Jilin Zheng // U49258796

EC330 Homework #8

1. [MST, 16 points]
Each of the following algorithms takes a connected weighted graph G(V,E) as input, and returns a set of edges T. For each algorithm, either explain why T is a minimum spanning tree or give a counter-example.

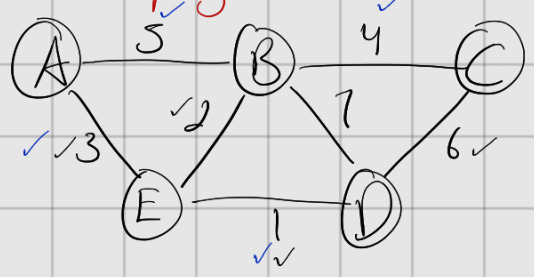
a. MAYBE-MST-A(G)

```
T = empty
for each edge e, taken in arbitrary order
    if T ∪ {e} has no cycles
        T = T ∪ {e}
return T
```

b. MAYBE-MST-B(G)

```
sort the edges into non-increasing order of edge weights w
T = E
for each edge e, in non-increasing order by weight
    if T – {e} is a connected graph
        T = T – {e}
return T
```


a. T is NOT a minimum spanning tree (MST)



$T = \{5, 4, 3, 1\}$; $5+4+3+1 = 13$

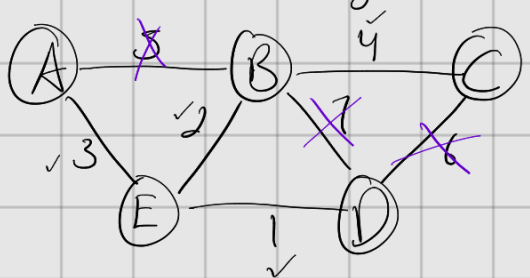
↳ NOT MST since $13 > 10$

$MST = \{1, 2, 3, 4\}$; $1+2+3+4 = 10$

(with Prim's Algorithm (start @ D))

visited = {D, E, B, A, C}

b. T is a MST because the extra edges are removed in non-increasing order; we preserve the edges with minimum weights



$T = \{7, 6, 5, 4, 3, 2, 1\}$

↓ remove 7 edge

$= \{6, 5, 4, 3, 2, 1\}$

↓ remove 6 edge

$= \{5, 4, 3, 2, 1\}$

↓ remove 5 edge

$= \{4, 3, 2, 1\}$; $4+3+2+1 = 10$, MST ✓

$MST = \{1, 2, 3, 4\}$; $1+2+3+4 = 10$

(Kruskal's Algorithm)

visited = {E, D, B, A, C}

removing any more edges results in an unconnected graph

2. [MST, 12 points]
Give an algorithm to find a maximum spanning tree in a connected undirected graph.

Apply a modified Kruskal's Algorithm.
Rather than sorting edges in nondecreasing order, sort in non INCREASING order.
Implement the rest of Kruskal's in the original way: add edges that do not form cycles until there are $|V|-1$ edges ($|V|$ is # of vertices).

3. [Single-Source Shortest Path, 12 points]
Design an efficient algorithm that outputs the overall number of paths that exist in a given directed acyclic graph. Analyze the runtime of your algorithm.

1. Topological Sort the graph — $\Theta(V+E)$
2. Create an array to store # of paths to each vertex — $\Theta(V)$
   Initialize the first node's # of paths to one, and the rest to zero
3. Traverse the vertices in topological order, updating the # of paths array — $\Theta(V+E)$
   For each vertex, add the # of paths @ the current vertex to the # of paths of its neighbors
4. Simply index into the # of paths array for the desired destination node to output the overall # of paths — $\Theta(1)$

Total Runtime: $\Theta(V+E)$

4. [Single source shortest path, 10 points]
Bellman Ford's shortest path algorithm has a runtime of $\Theta(VE)$. Explain why this is the case, and suggest a method of detecting whether the algorithm may stop early.

Bellman Ford has a runtime of $\Theta(VE)$ because each edge is relaxed $V-1$ times (V asymptotically).
The algorithm may stop early if for a particular iteration, we never update the distances to each vertex.