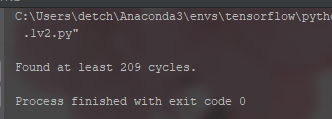
**DSA Homework 4**

**Ruiyu Zhang rz213**

**1.Write a program that answers the following for an undirected graph: Is a graph acyclic?  Run your program on graph**

A:

This graph is NOT acyclic. I found at least 209 cycles.

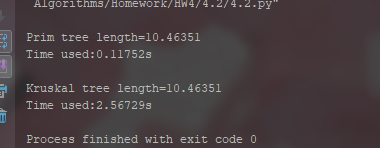
Code run under python 3.5 environment.

All cycles are printed in “cycles.txt” attached.

**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.**

A:

Both algorithms give tree of the same size:



Code run under python 3.5 environment.

Talking about performace, Prim is much better(faster) than Kruskal. The reason is, in Kruskal’s implementation, each time a link is about to be added, whether this link will make a cycle in tree will be checked. This takes union find method to complete, which takes much time if graph is large.

**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**

A:

START=0

END=1: No Access

END=2: Long=0.26 Short=0.26

END=3: No Access

END=4: No Access

END=5: No Access

END=6: No Access

END=7: No Access

START=1

END=0: Long=2.12 Short=1.39

END=2: Long=2.45 Short=1.02

END=3: Long=0.29 Short=0.29

END=4: Long=1.74 Short=1.74

END=5: No Access

END=6: Long=0.81 Short=0.81

END=7: Long=2.11 Short=0.68

START=2

END=0: No Access

END=1: No Access

END=3: No Access

END=4: No Access

END=5: No Access

END=6: No Access

END=7: No Access

START=3

END=0: Long=1.83 Short=1.10

END=1: No Access

END=2: Long=2.16 Short=0.73

END=4: Long=1.45 Short=1.45

END=5: No Access

END=6: Long=0.52 Short=0.52

END=7: Long=1.82 Short=0.39

START=4

END=0: Long=0.38 Short=0.38

END=1: No Access

END=2: Long=0.71 Short=0.64

END=3: No Access

END=5: No Access

END=6: No Access

END=7: Long=0.37 Short=0.37

START=5

END=0: Long=2.44 Short=0.73

END=1: Long=0.32 Short=0.32

END=2: Long=2.77 Short=0.62

END=3: Long=0.61 Short=0.61

END=4: Long=2.06 Short=0.35

END=6: Long=1.13 Short=1.13

END=7: Long=2.43 Short=0.28

START=6

END=0: Long=1.31 Short=0.58

END=1: No Access

END=2: Long=1.64 Short=0.40

END=3: No Access

END=4: Long=0.93 Short=0.93

END=5: No Access

END=7: Long=1.30 Short=1.30

START=7

END=0: No Access

END=1: No Access

END=2: Long=0.34 Short=0.34

END=3: No Access

END=4: No Access

END=5: No Access

END=6: No Access

Code run under python 3.5 environment.

Full result also printed in “4.3-result.txt” attached.

**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**

A:

















Code run under python 3.5 environment.

Full result also printed in “4.4a-result.txt” and “4.4a-result.xlsx” attached.

**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**

A:















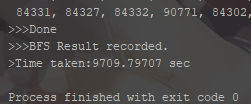


Code run under python 3.5 environment.

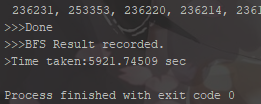
Full result also printed in “4.4b-result.txt” and “4.4b-result.xlsx” attached.

**5. Implement a DFS and BFS traversal for the data-set of the**[**undirected road network of New York City**](http://algs4.cs.princeton.edu/44sp/NYC.txt)**. 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.**

A:



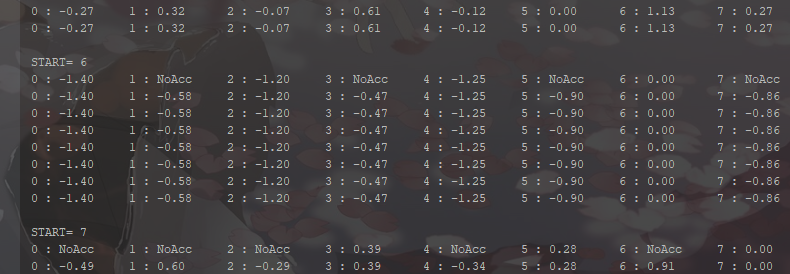
Result attached in txt

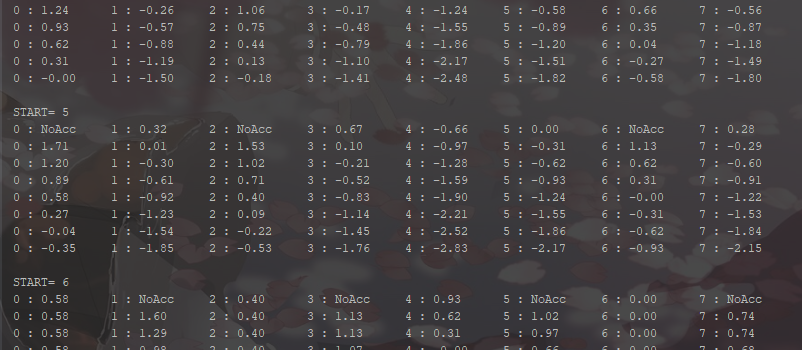


Above is the screenshot of both algorithms. Both results are printed in “4.5-BFS-Result.txt” and “4.5-DFS-Result.txt” attached.

**6. Implement the shortest path using Dijkstra’s Algorithm for the graph in HW4 Q4(b).  Then run your implementation of Dijkstra’s on HW5 4(a). What happens? Explain.**

A:

a)

b)

Part of results shown above in screenshots, Start=6/5. We can see that some results are incorrect, the reason is Dijkstra’s algorithm cannot handle a graph with negative weights. Full results printed in “4.6-result-run4.4a.txt” and “4.6-result-run4.4b.txt”.