HW4

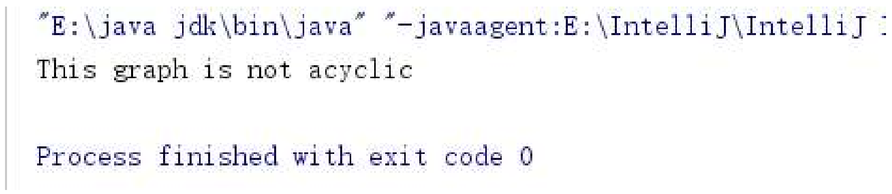
Topics: Graph

Course: DSA

Xinyu Lyu(xl422)

1. Write a program that answers the following for an undirected graph: Is a graph acyclic? Run your program on graph (linked after Q2)

   Result:

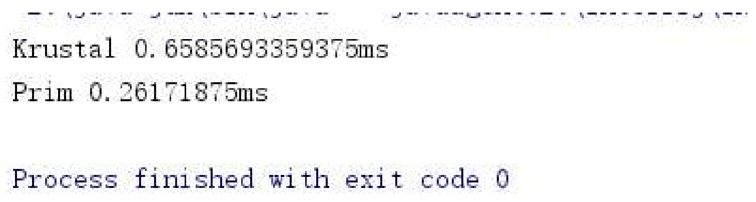


```
"E:\java jdk\bin\java" "-javaagent:E:\IntelliJ\IntelliJ
This graph is not acyclic

Process finished with exit code 0
```

2. Implement and execute Prim's and Kruskal's algorithms on the graph linked below (the third field is the weight of an edge). Which performs better? Explain your answer.

   Result:



```
Krustal 0.6585693359375ms
Prim 0.26171875ms

Process finished with exit code 0
```

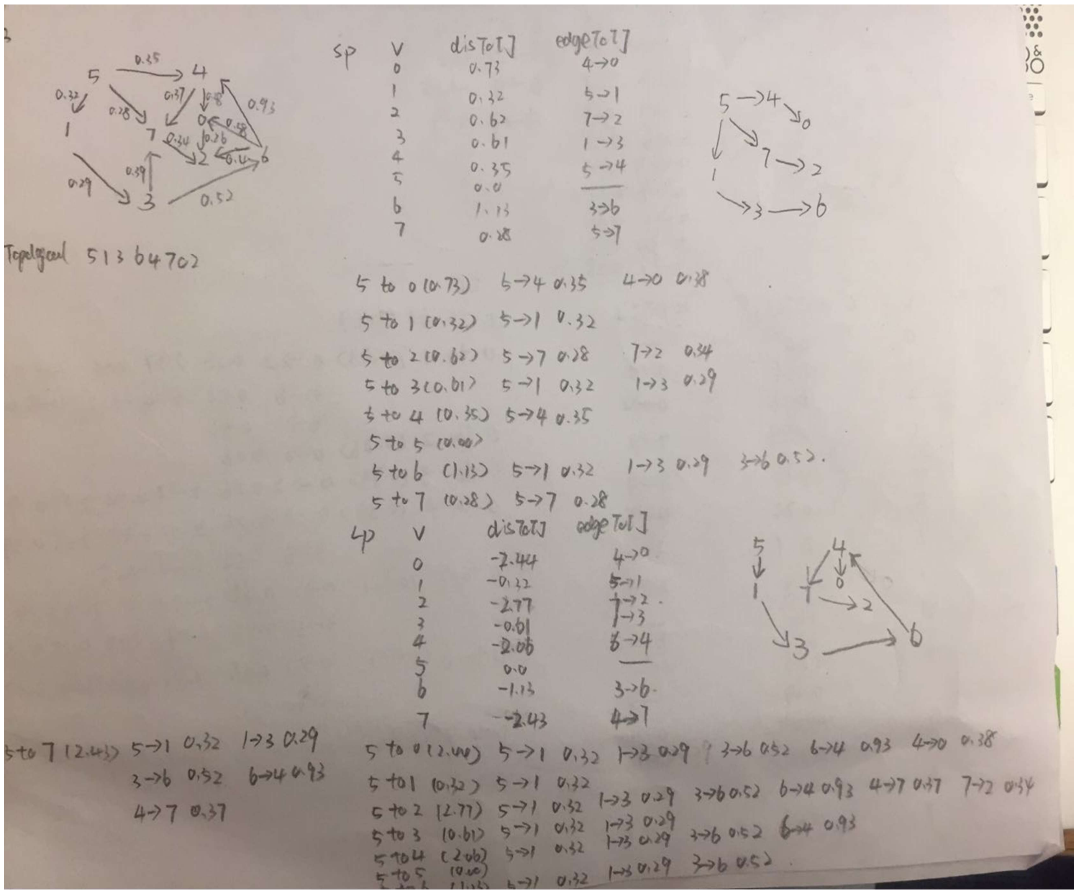
Prim is better, for the time complexity is ElogV. Krustal is ElogE. Here E is 1273 and V is 250. Therefore, ElogV is smaller than ElogE. So, Prim is better.

Use Graph linked here for Q1 and 2 [If the previous link does not work, equivalently download from: https://sakai.rutgers.edu/access/content/group/f73f2fd4-280d-4e7c-8cf2-9cc34bcffcff/HW-DataSet/mediumEWG.txt

3. For the edge-weighted directed acyclic graph given below, compute (i.e., manually trace) both the longest path and the shortest path.

8
13
5 4 0.35
4 7 0.37
5 7 0.28
5 1 0.32
4 0 0.38
0 2 0.26
3 7 0.39
1 3 0.29
7 2 0.34
6 2 0.40
3 6 0.52
6 0 0.58
6 4 0.93

Manually trace result:


0.35
0.32
0.28
0.37
0.38
0.93
0.58
0.34
0.26
0.40
0.39
0.29
0.52

Topological 5 1 3 6 4 7 0 2

SP	V	disTo[]	edgeTo[]
	0	0.73	4→0
	1	0.32	5→1
	2	0.62	7→2
	3	0.61	1→3
	4	0.35	5→4
	5	0.0	
	6	1.13	3→6
	7	0.28	5→7

5 to 0 (0.73) 5→4 0.35 4→0 0.38
5 to 1 (0.32) 5→1 0.32
5 to 2 (0.62) 5→7 0.28 7→2 0.34
5 to 3 (0.61) 5→1 0.32 1→3 0.29
5 to 4 (0.35) 5→4 0.35
5 to 5 (0.00)
5 to 6 (1.13) 5→1 0.32 1→3 0.29 3→6 0.52.
5 to 7 (0.28) 5→7 0.28

LP	V	disTo[]	edgeTo[]
	0	-2.44	4→0
	1	-0.32	5→1
	2	-2.77	7→2
	3	-0.61	1→3
	4	-2.06	6→4
	5	0.0	
	6	-1.13	3→6
	7	-2.43	4→7

5 to 7 (2.43) 5→1 0.32 1→3 0.29
3→6 0.52 6→4 0.93
4→7 0.37

5 to 0 (2.44) 5→1 0.32 1→3 0.29 3→6 0.52 6→4 0.93 4→0 0.38
5 to 1 (0.32) 5→1 0.32
5 to 2 (2.77) 5→1 0.32 1→3 0.29 3→6 0.52 6→4 0.93 4→7 0.37 7→2 0.34
5 to 3 (0.61) 5→1 0.32 1→3 0.29
5 to 4 (2.06) 5→1 0.32 1→3 0.29 3→6 0.52 6→4 0.93
5 to 5 (0.00)
[illegible] (1.13) 5→1 0.32 1→3 0.29 3→6 0.52

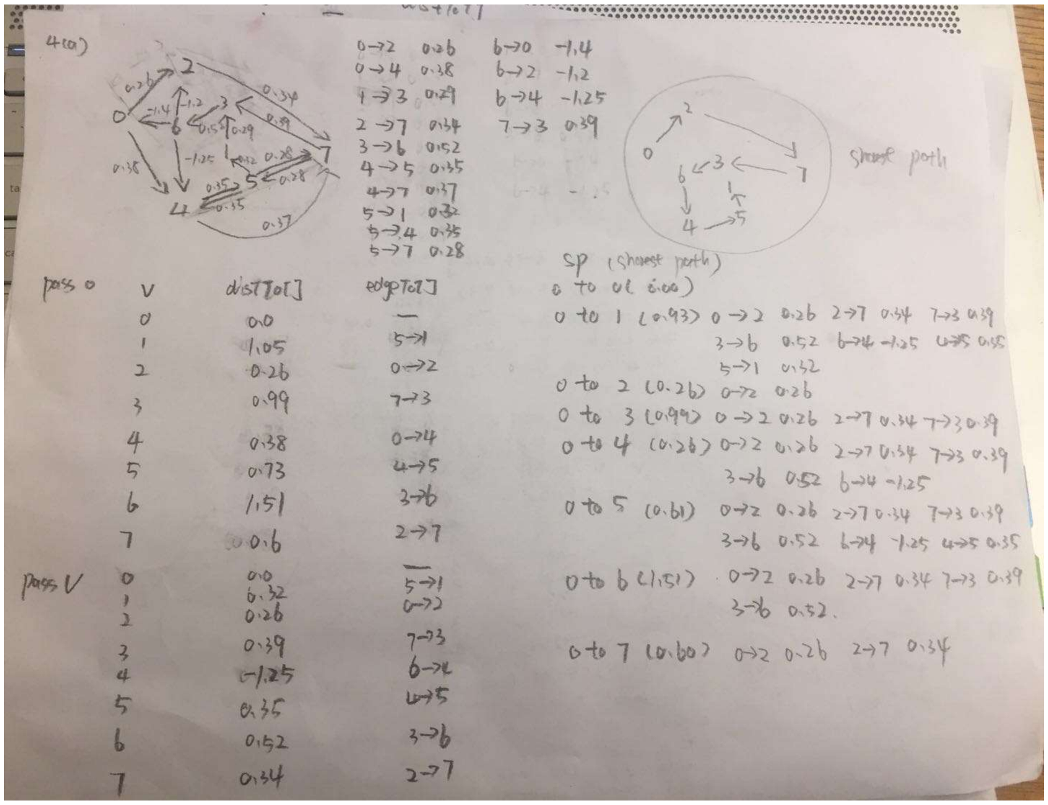
Code trace result:

```
SP
5 to 0 (0.73)  5->4  0.35   4->0  0.38
5 to 1 (0.32)  5->1  0.32
5 to 2 (0.62)  5->7  0.28   7->2  0.34
5 to 3 (0.61)  5->1  0.32   1->3  0.29
5 to 4 (0.35)  5->4  0.35
5 to 5 (0.00)
5 to 6 (1.13)  5->1  0.32   1->3  0.29   3->6  0.52
5 to 7 (0.28)  5->7  0.28
LP
5 to 0 (2.44)  5->1  0.32   1->3  0.29   3->6  0.52   6->4  0.93   4->0  0.38
5 to 1 (0.32)  5->1  0.32
5 to 2 (2.77)  5->1  0.32   1->3  0.29   3->6  0.52   6->4  0.93   4->7  0.37   7->2  0.34
5 to 3 (0.61)  5->1  0.32   1->3  0.29
5 to 4 (2.06)  5->1  0.32   1->3  0.29   3->6  0.52   6->4  0.93
5 to 5 (0.00)
5 to 6 (1.13)  5->1  0.32   1->3  0.29   3->6  0.52
5 to 7 (2.43)  5->1  0.32   1->3  0.29   3->6  0.52   6->4  0.93   4->7  0.37
```

4. (a) For the digraph with negative weights, compute (i.e. manually

trace) the progress of the Bellman-Ford Algorithm.

8
15
4 5   0.35
5 4   0.35
4 7   0.37
5 7   0.28
7 5   0.28
5 1   0.32
0 4   0.38
0 2   0.26
7 3   0.39
1 3   0.29
2 7   0.34
6 2 -1.20
3 6   0.52
6 0 -1.40
6 4 -1.25
Manually trace result:



4(a)

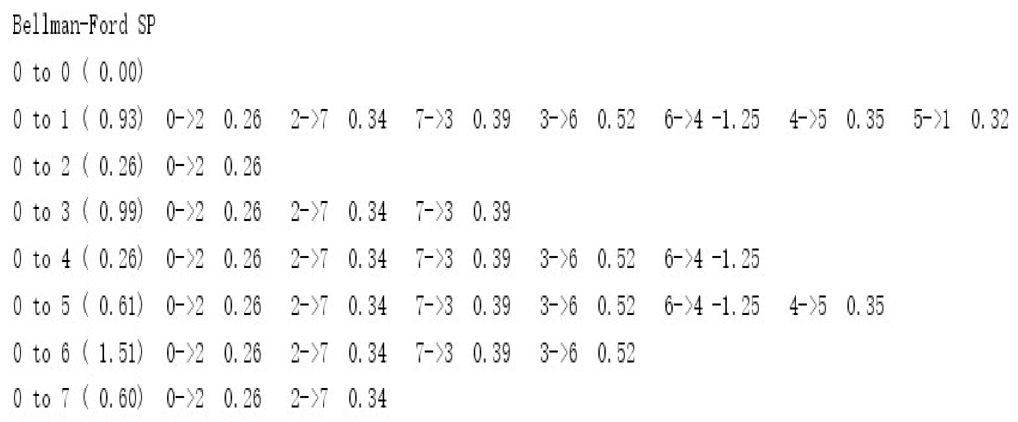
```
0→2  0.26     6→0  -1.4
0→4  0.38     6→2  -1.2
1→3  0.29     6→4  -1.25
2→7  0.34     7→3  0.39
3→6  0.52
4→5  0.35
4→7  0.37
5→1  0.32
5→4  0.35
5→7  0.28
```

shortest path

	v	distTo[]	edgeTo[]
pass 0	0	0.0	—
	1	1.05	5→1
	2	0.26	0→2
	3	0.99	7→3
	4	0.38	0→4
	5	0.73	4→5
	6	1.51	3→6
	7	0.6	2→7
pass V	0	0.0	—
	1	0.32	5→1
	2	0.26	0→2
	3	0.39	7→3
	4	-1.25	6→4
	5	0.35	4→5
	6	0.52	3→6
	7	0.34	2→7

```
sp (shortest path)
0 to 0 (0.00)
0 to 1 (0.93) 0→2 0.26 2→7 0.34 7→3 0.39
              3→6 0.52 6→4 -1.25 4→5 0.35
              5→1 0.32
0 to 2 (0.26) 0→2 0.26
0 to 3 (0.99) 0→2 0.26 2→7 0.34 7→3 0.39
0 to 4 (0.26) 0→2 0.26 2→7 0.34 7→3 0.39
              3→6 0.52 6→4 -1.25
0 to 5 (0.61) 0→2 0.26 2→7 0.34 7→3 0.39
              3→6 0.52 6→4 -1.25 4→5 0.35
0 to 6 (1.51) 0→2 0.26 2→7 0.34 7→3 0.39
              3→6 0.52.
0 to 7 (0.60) 0→2 0.26 2→7 0.34
```

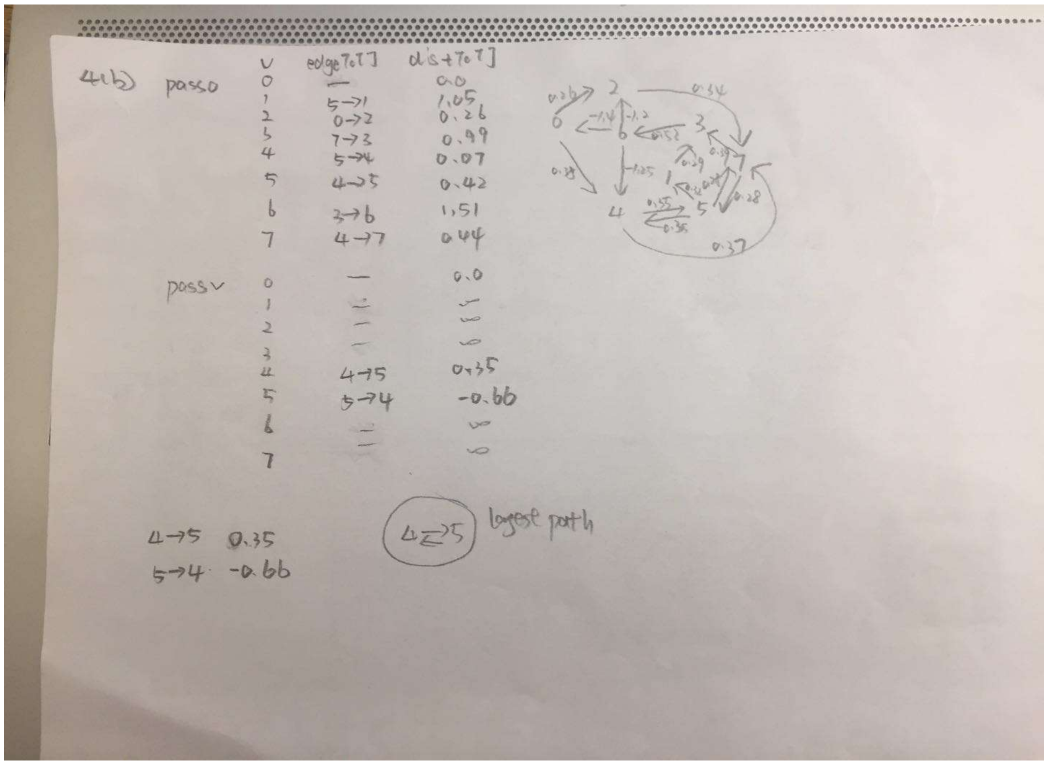
Code trace result;



```
Bellman-Ford SP
0 to 0 ( 0.00)
0 to 1 ( 0.93)  0->2  0.26   2->7  0.34   7->3  0.39   3->6  0.52   6->4 -1.25   4->5  0.35   5->1  0.32
0 to 2 ( 0.26)  0->2  0.26
0 to 3 ( 0.99)  0->2  0.26   2->7  0.34   7->3  0.39
0 to 4 ( 0.26)  0->2  0.26   2->7  0.34   7->3  0.39   3->6  0.52   6->4 -1.25
0 to 5 ( 0.61)  0->2  0.26   2->7  0.34   7->3  0.39   3->6  0.52   6->4 -1.25   4->5  0.35
0 to 6 ( 1.51)  0->2  0.26   2->7  0.34   7->3  0.39   3->6  0.52
0 to 7 ( 0.60)  0->2  0.26   2->7  0.34
```

4. (b) For the digraph with a negative cycle, compute (i.e. manually trace) the progress of the Bellman-Ford Algorithm.

```
8
15
4 5   0.35
5 4 -0.66
4 7   0.37
5 7   0.28
7 5   0.28
5 1   0.32
0 4   0.38
0 2   0.26
7 3   0.39
1 3   0.29
2 7   0.34
6 2   0.40
3 6   0.52
6 0   0.58
6 4   0.93
```


0
1
2
3
4
5
6
7
0.26
0.34
-1.4
-1.2
0.52
0.39
0.29
0.38
-1.25
0.32
0.34
0.28
0.35
0.35
0.37

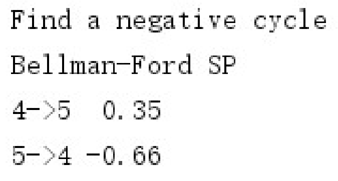
4(b)

	v	edgeTo[]	distTo[]
pass0	0	—	0.0
	1	5→1	1.05
	2	0→2	0.26
	3	7→3	0.99
	4	5→4	0.07
	5	4→5	0.42
	6	3→6	1.51
	7	4→7	0.44
passV	0	—	0.0
	1	—	∞
	2	—	∞
	3	—	∞
	4	4→5	0.35
	5	5→4	-0.66
	6	—	∞
	7	—	∞

4→5 0.35
5→4 -0.66

$4 \rightleftarrows 5$ logest path

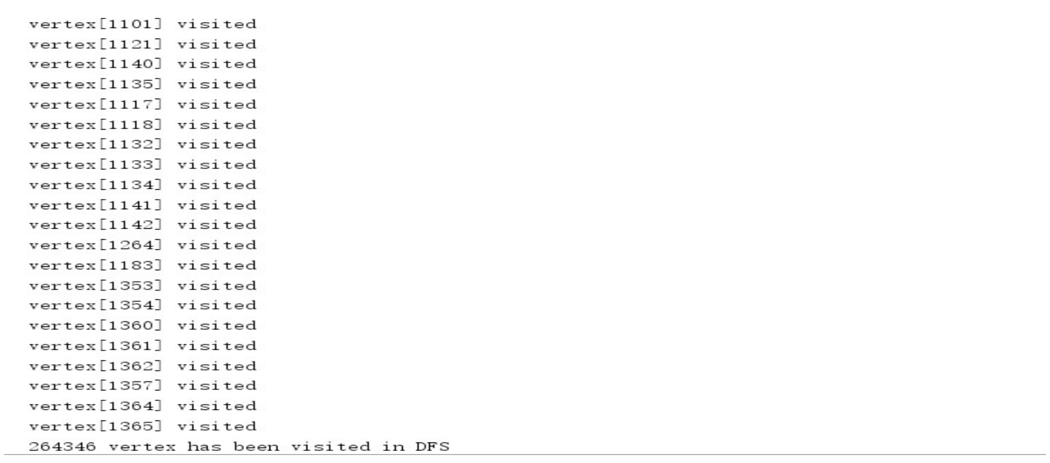
The graph has a negative cycle, therefore, BellmanFord can detect it.
Code trace result:



```
Find a negative cycle
Bellman-Ford SP
4->5  0.35
5->4 -0.66
```

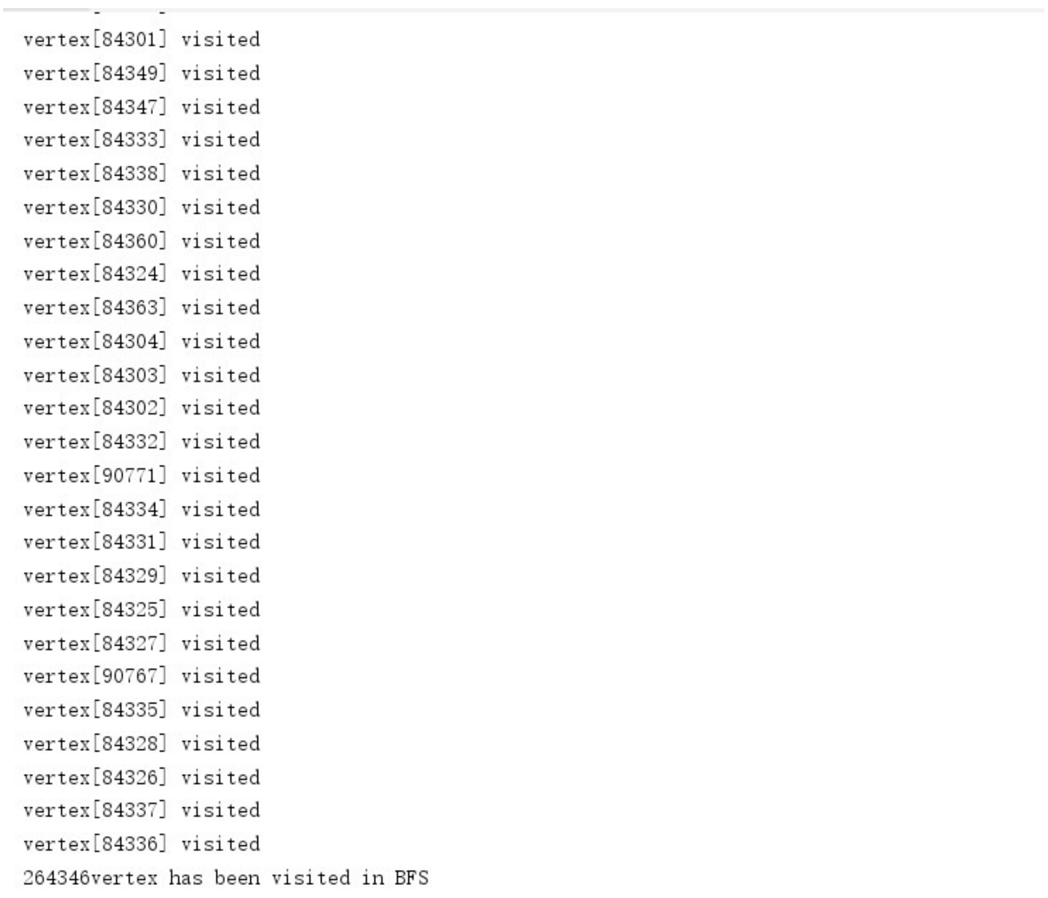
5. Implement a DFS and BFS traversal for the data-set of the undirected road network of New York City. The graph contains 264346 vertices and 733846 edges. It is connected, contains parallel edges, but no self-loops. The edge weights are travel times and are strictly positive.

The result for DFS:



```
vertex[1101] visited
vertex[1121] visited
vertex[1140] visited
vertex[1135] visited
vertex[1117] visited
vertex[1118] visited
vertex[1132] visited
vertex[1133] visited
vertex[1134] visited
vertex[1141] visited
vertex[1142] visited
vertex[1264] visited
vertex[1183] visited
vertex[1353] visited
vertex[1354] visited
vertex[1360] visited
vertex[1361] visited
vertex[1362] visited
vertex[1357] visited
vertex[1364] visited
vertex[1365] visited
264346 vertex has been visited in DFS
```

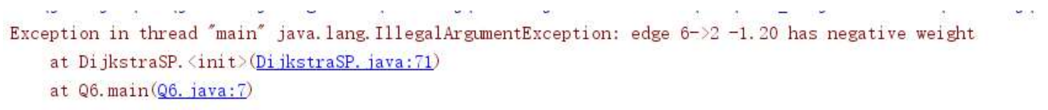
The result for BFS:



```
vertex[84301] visited
vertex[84349] visited
vertex[84347] visited
vertex[84333] visited
vertex[84338] visited
vertex[84330] visited
vertex[84360] visited
vertex[84324] visited
vertex[84363] visited
vertex[84304] visited
vertex[84303] visited
vertex[84302] visited
vertex[84332] visited
vertex[90771] visited
vertex[84334] visited
vertex[84331] visited
vertex[84329] visited
vertex[84325] visited
vertex[84327] visited
vertex[90767] visited
vertex[84335] visited
vertex[84328] visited
vertex[84326] visited
vertex[84337] visited
vertex[84336] visited
264346vertex has been visited in BFS
```

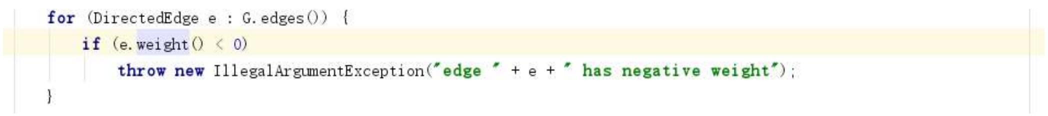
6. Implement the shortest path using Djikstra's Algorithm for the graph in HW5 Q 4(b). Then run your implementation of Djikstra's on HW5 4(a). What happens? Explain.

When running 4(a), there is an exception about the negative weight, for Djistra's Algorithm cannot deal with the negative weight.

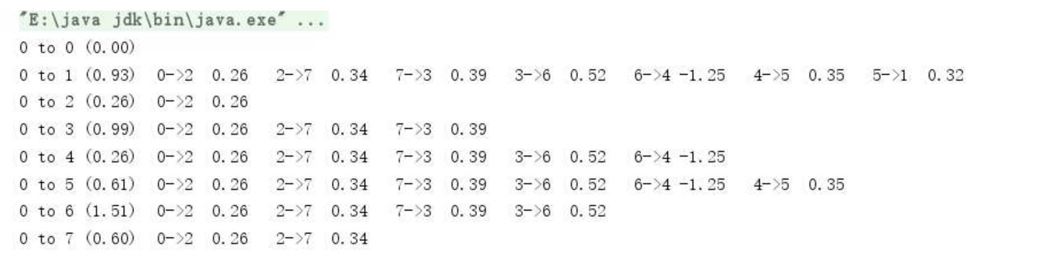


```
Exception in thread "main" java.lang.IllegalArgumentException: edge 6->2 -1.20 has negative weight
    at DijkstraSP.<init>(DijkstraSP.java:71)
    at Q6.main(Q6.java:7)
```

But if you comment the following codes in DijkstraSP.java, you could get the result, but it is not the correct result. The result is the same as the BellmanFord, but it is just a coincidence.

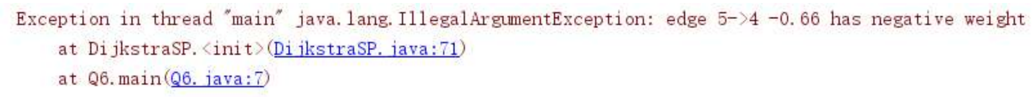


```
    for (DirectedEdge e : G.edges()) {
        if (e.weight() < 0)
            throw new IllegalArgumentException("edge " + e + " has negative weight");
    }
```



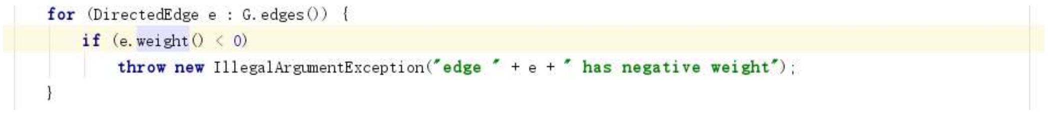
```
"E:\java jdk\bin\java.exe" ...
0 to 0 (0.00)
0 to 1 (0.93)  0->2  0.26   2->7  0.34   7->3  0.39   3->6  0.52   6->4 -1.25   4->5  0.35   5->1  0.32
0 to 2 (0.26)  0->2  0.26
0 to 3 (0.99)  0->2  0.26   2->7  0.34   7->3  0.39
0 to 4 (0.26)  0->2  0.26   2->7  0.34   7->3  0.39   3->6  0.52   6->4 -1.25
0 to 5 (0.61)  0->2  0.26   2->7  0.34   7->3  0.39   3->6  0.52   6->4 -1.25   4->5  0.35
0 to 6 (1.51)  0->2  0.26   2->7  0.34   7->3  0.39   3->6  0.52
0 to 7 (0.60)  0->2  0.26   2->7  0.34
```

When running 4(a). there is an exception about the negative weight, for it cannot deal with the negative weight cycle. We could find there is a negative cycle when running the Q4.



```
Exception in thread "main" java.lang.IllegalArgumentException: edge 5->4 -0.66 has negative weight
    at DijkstraSP.<init>(DijkstraSP.java:71)
    at Q6.main(Q6.java:7)
```

But if you comment the following codes in DijkstraSP.java, you couldn't get the result, for the negative cycle will make the codes in infinite loops.



```
    for (DirectedEdge e : G.edges()) {
        if (e.weight() < 0)
            throw new IllegalArgumentException("edge " + e + " has negative weight");
    }
```