.25
0.011299435028248588
Using Quality as Edge Property and (a + b + (a * b)) as operation With MatrixGraph:
PRED ARRAY:
0
0
1
0
0
2
3
4

DIST ARRAY:

0.0
2.0
5.0
4.0
4.0
71.0
14.0
54.0

TESTING PART 3
Using Graph With Size Equals to 10 to Calcute Importance of Each Vertex in the Graph
Vertex 0) 0.03
Vertex 1) 0.0
Vertex 2) 0.06
Vertex 3) 0.18
Vertex 4) 0.49
Vertex 5) 0.0
Vertex 6) 0.36
Vertex 7) 0.28
Vertex 8) 0.0
Vertex 9) 0.0

•       Using Graph With Size Equals to 5 to Calcute Importance of Each Vertex in the Graph

Vertex 0) 0.0
Vertex 1) 0.24
Vertex 2) 0.08
Vertex 3) 0.08
Vertex 4) 0.0

TESTING PART 2
AVG BREADFIRST RESULT FOR SIZE 1000: 3.49062199E7
AVG DEPTHFIRST RESULT FOR SIZE 1000: 2.21273397E7
AVG BREADFIRST RESULT FOR SIZE 2000: 1.0091344E8
AVG DEPTHFIRST RESULT FOR SIZE 2000: 7.64059202E7
AVG BREADFIRST RESULT FOR SIZE 5000: 7.491668399E8
AVG DEPTHFIRST RESULT FOR SIZE 5000: 5.772049199E8
```

```
1
0
1
2
4
5

DIST ARRAY:

0.0
2.0
0.6666666666666666
4.0
0.6666666666666666
0.05714285714285714
0.25
0.011299435028248588
Using Quality as Edge Property and (a + b + (a * b)) as operation With MatrixGraph:
PRED ARRAY:
0
0
1
0
0
2
3
4

DIST ARRAY:

0.0
2.0
5.0
4.0
4.0
71.0
14.0
54.0

TESTING PART 3
Using Graph With Size Equals to 10 to Calcute Importance of Each Vertex in the Graph
Vertex 0) 0.03
Vertex 1) 0.0
Vertex 2) 0.06
Vertex 3) 0.18
Vertex 4) 0.49
Vertex 5) 0.0
Vertex 6) 0.36
Vertex 7) 0.28
Vertex 8) 0.0
Vertex 9) 0.0

•       Using Graph With Size Equals to 5 to Calcute Importance of Each Vertex in the Graph

Vertex 0) 0.0
Vertex 1) 0.24
Vertex 2) 0.08
Vertex 3) 0.08
Vertex 4) 0.0

TESTING PART 2
AVG BREADFIRST RESULT FOR SIZE 1000: 3.49062199E7
AVG DEPTHFIRST RESULT FOR SIZE 1000: 2.21273397E7
AVG BREADFIRST RESULT FOR SIZE 2000: 1.0091344E8
AVG DEPTHFIRST RESULT FOR SIZE 2000: 7.64059202E7
AVG BREADFIRST RESULT FOR SIZE 5000: 7.491668399E8
AVG DEPTHFIRST RESULT FOR SIZE 5000: 5.772049199E8
AVG BREADFIRST RESULT FOR SIZE 10000: 3.3622540497E9
AVG DEPTHFIRST RESULT FOR SIZE 10000: 2.79068017E9

emr3s@DESKTOP-POGAK7B ~/java/cse222/hw8
$
```