Shreya Patel 11/28/18

# Assignment 5

1.

a. **Kruskal's Algorithm**

| Step # | Edge | Edge weight |
|---|---|---|
| 1 | 7 and 6 | 1 |
| 2 | 2 and 8 | 2 |
| 3 | 6 and 5 | 2 |
| 4 | 0 and 1 | 4 |
| 5 | 2 and 5 | 4 |
| 6 | 2 and 3 | 7 |
| 7 | 0 and 7 | 8 |
| 8 | 3 and 4 | 9 |

**b. Prim's Algorithm**

| Step # | Edge | Edge weight |
|---|---|---|
| 1 | 8 to 2 | 2 |
| 2 | 2 to 5 | 4 |
| 3 | 5 to 6 | 2 |
| 4 | 6 to 7 | 1 |
| 5 | 2 to 3 | 7 |
| 6 | 7 to 0 | 8 |
| 7 | 0 to 1 | 4 |
| 8 | 3 to 4 | 9 |

2. Answer:
   a. Kruskal's algorithm starts with picking the edge with the least amount of weight and continuing this pattern until all edges are connected to each other in some way. The way that this algorithm detects a cycle by not selecting the edges that have already been visited even if they turn out to be lower in weight than the other remaining ones. Since it never visits a vertex more than once a cycle would not be able to form.
   b. Prim's algorithm starts out with picking an arbitrary node and then following the edge lowest in weight from that picked node. And then it continues by selecting the lowest weight edges that can be reached with the already visited vertices. Since the vertex that have already been visited would not get selected again regardless of their edge weight.
   c. If you used Prim's algorithm cycle detection in Kruskal's algorithm it would work because they both work towards creating a minimum spanning tree. Both algorithms also work through finding the lowest edge weight and not repeating vertices that they have already visited.

| Step # | Edge | Edge weight |
|---|---|---|
| 1 | 7 and 6 | 1 |

| | | |
|---|---|---|
| 2 | 6 and 5 | 2 |
| 3 | 5 and 2 | 4 |
| 4 | 2 and 8 | 2 |
| 5 | 2 and 3 | 7 |
| 6 | 0 and 7 | 8 |
| 7 | 0 and 1 | 4 |
| 8 | 3 and 4 | 9 |

If it was just Kruskal algorithm before the 5 to 2 it would be 2 and 8 since its lower than 5 and 2. And also 0 and 1 would before 2 and 3. So using Prim's algorithm of cycle detection does violate some rules of the Kruskal algorithm.

3. Trying to use Kruskal or Prim's algorithm in a directed graph doesn't work to create a minimum spanning tree most of the times because a directed graph is already restricted in its directions. A minimum spanning tree tries to find a way to follow the edges with the least weight and to do that it has to go back and tweak its path several times. In a directed graph there is only one way that the path can go so the restriction doesn't allow Kruskal's or Prim's algorithm to produce a minimum spanning tree.

**4. Prim's algorithm**

- Key = edge; priority = weight of edge.
- DELETE-MIN to determine next edge e = v–w to add to T.
- If both endpoints v and w are marked (both in T), disregard.
- Otherwise, let w be the unmarked vertex (not in T):
    - add e to T and mark w
    - add to PQ any edge incident to w (assuming other endpoint not in T)
- Start with vertex 0 and greedily grow tree T. (In Eager Prim's the vertex starts arbitrarily)
- Add to T the min weight edge with exactly one endpoint in T.
- Repeat until V – 1 edges.

**Eager Prim's algorithm**

- Delete min vertex v; add its associated edge e = v–w to T.
- Update PQ by considering all edges e = v–x incident to v
    - ignore if x is already in T
    - add x to PQ if not already on it
    - decrease priority of x if v–x becomes lightest edge connecting x to T
- Insert a key associated with a given index.
- Delete a minimum key and return associated index.
- Decrease the key associated with a given index.

Time and Space Complexities:

- Naïve Prim's algorithm has time complexity of O(ElogV)
- Eager Prim's algorithm has time complexity of O(E + logV)
- The Eager Prim's algorithm is a much faster algorithm because it incorporates the Fibonacci heap priority queue.

5. Minimum Spanning Tree validation
    a. Edge is added to the graph G
        i. Compare the weight of the new added edge with the other edges (E)
        ii. If the weight of the new added edge is greater than the last added edge
            1. The tree is still a MST
        iii. If the weight of the new added edge is smaller than any one of the existing edges
            1. The tree is not a MST anymore

    b. An edge of the tree is made smaller
        i. Compare the weight of the changed edge to the other edges
        ii. If the weight of the edge is still **larger** than the weight for the edges of the vertices visited **before** it then
            1. The tree is still a MST
        iii. If the weight of the edge is **smaller** than the weight for the edges of them vertices visited **before** it then
            1. The tree is not a MST

    c. An edge of the tree is made larger
        i. Compare the weight of the changed edge to the other edges
        ii. If the weight of the edge is **smaller** than the weight for the edges of the vertices visited **after** it then
            1. The tree is still a MST
        iii. If the weight of the edge is **larger** than the weight for the edges of the vertices visited **after** it then
            1. The tree is not a MST anymore

6.
    a. Bellman Ford Algorithm:

**1st pass:**

| Vertex | distTo[] |
|---|---|
| A | 0 |
| 1 | 6 |
| 2 | 5 |
| 3 | 10 |
| 4 | 13 |
| 5 | 11 |
| 6 | 16 |
| 7 | 18 |
| 8 | 21 |
| B | 26 |

| Vertex | edgeTo[] |
|---|---|
| A | - |
| 1 | $A \rightarrow 1$ |
| 2 | $A \rightarrow 2$ |

| | |
|---|---|
| 3 | $1 \rightarrow 3$ |
| 4 | $1 \rightarrow 4$ |
| 5 | $2 \rightarrow 5$ |
| 6 | $3 \rightarrow 6$ |
| 7 | $5 \rightarrow 7$ |
| 8 | $4 \rightarrow 8$ |
| B | $5 \rightarrow B$ |

**2nd pass:**

| Vertex | distTo[] |
|---|---|
| A | 0 |
| 1 | 6 |
| 2 | 5 |
| 3 | 7 |
| 4 | 13 |
| 5 | 11 |
| 6 | 13 |
| 7 | 18 |
| 8 | 21 |
| B | 21 |

| Vertex | edgeTo [] |
|---|---|
| A | - |
| 1 | $A \rightarrow 1$ |
| 2 | $A \rightarrow 2$ |
| 3 | $2 \rightarrow 3$ |
| 4 | $1 \rightarrow 4$ |
| 5 | $2 \rightarrow 5$ |
| 6 | $3 \rightarrow 6$ |
| 7 | $5 \rightarrow 7$ |
| 8 | $4 \rightarrow 8$ |
| B | $7 \rightarrow B$ |

**3rd pass:** No change

b. Djikstra's Algorithm:

| Vertex | distTo [] |
|---|---|
| A | 0 |
| 1 | 6 |
| 2 | 5 |
| 3 | 7 |
| 4 | 13 |
| 5 | 11 |
| 6 | 13 |

| | |
|---|---|
| 7 | 18 |
| 8 | 21 |
| B | 21 |

| Vertex | edgeTo [] |
|---|---|
| A | - |
| 1 | $A \rightarrow 1$ |
| 2 | $A \rightarrow 2$ |
| 3 | $2 \rightarrow 3$ |
| 4 | $1 \rightarrow 4$ |
| 5 | $2 \rightarrow 5$ |
| 6 | $3 \rightarrow 6$ |
| 7 | $5 \rightarrow 7$ |
| 8 | $4 \rightarrow 8$ |
| B | $7 \rightarrow B$ |

| Vertex | relaxCount |
|---|---|
| A | 0 |
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |
| 4 | 1 |
| 5 | 2 |
| 6 | 3 |
| 7 | 1 |
| 8 | 3 |
| B | 3 |

c. Topological order

| Vertex | distTo [] |
|---|---|
| A | 0 |
| 1 | 6 |
| 2 | 5 |
| 3 | 7 |
| 4 | 13 |
| 5 | 11 |
| 6 | 13 |
| 7 | 18 |
| 8 | 21 |
| B | 21 |

| Vertex | edgeTo [] |
|---|---|

| A | - |
|---|---|
| 1 | $A \rightarrow 1$ |
| 2 | $A \rightarrow 2$ |
| 3 | $2 \rightarrow 3$ |
| 4 | $1 \rightarrow 4$ |
| 5 | $2 \rightarrow 5$ |
| 6 | $3 \rightarrow 6$ |
| 7 | $5 \rightarrow 7$ |
| 8 | $4 \rightarrow 8$ |
| B | $7 \rightarrow B$ |

| Vertex | relaxCount |
|---|---|
| A | 0 |
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |
| 4 | 1 |
| 5 | 1 |
| 6 | 3 |
| 7 | 2 |
| 8 | 3 |
| B | 3 |

7. Topological sort requires a DAG because this sort is like a prerequisite where the previous node in a list has to be visited before the nodes connected to the first node can be reached. In a DAG there are no directed cycles so each edge is directed from one vertex to another. Since topological sort needs to visit each node in a sequence DAG is required. You can't run the algorithm on a non-DAG because a non-DAG doesn't follow the prerequisite rule.

- Consider vertices in topological order.
- Relax all edges adjacent from that vertex
- Let w be next vertex (in topological order) added to T.
- Let P be the $s \rightsquigarrow w$ path of length distTo[w].
- Consider any other $s \rightsquigarrow w$ path $P'$.
- $P'$ must be a path to a vertex in T plus one extra edge, say $x \rightarrow w$

8. Dijkstra's algorithm doesn't need a cycle detection because it checks every node to get a path that has the least weighted edge. Since it only depends on the weight of the edge there is no need for a cycle detection since a cycle wouldn't occur.

9. Dijkstra's algorithm would fail at negative edge weights because the vertices are considered in increasing order of the distance from the ending vertex. A negative edge value can't be a potential distance from the ending vertex so that's why negative edge vertices aren't allowed in Dijkstra's algorithm.

- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest distTo[] value).
- Add vertex to tree and relax all edges adjacent from that vertex.

10.

a. Ford Fulkerson Algorithm

| Path # | Path Sequence | Edge weight sequence |
|---|---|---|
| 1 | S → 1 → 4 → t | --- → 10/11 → 10/18 → 10/10 |
| 2 | S → 2 → 5 → t | --- → 16/22 → 16/17 → 16/16 |
| 3 | S → 3 → 6 → t | --- → 10/10 → 10/16 → 10/16 |
| 4 | S → 2 → 5 → 6 → t | --- → 17/22 → 17/17 → 1/5 → 11/16 |
| 5 | S → 2 → 3 → 6 → t | --- → 21/22 → 4/4 → 14/16 → 15/16 |

b. The mincut consists of the vertices s, 2, 1, 5

c. The capacity of the mincut is 26.