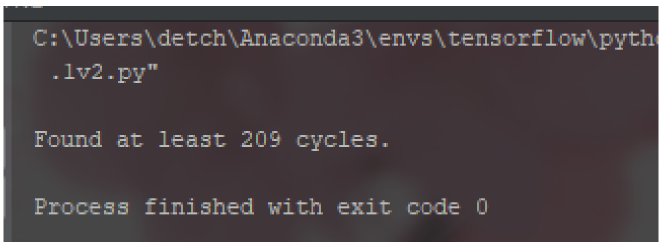
DSA Homework 4

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.



```
C:\Users\detch\Anaconda3\envs\tensorflow\pyth
 .1v2.py"

Found at least 209 cycles.

Process finished with exit code 0
```

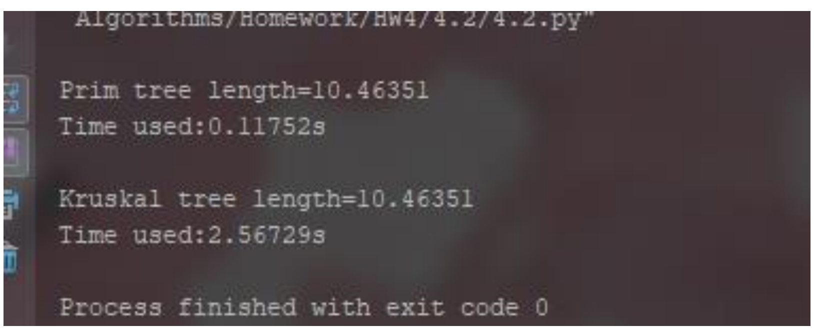
Code run under python 3.5 environment.

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:



```
 Algorithms/Homework/HW4/4.2/4.2.py"

Prim tree length=10.46351
Time used:0.11752s

Kruskal tree length=10.46351
Time used:2.56729s

Process finished with exit code 0
```

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:

START= 0						
END=1	END=2	END=3	END=4	END=5	END=6	END=7
NoAccess	0.26	NoAccess	0.38	NoAccess	NoAccess	NoAccess
1.05	0.26	0.99	0.38	0.73	1.51	0.6
0.93	0.26	0.99	0.26	0.61	1.51	0.6
0.93	0.26	0.99	0.26	0.61	1.51	0.6
START= 1						
END=0	END=2	END=3	END=4	END=5	END=6	END=7
NoAccess	-0.39	0.29	NoAccess	NoAccess	0.81	NoAccess
-0.59	-0.39	0.29	-0.44	-0.09	0.81	-0.05
-0.59	-0.39	0.29	-0.44	-0.09	0.81	-0.05
START= 2						
END=0	END=1	END=3	END=4	END=5	END=6	END=7
NoAccess	0.94	0.73	NoAccess	0.62	1.25	0.34
-0.15	0.67	0.73	0	0.35	1.25	0.34
-0.15	0.67	0.73	0	0.35	1.25	0.34
START= 3						
END=0	END=1	END=2	END=4	END=5	END=6	END=7
NoAccess	NoAccess	-0.68	NoAccess	NoAccess	0.52	NoAccess
-0.88	-0.06	-0.68	-0.73	-0.38	0.52	-0.34
-0.88	-0.06	-0.68	-0.73	-0.38	0.52	-0.34

START= 4						
END=0	END=1	END=2	END=3	END=5	END=6	END=7
NoAccess	0.67	NoAccess	NoAccess	0.35	NoAccess	0.63
NoAccess	0.67	0.28	0.96	0.35	1.48	0.63
0.08	0.67	0.28	0.96	0.35	1.48	0.62
0.08	0.67	0.28	0.96	0.35	1.48	0.62
START= 5						
END=0	END=1	END=2	END=3	END=4	END=6	END=7
NoAccess	0.32	NoAccess	NoAccess	0.35	NoAccess	0.28
NoAccess	0.32	-0.07	0.61	0.35	1.13	0.28
-0.27	0.32	-0.07	0.61	-0.12	1.13	0.27
-0.27	0.32	-0.07	0.61	-0.12	1.13	0.27
START= 6						
END=0	END=1	END=2	END=3	END=4	END=5	END=7
-1.4	-0.58	-1.2	NoAccess	-1.25	-0.9	-0.62
-1.4	-0.58	-1.2	-0.47	-1.25	0.9	0.86
-1.4	-0.58	-1.2	-0.47	-1.25	0.9	0.86
START= 7						
END=0	END=1	END=2	END=3	END=4	END=5	END=6
NoAccess	0.6	-0.29	0.39	NoAccess	0.28	0.91
-0.49	0.33	-0.29	0.39	-0.34	0.01	0.91
-0.49	0.33	-0.29	0.39	-0.34	0.01	0.91

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:

START= 0						
END=1	END=2	END=3	END=4	END=5	END=6	END=7
NoAccess	0.26	NoAccess	0.38	NoAccess	NoAccess	NoAccess
1.05	0.26	0.99	0.38	0.73	1.51	0.6
0.74	0.26	0.99	0.07	0.42	1.51	0.44
0.43	0.26	0.83	-0.24	0.11	1.35	0.13
0.12	0.26	0.52	-0.55	-0.2	1.04	-0.18
-0.19	0.26	0.21	-0.86	-0.51	0.73	-0.49
-0.5	0.26	-0.1	-1.17	-0.82	0.42	-0.8
-0.5	0.26	-0.1	-1.17	-0.82	0.42	-0.8

START= 1						
END=0	END=2	END=3	END=4	END=5	END=6	END=7
NoAccess	1.21	0.29	NoAccess	NoAccess	0.81	NoAccess
1.39	1.21	0.29	1.74	1.83	0.81	1.55
1.39	1.21	0.29	1.17	1.52	0.81	1.54
1.39	1.21	0.29	0.86	1.21	0.81	1.23
1.39	1.21	0.29	0.55	0.9	0.81	0.92
1.39	1.21	0.29	0.24	0.59	0.81	0.61
1.39	1.21	0.29	-0.07	0.28	0.81	0.3
1.39	1.21	0.29	-0.07	0.28	0.81	0.3

START= 2						
END=0	END=1	END=3	END=4	END=5	END=6	END=7
NoAccess	0.94	0.73	NoAccess	0.62	1.25	0.34
1.83	0.63	0.73	-0.04	0.31	1.25	0.33
1.83	0.32	0.72	-0.35	0	1.24	0.02
1.82	0.01	0.41	-0.66	-0.31	0.93	-0.29
1.51	-0.3	0.1	-0.97	-0.62	0.62	-0.6
1.2	-0.61	-0.21	-1.28	-0.93	0.31	-0.91
0.89	-0.92	-0.52	-1.59	-1.24	0	-1.22
0.89	-0.92	-0.52	-1.59	-1.24	0	-1.22

START= 3						
END=0	END=1	END=2	END=4	END=5	END=6	END=7
NoAccess	NoAccess	0.92	NoAccess	NoAccess	0.52	NoAccess
1.1	1.86	0.92	1.45	1.54	0.52	1.26
1.1	1.55	0.92	0.88	1.23	0.52	1.25
1.1	1.24	0.92	0.57	0.92	0.52	0.94
1.1	0.93	0.92	0.26	0.61	0.52	0.63
1.1	0.62	0.92	-0.05	0.3	0.52	0.32
1.1	0.31	0.92	-0.36	-0.01	0.52	0.01
1.1	0.31	0.92	-0.36	-0.01	0.52	0.01

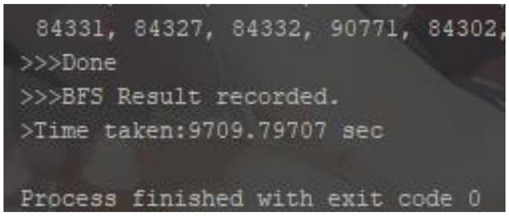
START= 4						
END=0	END=1	END=2	END=3	END=5	END=6	END=7
NoAccess	0.67	NoAccess	NoAccess	0.35	NoAccess	0.37
NoAccess	0.36	1.68	0.76	0.04	1.28	0.06
1.86	0.05	1.37	0.45	-0.27	0.97	-0.25
1.55	-0.26	1.06	0.14	-0.58	0.66	-0.56
1.24	-0.57	0.75	-0.17	-0.89	0.35	-0.87
0.93	-0.88	0.44	-0.48	-1.2	0.04	-1.18
0.62	-1.19	0.13	-0.79	-1.51	-0.27	-1.49
0.62	-1.19	0.13	-0.79	-1.51	-0.27	-1.49
START= 5						
END=0	END=1	END=2	END=3	END=4	END=6	END=7
NoAccess	0.01	NoAccess	NoAccess	-0.66	NoAccess	-0.29
NoAccess	-0.3	1.02	0.1	-0.97	0.62	-0.6
1.2	-0.61	0.71	-0.21	-1.28	0.31	-0.91
0.89	-0.92	0.4	-0.52	-1.59	0	-1.22
0.58	-1.23	0.09	-0.83	-1.9	-0.31	-1.53
0.27	-1.54	-0.22	-1.14	-2.21	-0.62	-1.84
-0.04	-1.85	-0.53	-1.45	-2.52	-0.93	-2.15
-0.04	-1.85	-0.53	-1.45	-2.52	-0.93	-2.15
START= 6						
END=0	END=1	END=2	END=3	END=4	END=5	END=7
0.58	1.6	0.4	NoAccess	0.93	1.28	1.3
0.58	1.29	0.4	1.13	0.62	0.97	0.74
0.58	0.98	0.4	1.13	0.31	0.66	0.68
0.58	0.67	0.4	1.07	0	0.35	0.37
0.58	0.36	0.4	0.76	-0.31	0.04	0.06
0.58	0.05	0.4	0.45	-0.62	-0.27	-0.25
0.58	-0.26	0.4	0.14	-0.93	-0.58	-0.56
0.58	-0.26	0.4	0.14	-0.93	-0.58	-0.56
START= 7						
END=0	END=1	END=2	END=3	END=4	END=5	END=6
NoAccess	0.6	1.31	0.39	NoAccess	0.28	0.91
1.49	0.29	1.31	0.39	-0.38	-0.03	0.91
1.49	-0.02	1.3	0.38	-0.69	-0.34	0.9
1.48	-0.33	0.99	0.07	-1	-0.65	0.59
1.17	-0.64	0.68	-0.24	-1.31	-0.96	0.28
0.86	-0.95	0.37	-0.55	-1.62	-1.27	-0.03
0.55	-1.26	0.06	-0.86	-1.93	-1.58	-0.34
0.55	-1.26	0.06	-0.86	-1.93	-1.58	-0.34

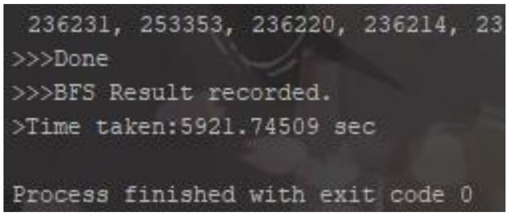
Code run under python 3.5 environment.

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. 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:





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)

```
0 : -0.27    1 : 0.32     2 : -0.07    3 : 0.61     4 : -0.12    5 : 0.00     6 : 1.13     7 : 0.27
0 : -0.27    1 : 0.32     2 : -0.07    3 : 0.61     4 : -0.12    5 : 0.00     6 : 1.13     7 : 0.27

START= 6
0 : -1.40    1 : NoAcc    2 : -1.20    3 : NoAcc    4 : -1.25    5 : NoAcc    6 : 0.00     7 : NoAcc
0 : -1.40    1 : -0.58    2 : -1.20    3 : -0.47    4 : -1.25    5 : -0.90    6 : 0.00     7 : -0.86
0 : -1.40    1 : -0.58    2 : -1.20    3 : -0.47    4 : -1.25    5 : -0.90    6 : 0.00     7 : -0.86
0 : -1.40    1 : -0.58    2 : -1.20    3 : -0.47    4 : -1.25    5 : -0.90    6 : 0.00     7 : -0.86
0 : -1.40    1 : -0.58    2 : -1.20    3 : -0.47    4 : -1.25    5 : -0.90    6 : 0.00     7 : -0.86
0 : -1.40    1 : -0.58    2 : -1.20    3 : -0.47    4 : -1.25    5 : -0.90    6 : 0.00     7 : -0.86
0 : -1.40    1 : -0.58    2 : -1.20    3 : -0.47    4 : -1.25    5 : -0.90    6 : 0.00     7 : -0.86
0 : -1.40    1 : -0.58    2 : -1.20    3 : -0.47    4 : -1.25    5 : -0.90    6 : 0.00     7 : -0.86

START= 7
0 : NoAcc    1 : NoAcc    2 : NoAcc    3 : 0.39     4 : NoAcc    5 : 0.28     6 : NoAcc    7 : 0.00
0 : -0.49    1 : 0.60     2 : -0.29    3 : 0.39     4 : -0.34    5 : 0.28     6 : 0.91     7 : 0.00
```

b)

```
0 : 1.24     1 : -0.26    2 : 1.06     3 : -0.17    4 : -1.24    5 : -0.58    6 : 0.66     7 : -0.56
0 : 0.93     1 : -0.57    2 : 0.75     3 : -0.48    4 : -1.55    5 : -0.89    6 : 0.35     7 : -0.87
0 : 0.62     1 : -0.88    2 : 0.44     3 : -0.79    4 : -1.86    5 : -1.20    6 : 0.04     7 : -1.18
0 : 0.31     1 : -1.19    2 : 0.13     3 : -1.10    4 : -2.17    5 : -1.51    6 : -0.27    7 : -1.49
0 : -0.00    1 : -1.50    2 : -0.18    3 : -1.41    4 : -2.48    5 : -1.82    6 : -0.58    7 : -1.80

START= 5
0 : NoAcc    1 : 0.32     2 : NoAcc    3 : 0.67     4 : -0.66    5 : 0.00     6 : NoAcc    7 : 0.28
0 : 1.71     1 : 0.01     2 : 1.53     3 : 0.10     4 : -0.97    5 : -0.31    6 : 1.13     7 : -0.29
0 : 1.20     1 : -0.30    2 : 1.02     3 : -0.21    4 : -1.28    5 : -0.62    6 : 0.62     7 : -0.60
0 : 0.89     1 : -0.61    2 : 0.71     3 : -0.52    4 : -1.59    5 : -0.93    6 : 0.31     7 : -0.91
0 : 0.58     1 : -0.92    2 : 0.40     3 : -0.83    4 : -1.90    5 : -1.24    6 : -0.00    7 : -1.22
0 : 0.27     1 : -1.23    2 : 0.09     3 : -1.14    4 : -2.21    5 : -1.55    6 : -0.31    7 : -1.53
0 : -0.04    1 : -1.54    2 : -0.22    3 : -1.45    4 : -2.52    5 : -1.86    6 : -0.62    7 : -1.84
0 : -0.35    1 : -1.85    2 : -0.53    3 : -1.76    4 : -2.83    5 : -2.17    6 : -0.93    7 : -2.15

START= 6
0 : 0.58     1 : NoAcc    2 : 0.40     3 : NoAcc    4 : 0.93     5 : NoAcc    6 : 0.00     7 : NoAcc
0 : 0.58     1 : 1.60     2 : 0.40     3 : 1.13     4 : 0.62     5 : 1.02     6 : 0.00     7 : 0.74
0 : 0.58     1 : 1.29     2 : 0.40     3 : 1.13     4 : 0.31     5 : 0.97     6 : 0.00     7 : 0.74
```

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".