CSE 373 HW5 Write Up

Chongyi Xu

May 18, 2017

1. **Topological Sort:** Determine a topological ordering for the following directed acylic graph. Show your work

	A	B	C	D	E	F	G
removed	x	x	x	x	x	x	x
degree	0	$\cancel{1}$ 0	$\cancel{1}$ 0	$\cancel{2}$ $\cancel{1}$ 0	$\cancel{2}$ $\cancel{1}$ 0	$\cancel{1}$ 0	0

One possible ordering will be [A, G, F, B, D, E, C]

2. **Minimum Spanning Algorithm:** Use the following graph for both of the following minimum spanning tree algorithms.

   (a) Build a minimum spanning tree for the graph using Kruskals algorithm. Number each edge according to when it is entered into the minimum spanning tree. THe first edge will get number 1, etc. Show the ordering of the edges you consider by showing the state of the pending set of edges to consider.

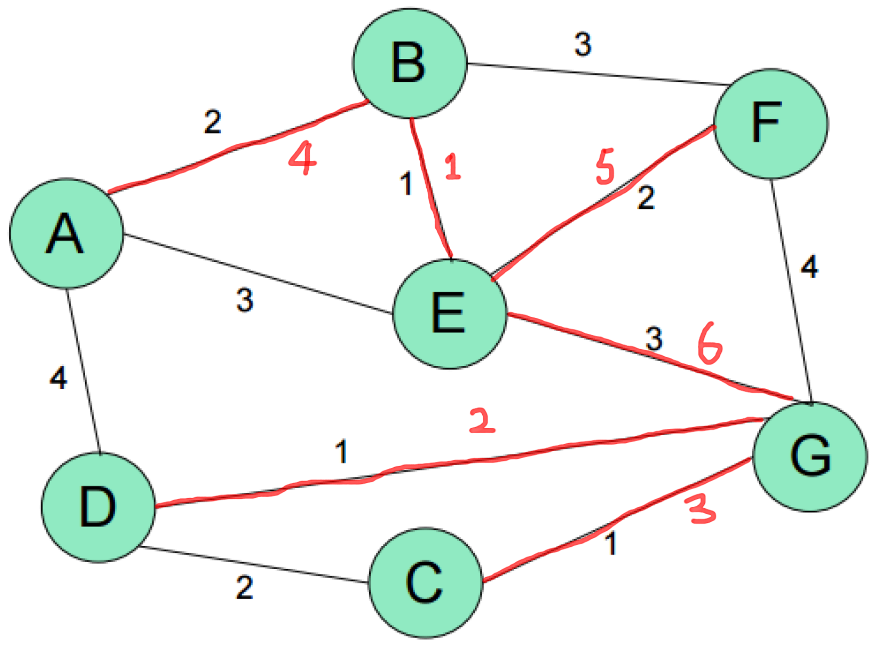
   Edges in sorted order:

   cost 1: (B, E) (D, G) (C, G)

   cost 2: (A, B) (E, F) (C, D)

   cost 3: (B, F) (A, E) (E, G)

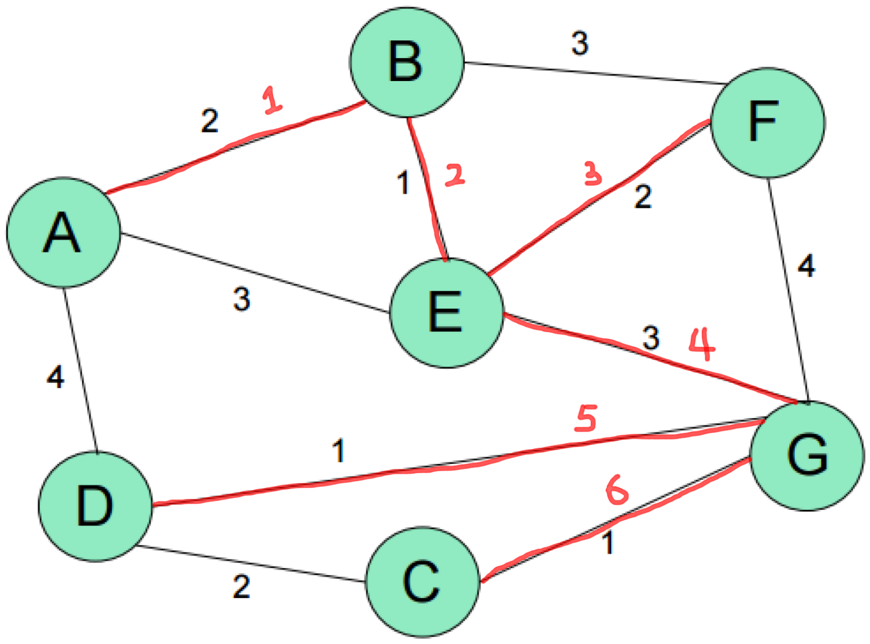
   cost 4: (A, D) (F, G)



So the ordering will be [(B, E), (D, G), (C, G), (A, B), (E, F), (E, G)]

(b) Now build a MST for the graph using Prims algorithm, starting with vertex A.

	known?	cost	previous
A	✓	0	
B	✓	2	A
C	✓	1	G
D	✓	~~4~~ 1	~~A~~ G
E	✓	~~3~~ 1	~~A~~ B
F	✓	~~3~~ 2	~~B~~ E
G	✓	3	E

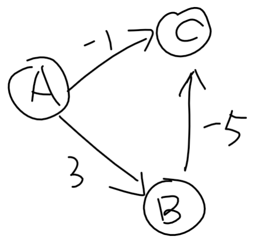


3. **Dijkstra's and Negative Edges:**

(a) If there is more than one minimum cost path from v to w, will Dijkstra's Algorithm always find the path with the fewest edges? If not, explain in a few sentences how to modify Dijkstra's Algorithm so that if there is more than one minimum path from v to w, a pth with the fewest edges is chosen. Assume no negative weight edges or negative weight cycles.

It will not always find the shortest path since when we are comparing costs, we just compare the value without considering the number of edges. If the costs are the same, the second path will be ignored. If we would like to compare the number of edges, we could make a field to keep track of the least number of edges. And every time we find two paths with the same cost, we could then compare the number of edges to decide which path we should pick.

(b) Given an example where Dijkstra's Algorithm gives the wrong answer in the presence of a negaive cost edge but no negative-cost cycles.



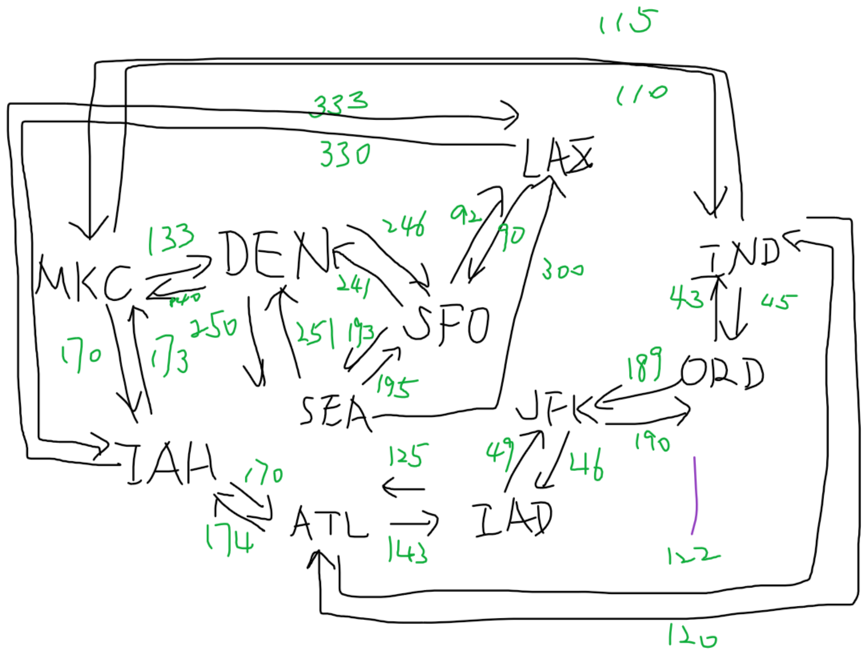
In this case, we first check **C** as known with cost -1, then we check **B** as known with cost 3. But after we check **B**, we find another path to **C** which has a lower cost comparing to what we have assigned to **C**, which is -2. So Dijkstra's Algorithm fails here.

(c) Suppose you are given a graph that has negative-cost edges but no negative-cost cycles. Consider the following strategy to find shortest paths in this graph: Uniformly add a constant k to the cost of every edge, so that all costs become non-negative, then run Dijstra's Algorithm and return that result with the edge costs reverted back to their original values.

$$\begin{tikzcd}
A \arrow[d, "1"'] & & & & A \arrow[d, "4"'] & & \\
D \arrow[rr, "-1"] \arrow[dr, "0"'] & & G \arrow[dl, "-1"] \arrow[rr, "k=+3"'] & & D \arrow[rr, "2"] \arrow[dr, "3"'] & & G \arrow[dl, "2"] \\
& C & & & & C &
\end{tikzcd}$$

This method fails in this case when we want to find the shortest path from **A** to **C**. In the original graph (with negative-cost edges), the shortest path will be **A** $\rightarrow$ **D** $\rightarrow$ **G** $\rightarrow$ **C** with cost -1, but after the transformation, the shortest path will be **A** $\rightarrow$ **D** $\rightarrow$ **C** with cost 7.
In general, it fails because the transformation is adding a constant to every edge, which means the path with more edges will have a greater increasement rather than the one with less edges.

4. Describe how you tested your `shortestPath` method. Explain any difficulties in implementation you may have experinced and how testing helped find the problem, if applicable.
Firstly, I draw the graph to have a clear image of what am I testing.



Then I do a lot of tests. In particulr, I tested the following special cases:

- When the start point and the destination point are the same
- Make multiple times of different calls to check if the original data is not changed
- Try different start points

```
Path shortest = g.shortestPath(a, b);
if (shortest == null) {
        System.out.println(a.toString() + " is not reachable from " + b.toString());
} else {
        System.out.println("The least cost will be " + shortest.cost);
        System.out.println("The shortest path will be " + shortest.vertices.toString());
}
```

As I tested my `shortestPath` method, I found out that my code does not successfully keep the original data(vertices) unchanged. This problem was found when I tried call two times of `shortestPath` method, the second time gave me the wrong path. Then I went back to my method and created another container(vertex) to do Dijkstra's Algorithm. In such way, my original data will not be changed so that I could make multiple calls.