

Assignment 4

Name:Tian Zhou

NETID:tz164

1.

In the program, we store the adjacent information in the list. While connecting edges, the AddEdge() function add adjacent element in the list (both sides for undirected graph). Using a BFS algorithm in this program. And run the acyclic() function to return the result. While any unmarked vertex adjacent at least two marked vertices, the graph is not acyclic. 0 represents a not acyclic graph and 1 represents a acyclic graph.

The result for given data-set is : The given graph is not an acyclic graph.

2.

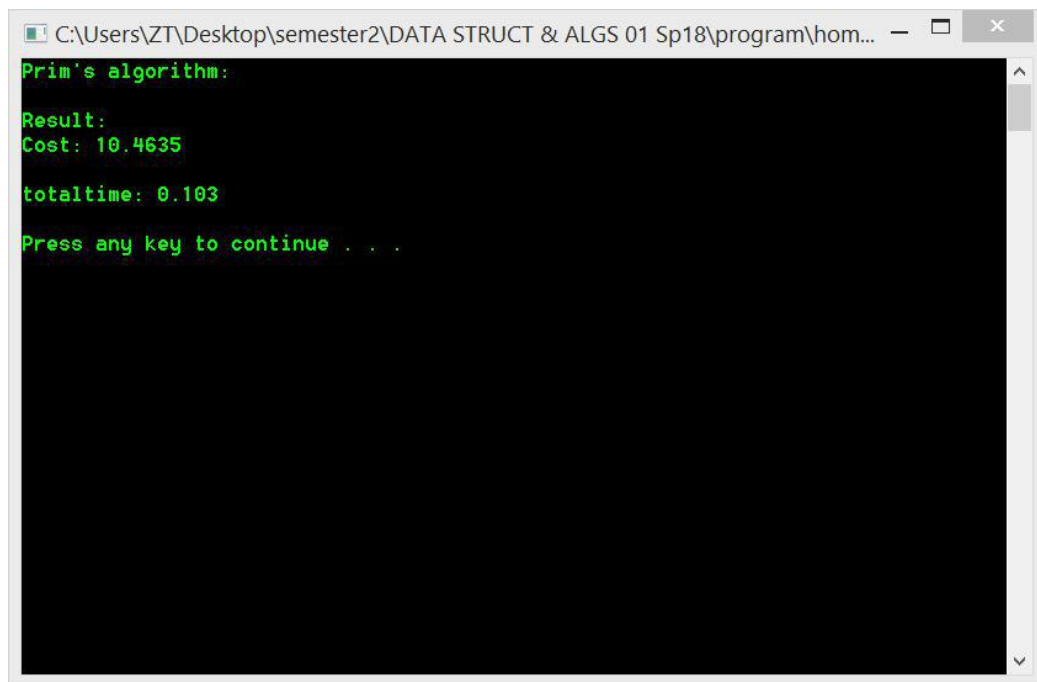
The implementation of Prim's and Kruskal's algorithms on the given graph is in the Q2 file.

The screen shot of two algorithms is Figure 4-2-1 and Figure 4-2-2. The program just record the searching part time for each algorithm and compare with each other(For Kruskal's algorithm, we also include the sorting time). The total time for Prim's algorithm is 0.103s and Kruskal's algorithm is 0.021. It's obvious that Kruskal's algorithm performs better than Prim's algorithm.

Reason: Prim's algorithm needs to compare every adjacent edges for each connection, but Kruskal's algorithm sort the weight of edges at the beginning.

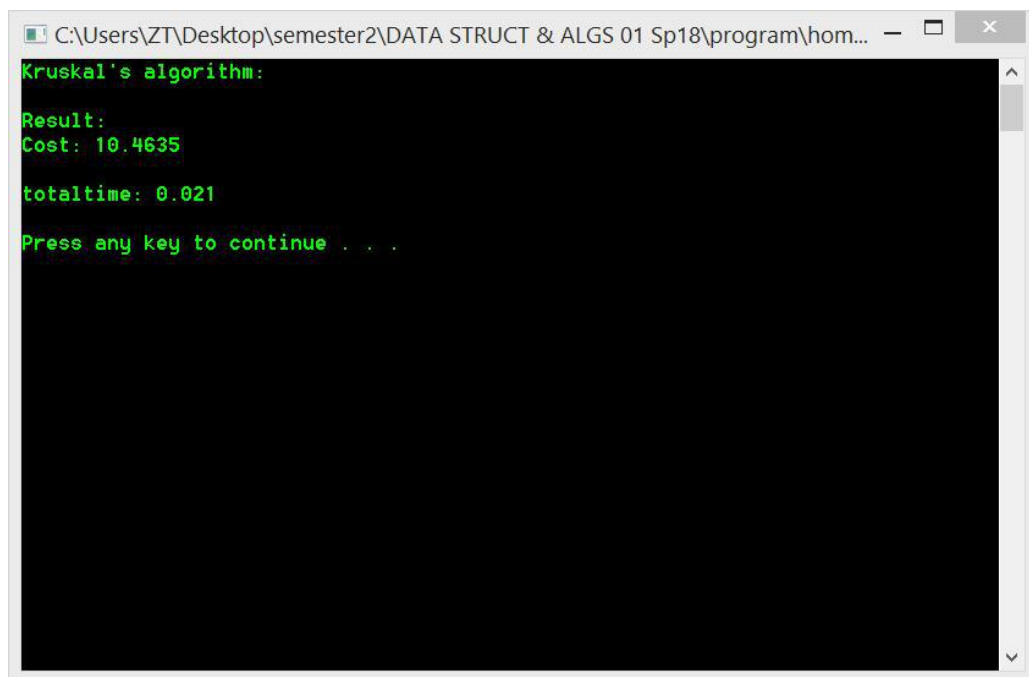
In that case, kruskal's algorithm performs better in a sparse graph because it's easy to distinguish the cycle and sorting for weight of edges is much easier. Prim's algorithm performs better in a dense graph because the sorting of large number of edges also cost a lot of time in Kruskal's algorithm. The number of vertices in the given graph is 250 and the number of edges in the given graph is 1273. It is a sparse graph, so the Kruskal's algorithm performs better than Prim's algorithm in the given graph.

Figure 4-2-1



```
C:\Users\ZT\Desktop\semester2\DATA STRUCT & ALGS 01 Sp18\program\hom...  
Prim's algorithm:  
Result:  
Cost: 10.4635  
totaltime: 0.103  
Press any key to continue . . .
```

Figure 4-2-2



```
C:\Users\ZT\Desktop\semester2\DATA STRUCT & ALGS 01 Sp18\program\hom...  
Kruskal's algorithm:  
Result:  
Cost: 10.4635  
totaltime: 0.021  
Press any key to continue . . .
```

3.

An edge-weighted directed acyclic graph could be Topological sort as below:

5-1-3-6-4-7-0-2.

According to the order of Topological sort, we can draw the shortest and longest path of DAG.

According to the Topological sort, the relax of each edge could result in:

Table 4.3.1 Shortest Path

				0	1	2	3	4	5	6	7
5	4	0.35	dis					0.35	0		
			path					54	5		
5	7	0.28	dis					0.35	0		0.28
			path					54	5		57
5	1	0.32	dis		0.32			0.35	0		0.28
			path		51			54	5		57
1	3	0.29	dis		0.32		0.61	0.35	0		0.28
			path		51		513	54	5		57
3	7	0.39	dis		0.32		0.61	0.35	0		0.28
			path		51		513	54	5		57
3	6	0.52	dis		0.32		0.61	0.35	0	1.13	0.28
			path		51		513	54	5	5136	57
6	2	0.4	dis		0.32	1.53	0.61	0.35	0	1.13	0.28
			path		51	51362	513	54	5	5136	57
6	0	0.58	dis	1.71	0.32	1.53	0.61	0.35	0	1.13	0.28
			path	51360	51	51362	513	54	5	5136	57
6	4	0.93	dis	1.71	0.32	1.53	0.61	0.35	0	1.13	0.28
			path	51360	51	51362	513	54	5	5136	57
4	7	0.37	dis	1.71	0.32	1.53	0.61	0.35	0	1.13	0.28
			path	51360	51	51362	513	54	5	5136	57
4	0	0.38	dis	0.73	0.32	1.53	0.61	0.35	0	1.13	0.28
			path	540	51	51362	513	54	5	5136	57
7	2	0.34	dis	0.73	0.32	0.62	0.61	0.35	0	1.13	0.28
			path	540	51	572	513	54	5	5136	57
0	2	0.26	dis	0.73	0.32	0.62	0.61	0.35	0	1.13	0.28
			path	540	51	572	513	54	5	5136	57

As we can see in the above table. The shortest path (source is 5) for each vertex is:

	0	1	2	3	4	5	6	7
dis	0.73	0.32	0.62	0.61	0.35	0	1.13	0.28
path	540	51	572	513	54	5	5136	57

Table 4.3.2 Longest Path

				0	1	2	3	4	5	6	7
5	4	-0.3 5	dis					-0.35	0.00		
			path					54	5		
5	7	-0.2 8	dis					-0.35	0.00		-0.28
			path					54	5		57
5	1	-0.3 2	dis		-0.32			-0.35	0.00		-0.28
			path		51			54	5		57
1	3	-0.2 9	dis		-0.32		-0.61	-0.35	0.00		-0.28
			path		51		513	54	5		57
3	7	-0.3 9	dis		-0.32		-0.61	-0.35	0.00		-1.00
			path		51		513	54	5		5137
3	6	-0.5 2	dis		-0.32		-0.61	-0.35	0.00	-1.13	-1.00
			path		51		513	54	5	5136	5137
6	2	-0.4 0	dis		-0.32	-1.53	-0.61	-0.35	0.00	-1.13	-1.00
			path		51	51362	513	54	5	5136	5137
6	0	-0.5 8	dis	-1.71	-0.32	-1.53	-0.61	-0.35	0.00	-1.13	-1.00
			path	51360	51	51362	513	54	5	5136	5137
6	4	-0.9 3	dis	-1.71	-0.32	-1.53	-0.61	-2.06	0.00	-1.13	-1.00
			path	51360	51	51362	513	51364	5	5136	5137
4	7	-0.3 7	dis	-1.71	-0.32	-1.53	-0.61	-2.06	0.00	-1.13	-2.43
			path	51360	51	51362	513	51364	5	5136	513647
4	0	-0.3 8	dis	-2.44	-0.32	-1.53	-0.61	-2.06	0.00	-1.13	-2.43
			path	513640	51	51362	513	51364	5	5136	513647
7	2	-0.3 4	dis	-2.44	-0.32	-2.77	-0.61	-2.06	0.00	-1.13	-2.43
			path	513640	51	5136472	513	51364	5	5136	513647
0	2	-0.2 6	dis	-2.44	-0.32	-2.77	-0.61	-2.06	0.00	-1.13	-2.43
			path	513640	51	5136472	513	51364	5	5136	513647

As we can see in the above table. The longest path (source is 5) for each vertex is:

	0	1	2	3	4	5	6	7
dis	-2.44	-0.32	-2.77	-0.61	-2.06	0.00	-1.13	-2.43
path	513640	51	5136472	513	51364	5	5136	513647

4.

Relax each edge in the graph for $n-1$ times (n is the number of vertex in the graph).

Dis[] array record the shortest path from source for each vertex.

Set $dis[5]=0$ and any other $dis[]=\infty$.

Each loop would relax every edge in the graph according to the order of input.

If $dis[v]$ does not change during pass i , no need to relax any edge pointing from v in the pass $i+1$.

a)

Table 4.4.1

loop	1	2	3
dis[0]	-0.27	-0.27	-0.27
dis[1]	0.32	0.32	0.32
dis[2]	*	-0.07	-0.07
dis[3]	0.61	0.61	0.61
dis[4]	-0.12	-0.12	-0.12
dis[5]	0	0	0
dis[6]	1.13	1.13	1.13
dis[7]	0.28	0.25	0.25

The program end before the loop reach the number of vertex. The graph has no negative cycle.

The table above is the record of dis[] array for each loop. Then the result dis[] array represents the shortest path from source 5.

b)

Table 4.4.2

loop	1	2	3	4	5	6	7	8
dis[0]	1.2	0.89	0.58	0.27	-0.04	-0.35	-0.66	-0.97
dis[1]	0.31	0	-0.31	-0.62	-0.93	-1.24	-1.55	-1.86
dis[2]	*	1.02	0.71	0.4	0.09	-0.22	-0.53	-0.84
dis[3]	0.1	-0.21	-0.52	-0.83	-1.14	-1.45	-1.76	-2.07
dis[4]	-0.66	-0.97	-1.28	-1.59	-1.9	-2.21	-2.52	-2.83
dis[5]	-0.01	-0.32	-0.63	-0.94	-1.25	-1.56	-1.87	-2.18
dis[6]	0.62	0.31	0	-0.31	-0.62	-0.93	-1.24	-1.55
dis[7]	-0.29	-0.6	-0.91	-1.22	-1.53	-1.84	-2.15	-2.46

After running $n-1$ (n is the number of vertex, here $n=8$) loops, the value of dis[v] array is not fixed. Then run another loops for n th time, the value of dis[v] array change again. Then we could say the given graph has a negative cycle.

5.

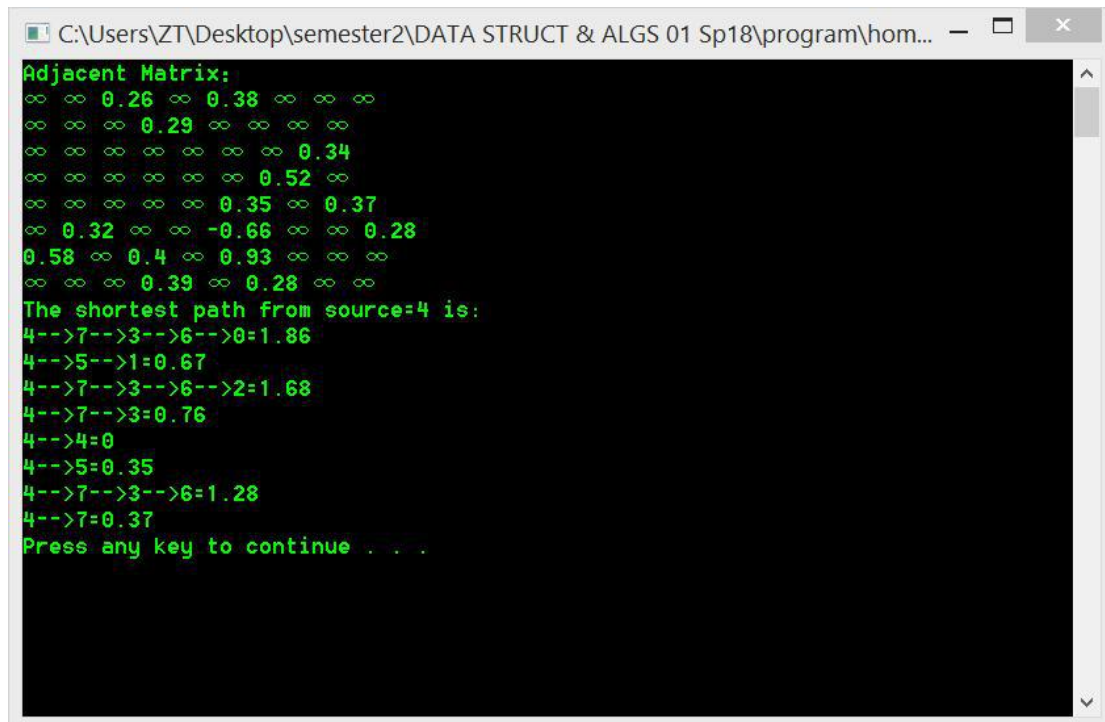
The implementation of DFS and BFS traversal for the data-set of the undirected road net work of New York City is in the Q5 file.

First of all, connect all the edges and record the weight for each edge. Then run the DFS() function and BFS() function. The DFS() function is a DFS traversal and BFS() function is a BFS traversal.

6.

The implementation of shortest path using Dijkstra's Algorithm for the graph in HW4 Q4(b) is in the Q6 file. The result of shortest path is :

Figure 4-6-1(b)



```
C:\Users\ZT\Desktop\semester2\DATA STRUCT & ALGS 01 Sp18\program\hom...
Adjacent Matrix:
∞ ∞ 0.26 ∞ 0.38 ∞ ∞ ∞
∞ ∞ ∞ 0.29 ∞ ∞ ∞ ∞
∞ ∞ ∞ ∞ ∞ ∞ ∞ 0.34
∞ ∞ ∞ ∞ ∞ ∞ 0.52 ∞
∞ ∞ ∞ ∞ ∞ 0.35 ∞ 0.37
∞ 0.32 ∞ ∞ -0.66 ∞ ∞ 0.28
0.58 ∞ 0.4 ∞ 0.93 ∞ ∞ ∞
∞ ∞ ∞ 0.39 ∞ 0.28 ∞ ∞
The shortest path from source=4 is:
4-->7-->3-->6-->0=1.86
4-->5-->1=0.67
4-->7-->3-->6-->2=1.68
4-->7-->3=0.76
4-->4=0
4-->5=0.35
4-->7-->3-->6=1.28
4-->7=0.37
Press any key to continue . . . .
```

In this part, the program set point 4 as a source. Actually there is a negative cycle in the graph 4(b), the graph has no shortest path. However the program can't distinguish the negative cycle in the graph. And the edge return to the source point won't be used in the Dijkstra's Algorithm, so the negative cycle is ignored by the program. The result of program here is the shortest path in the graph from the source point 4 while the negative edge 5-4 doesn't exist. In conclusion, the result of shortest path is not correct here.

Figure 4-6-2(a)



```

C:\Users\ZT\Desktop\semester2\DATA STRUCT & ALGS 01 Sp18\program\hom...
Adjacent Matrix:
∞ ∞ 0.26 ∞ 0.38 ∞ ∞ ∞
∞ ∞ ∞ 0.29 ∞ ∞ ∞ ∞
∞ ∞ ∞ ∞ ∞ ∞ ∞ 0.34
∞ ∞ ∞ ∞ ∞ ∞ 0.52 ∞
∞ ∞ ∞ ∞ ∞ 0.35 ∞ 0.37
∞ 0.32 ∞ ∞ 0.35 ∞ ∞ 0.28
-1.4 ∞ -1.2 ∞ -1.25 ∞ ∞ ∞
∞ ∞ ∞ 0.39 ∞ 0.28 ∞ ∞
The shortest path from source=4 is:
4-->7-->3-->6-->0=-0.12
4-->5-->1=0.67
4-->7-->3-->6-->2=0.08
4-->7-->3=0.76
4-->4=0
4-->5=0.35
4-->7-->3-->6=1.28
4-->7=0.37
Press any key to continue . . .
  
```

For the graph in HW4 Q4(a), the adjacent matrix has negative element, which means the graph has negative weight edges. In general, Dijkstra's Algorithm can't solve shortest path problem for the digraph with a negative cycle. However the Dijkstra's Algorithm can't make sure that the result path is the shortest path. But for this graph, the Dijkstra's Algorithm still output the shortest path of the graph, because there is no multi-edge path shorter than single-edge path in the graph with negative weights. In that case, Dijkstra's Algorithm could still output the shortest path.