DSA HW4

Yiming Bi – yb127 – 146004795

Q1

Python code is another file.
The graph is acyclic.

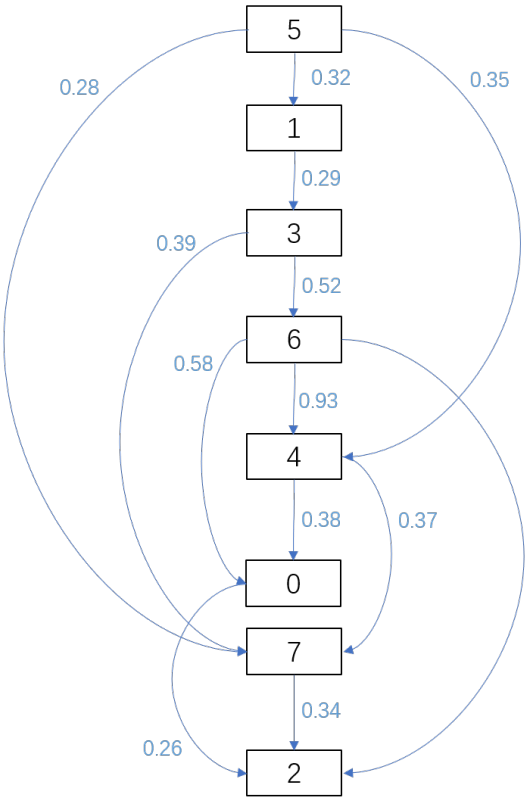
Q2

Python code is another file.
For both methods, the total weight is 1.04635.
Prim method costs 0.00598s and Kruskal method costs 0.000997s.
Kruskal method has better time performance.

Q3

Shortest path:

Below is topological sort of the graph:



Trace table:

trace	5	1	3	6	4	0	7	2
0	0	0.32	inf	inf	0.35	inf	0.28	inf
1	0	0.32	Inf	Inf	0.35	Inf	0.28	0.62
2	0	0.32	0.61	inf	0.35	inf	0.28	0.62
3	0	0.32	0.61	inf	0.35	0.73	0.28	0.62
4	0	0.32	0.61	1.13	0.35	0.73	0.28	0.62
5	0	0.32	0.61	1.13	0.35	0.73	0.28	0.62
6	0	0.32	0.61	1.13	0.35	0.73	0.28	0.62

Path:

0: 5 →4 →0

1: 5 →1

2: 5 →7 →2

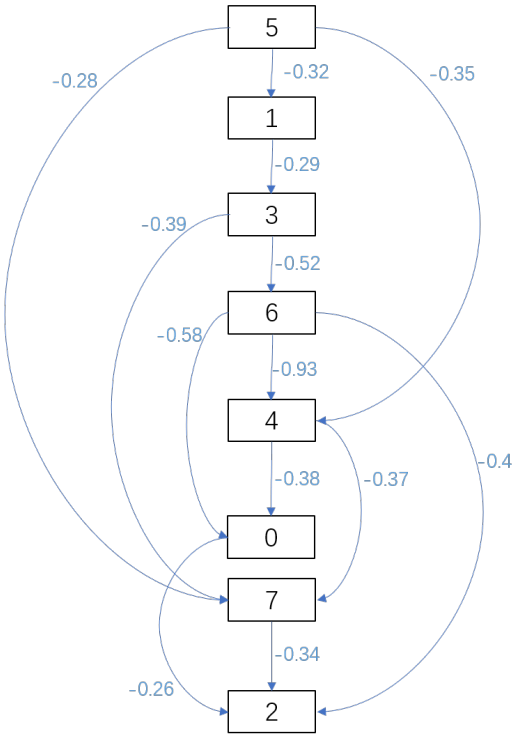
3: 5 →1 →3

4: 5 →4

6: 5 →1 →3 →6

7: 5 →7

Longest path:



Trace table:

trace	5	1	3	6	4	0	7	2
0	0	-0.32	-inf	-inf	-0.35	-inf	-0.28	-inf
1	0	-0.32	-0.61	-inf	-0.35	-inf	-0.28	-inf
2	0	-0.32	-0.61	-1.13	-0.35	-inf	-1	-inf
3	0	-0.32	-0.61	-1.13	-2.06	-1.71	-1	-1.53
4	0	-0.32	-0.61	-1.13	-2.06	-2.44	-2.43	-1.53
5	0	-0.32	-0.61	-1.13	-2.06	-2.44	-2.43	-2.7
6	0	-0.32	-0.61	-1.13	-2.06	-2.44	-2.43	-2.77
7	0	0.32	0.61	1.13	2.06	2.44	2.43	2.77

Path:

0: 5 →1 →3 →6 →4 →0

1: 5 →1

2: 5 →1 →3 →6 →4 →7 →2

3: 5 →1 →3

4: 5 →1 →3 →6 →4

6: 5 →1 →3 →6

7: 5 →1 →3 →6 →4 →7

Q4

a

Setup:

I choose vertex 0 as start vertex.

vertex	0	1	2	3	4	5	6	7
distance	0	∞	∞	∞	∞	∞	∞	∞

Iteration 1:

vertex	0	1	2	3	4	5	6	7
distance	0	∞	0.26	∞	0.38	∞	∞	0.6

Iteration 2:

vertex	0	1	2	3	4	5	6	7
distance	0	∞	0.26	∞	0.38	0.73	∞	0.6

Iteration 3:

vertex	0	1	2	3	4	5	6	7
distance	0	1.05	0.26	0.99	0.26	0.73	1.51	0.6

Iteration 4:

vertex	0	1	2	3	4	5	6	7
distance	0	0.93	0.26	0.99	0.26	0.61	1.51	0.6

Iteration 5:

vertex	0	1	2	3	4	5	6	7
distance	0	0.93	0.26	0.99	0.26	0.61	1.51	0.6

Iteration 6:

vertex	0	1	2	3	4	5	6	7
distance	0	0.93	0.26	0.99	0.26	0.61	1.51	0.6

Iteration 7:

vertex	0	1	2	3	4	5	6	7
distance	0	0.93	0.26	0.99	0.26	0.61	1.51	0.6

Iteration 7 is the result since this graph has no negative loop.

b

Setup:

I choose vertex 0 as start vertex.

vertex	0	1	2	3	4	5	6	7
distance	0	∞	∞	∞	∞	∞	∞	∞

Iteration 1:

vertex	0	1	2	3	4	5	6	7
distance	0	∞	0.26	∞	0.38	∞	∞	0.6

Iteration 2:

vertex	0	1	2	3	4	5	6	7
distance	0	1.04	0.26	0.83	0.07	0.72	1.35	0.44

Iteration 3:

vertex	0	1	2	3	4	5	6	7
distance	0	0.73	0.26	0.52	-0.24	0.41	1.04	0.13

Iteration 4:

vertex	0	1	2	3	4	5	6	7
distance	0	0.42	0.26	0.21	-0.55	0.1	0.73	-0.18

Iteration 5:

vertex	0	1	2	3	4	5	6	7
distance	0	0.11	0.26	-0.1	-0.86	-0.21	0.42	-0.49

Iteration 6:

vertex	0	1	2	3	4	5	6	7
distance	0	-0.2	0.26	-0.41	-1.17	-0.52	0.11	-0.8

Iteration 7:

vertex	0	1	2	3	4	5	6	7
distance	0	-0.51	0.26	-0.72	-1.48	-0.85	-0.2	-1.11

After 7 iterations, I check the distance of vertex 5 and find it still can be updated by a smaller number. So, this graph does have a negative weighted loop.

Q5

Python code is another file.
Both DFS and BFS visited all vertexes.

Q6

Python code is another file.
Results:
For Q4b:

```
Vertex  Distance from Source
0       0
1       1.2000000000000002
2       0.26
3       0.9900000000000001
4       0.38
5       0.8800000000000001
6       0.58
7       0.6000000000000001
```

For Q4a:

```
Vertex  Distance from Source
0       0
1       1.05
2       0.26
3       0.9900000000000001
4       0.38
5       0.73
6       1.5100000000000002
7       0.6000000000000001
```

Both results are not right. Because Djikstra's algorithm only iterate every vertex once and fix the distance from the start, doesn't update the distance, so Djikstra's algorithm doesn't fit graph with negative edge wiehgts.