Yong Zhou
Hw10

Q1. Algorithm: Run through all the nodes by topological order in reversed graph from s to t.

```
Initial: n <- length of nodes;
        Array count [n]
        For each node k in G
                 Node value <-  0
                count[k] <- 0
        Reversed graph G*

        source node s = 1;
        if number node in G is 1:
                return 0;
        For i in node (1 to n -1):
                Foreach node(in adjacency list) of i in G* :
                        count[i] += count[k]
        Return count[i]
```

Complexity:

- Loop through all the nodes by topological order takes O(V+E), calculation for each step by each edge takes constant time. Initial all the node value take O(n). reverse G take O (V+E). Total running time takes O(V+E).

Correctness of prove.

- Prove by induction.
    - Base case: If number of the node is 1, the incoming edge is zero, the email receive is 0.
    - Claim: it holds of the total amount of copies to the kth node is n, the sum of the (k+1) th node will be $\sum \text{all parents nodes of } (k+1)$, include value of the node n.
    - Inductive step:
        - By the algorithm. The count is accumulated from parents' nodes. For (k+1) node, it will loop through all the incoming edges and get sum from each parents node (include value of node k) to (k+1) node. Therefore, the count(k+1) holds.

Q2,

- 2,1: to the graph G, we could run BFS from the residual graph s to t. If we find a single path from s to t, then f is not max flow, otherwise, f is a max flow.

- Algorithm:

```
Initial: n <- number of the node in G.
Def( G, c, s, t, f):
    G* <- a residual graph of G.
        Run BFS on G* from s to t:
            If exist a single path from s to t:
                Return it is not a max flow;

        return it is a max flow;
```

Complexity:

- run BFS takes O(V+E), and loop through new G* take O(V+E). Overall running time is O(V+E)

Proof:

- By the residual graph of max-flow and min-cut (P.350)

2.2:

- By increasing 1 of the edge weights. The maximum flow can only increase or not change.
- There are 2 cases.
  - 1, the increase edge e* is not part of the minimum cut. f(e*) < c(e*). The old max flow remains the max flow in this new setup.
  - 2. Edge e* is part of min cut. increasing its capacity will increase the value of the cut, and it will still be part of the min cut. The total value of the new flow will be increase by one.
    - 3, if there are multiple min cuts: increase one of the e*, will not increase by 1.

2.3:

- On the residual graph.

```
Def (G, c, f):
    G* <- residual of G;
    Run BFS in residual graph of G* start s to t
    if there exist a path from s to t:
        return updated the flow
    Else:
        return max flow not change
```

Complexity:

Run BFS takes O(V+E),

Proof correctness:

- By the residual graph G* with the max flow, before increase e*, when we run BFS from s, we could not find a path from s to t by the definition of the residual graph on max flow. On the other hand, if we find a path in residual graph G*, that must be caused by the e*(increases by 1).

2.4

- By reducing 1 of the edge weights, the maximum flow can only reduce and will not increase.
    - There are two cases:
        - 1, edge e* is not part of the minimum cut. f(e*) < c(e*). The old max flow remains the max flow in this new setup.
        - 2. Edge e* is part of the min cut. reducing its capacity will decrease the value of the cut, hence it will still be part of the min cut. the total value of the new flow will be decrease by one than the total value of the old flow. We need to make sure to reinstate the flow conservation property.
            - Find one path from s to t with positive flow values using edge e* and reduce the flow value along each edge by 1. This can be done in worst case in O(n+m) time. Doing so we could use BFS (same as 2.1 and for positive values) from s to the end of e* and from the right end of e* to t, respectively) when we reduce by one, we take into consideration that there is an integer solution to the max flow problem, thus each non-zero edge can be reduced by at least one.
- Ideas:

    - Let us say e* is the edge of (u, v). G* is the modified network.
    - If max flow f of G satisfies f(u, v) < c(u, v), then f will also be a flow(hence the max flow) in G* where e* have capacity e* -1.
    - If f(u, v) = c(u, v) in G, then max flow of G* might have value less than f.
- Algorithm:
    - If we had f(u,v) < c(u,v) in G, then f is a max flow in G*. return f;
      Otherwise, if we have f(u, v) = c(u, v)
      1, find a simple path A in flow of G from u to s.
      2, find a simple path B in flow of G from t to v.
      3, Take the path B, (v, u), A in flow G, and route 1 unite of flow from t to s along this path. Adding this to f, then we get f* of f-1 in G, with f* (u, v) < c(u, v). f* is a flow in G* as well.
    - Thus f* is a flow in G*.
      4, perform a search in flow of G*, for an augmenting path.
      5, If we find an augmenting path P in flow of G*, return flow f* + path of f.
      6, otherwise return f*.

    - We do BFS in step 1, 2, and 4.
- Complexity:
  The running time is O(V+E)

- Proof:
    - Reducing edges's capacity that has available flow capacity will not affect other path of the network.
    - By the max-flow and min cut theory.