Data Structure & Algorithm Homework 4

Zhuohang Li

Q1.

No. This diagram has cycle.

```
"C:\Program Files\Java\jdk1.8.0_152\bin\java" ...
This graph has cycle.

Process finished with exit code 0
```

Q2.

Kruskal's algorithm performs better. The reason is that Prim's algorithm is from vertices' perspective, while Kruskal's algorithm is from edges' perspective. So Kruskal's algorithm is better when dealing with graph that has large number of vertices and relatively small number of edges (sparse graph). Prim's algorithm has better performance when dealing graph that has large number of edges and small number of vertices (dense graph).

```
"C:\Program Files\Java\jdk1.8.0_152\bin\java" ...
Prim's Algorithm total weights: 10.463510000000001
Prim's Run Time:8(ms)
Kruskal's Algorithm total weights: 10.463510000000001
Kruskal's Run Time:4(ms)

Process finished with exit code 0
```

Q3.

Shortest path:

edgeTo[]	
0	
1	5->1
2	
3	
4	5->4
5	
6	
7	5->7

distTo[]	
0	
1	0.32
2	
3	
4	0.35
5	
6	
7	0.28

edgeTo[]	
0	
1	5->1
2	
3	1->3
4	5->4
5	
6	
7	5->7

distTo[]	
0	
1	0.32
2	
3	0.61
4	0.35
5	
6	
7	0.28

edgeTo[]	
0	
1	5->1
2	
3	1->3
4	5->4
5	
6	3->6
7	5->7

distTo[]	
0	
1	0.32
2	
3	0.61
4	0.35
5	
6	1.13
7	0.28

edgeTo[]	
0	6->0
1	5->1
2	6->2
3	1->3
4	5->4
5	
6	3->6
7	5->7

distTo[]	
0	1.71
1	0.32
2	1.53
3	0.61
4	0.35
5	
6	1.13
7	0.28

edgeTo[]	
0	4->0
1	5->1
2	6->2
3	1->3
4	5->4
5	
6	3->6
7	5->7

distTo[]	
0	0.73
1	0.32
2	1.53
3	0.61
4	0.35
5	
6	1.13
7	0.28

edgeTo[]	
0	4->0
1	5->1
2	7->2
3	1->3
4	5->4
5	
6	3->6
7	5->7

distTo[]	
0	0.73
1	0.32
2	0.62
3	0.61
4	0.35
5	
6	1.13
7	0.28

Longest Path: negate all edge weights and compute shortest path for the new graph

edgeTo[]	
0	
1	5->1
2	
3	
4	5->4
5	
6	
7	5->7

distTo[]	
0	
1	-0.32
2	
3	
4	-0.35
5	
6	
7	-0.28

edgeTo[]	
0	
1	5->1
2	
3	1->3
4	5->4
5	
6	
7	5->7

distTo[]	
0	
1	-0.32
2	
3	-0.61
4	-0.35
5	
6	
7	-0.28

edgeTo[]	
0	
1	5->1
2	
3	1->3
4	5->4
5	
6	3->6
7	5->7

distTo[]	
0	
1	-0.32
2	
3	-0.61
4	-0.35
5	
6	-1.13
7	-0.28

edgeTo[]	
0	6->0
1	5->1
2	6->2
3	1->3
4	5->4
5	
6	3->6
7	5->7

distTo[]	
0	-1.71
1	-0.32
2	-1.53
3	-0.61
4	-0.35
5	
6	-1.13
7	-0.28

Q4.

a:

edgeTo[]	
0	
1	
2	0->2
3	
4	0->4
5	4->5
6	
7	2->7

distTo[]	
0	
1	
2	0.26
3	
4	0.38
5	0.73
6	
7	0.6

edgeTo[]	
0	
1	5->1
2	0->2
3	7->3
4	0->4
5	4->5
6	
7	2->7

distTo[]	
0	
1	1.05
2	0.26
3	0.99
4	0.38
5	0.73
6	
7	0.6

edgeTo[]	
0	
1	5->1
2	0->2
3	7->3
4	0->4
5	4->5
6	3->6
7	2->7

distTo[]	
0	
1	1.05
2	0.26
3	0.99
4	0.38
5	0.73
6	1.51
7	0.6

edgeTo[]	
0	
1	5->1
2	0->2
3	7->3
4	6->4
5	4->5
6	3->6
7	2->7

distTo[]	
0	
1	1.05
2	0.26
3	0.99
4	0.26
5	0.73
6	1.51
7	0.6

edgeTo[]	
0	
1	5->1
2	0->2
3	7->3
4	6->4
5	4->5
6	3->6
7	2->7

distTo[]	
0	
1	1.05
2	0.26
3	0.99
4	0.26
5	0.61
6	1.51
7	0.6

edgeTo[]	
0	
1	5->1
2	0->2
3	7->3
4	6->4
5	4->5
6	3->6
7	2->7

distTo[]	
0	
1	0.93
2	0.26
3	0.99
4	0.26
5	0.61
6	1.51
7	0.6

b:

edgeTo[]	
0	
1	
2	0->2
3	
4	0->4
5	4->5
6	
7	2->7

distTo[]	
0	
1	
2	0.26
3	
4	0.38
5	0.73
6	
7	0.6

edgeTo[]	
0	
1	5->1
2	0->2
3	7->3
4	5->4
5	4->5
6	
7	2->7

distTo[]	
0	
1	1.05
2	0.26
3	0.99
4	0.07
5	0.73
6	
7	0.6

edgeTo[]	
0	
1	5->1
2	0->2
3	7->3
4	5->4
5	4->5
6	3->6
7	4->7

distTo[]	
0	
1	1.05
2	0.26
3	0.99
4	0.07
5	0.42
6	1.51
7	0.44

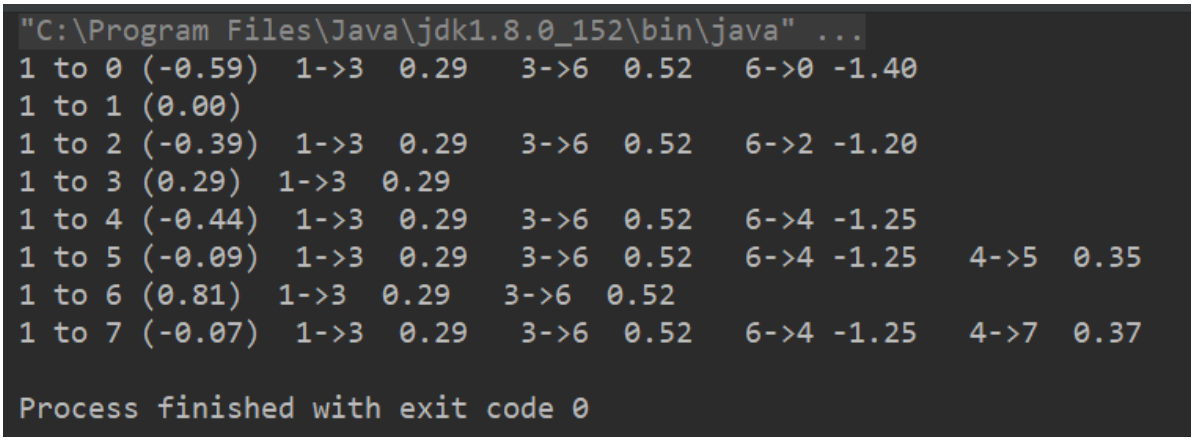
edgeTo[]	
0	
1	5->1
2	0->2
3	7->3
4	5->4
5	4->5
6	3->6
7	4->7

distTo[]	
0	
1	0.74
2	0.26
3	0.83
4	-0.24
5	0.42
6	1.51
7	0.44

And so on and so forth, keeps doing theses steps and each time values get smaller.

Q6.

For 4(a), Djikstra's algorithm gets a wrong answer.



For 4(b), the algorithm gets stuck in an endless loop, never finishes execution.

The reason is Djikstra's algorithm only works for graphs without negative weights. For graph with negative cycle, the relaxation operation will not converge.