CS 218, Spring 2017 Homework 8
Abhishek Kumar SRIVASTAVA June 7, 2017
Student ID: 861307778

I certify that this submission represents my own original work

---

Solution 1. Part a. Given a flow network $G = (V, E)$, edge $e \in E$ is *upward critical* if increasing the capacity of e increases the value of the maximum flow.

1. No, every network will not have an upward critical edge. For example, consider the below network which has same capacity value for each edges. So even if we increase the capacity of a single edge, other edge will be the bottleneck for the augmenting path between source and sink and will not increase the value of the maximum flow.

$$a \to b \to c$$

2. **Algorithm:** An edge (u, v) is upward critical if there exists a flow augmenting path from s to u, and a flow augmenting path from v to t, which is identifying all possible edges from s to t. For my algorithm identifying all such edge will give the set of upward critical-edges. To identify set of augmenting path we need to find the max-flow for the given network which can be done in $\mathbf{O(n \cdot m)}$ *i.e* $\mathbf{O(n^3)}$ and use the corresponding residual network for finding upward-residual network. We then find the set of all the vertices reachable by an augmenting path from s and call it V_1, V_1 will not include t, or else this would not be a max flow. Similarly we find the set of all the vertices that have an augmenting path to t and call it V_2 which also obey the same properties of V_1. Finding these sets V_1 and V_2 will take **linear time ($\mathbf{O(n + m)}$)** by using BFS/DFS. Now an edge (u, v) is upwards critical if and only if $(u, v) \in E$ and $u \in V_1$ and $v \in V_2$. We can now iterate over all the edges present in the E and check for an edge (u, v) if $u \in V_1$ & $v \in V_2$ and find out all such pairs of (u, v) which takes linear time $\mathbf{O(m)}$ as well.

Correctness: Lets assume that if we can increase the capacity of all the upward-critical edges by δ without affecting the max flow then there should not be any possible path whose flow can be increased but since the capacity of bottlenecked edge (u, v) in G_f is increased the flow can be increased by amount δ.

Complexity: So the worst case complexity of the algorithm will be time required to find the max flow with addition of creating set V_1 & V_2 and macthing all the edges in set V_1 and V_2:

$$\mathbf{T(n) = O(n^3) + O(n + m) + O(m)}$$

$$\boxed{\mathbf{T(n) = O(n^3)}}$$

Part b. Given a flow network $G = (V, E)$, edge $e \in E$ is downward critical if decreasing the capacity of e (by any amount) decreases the value of the maximum flow.

1. No, the set of upward critical edges are not same as the set of downward-critical edges. Every network has a downward-critical edge where it might be in an upward-critical edge set. Any edge to be a downward critical edge it should be a part of min cut edges as decreasing its value decreases the value of maximum flow $i.e$ bottleneck edges.

2. **Algorithm:** Problem of finding all the downward critical edges is similar to as finding all the bottleneck edges which is calculate the max flow in the given network G (which can be done in $\mathbf{O(n^3)}$ time) and using that residual network to looking for edges (u, v) in G for which there no longer exist a path in the residual network between s and t. We can find all such edges by taking every edge $(u, v) \in E$ in G and check if there exists a path between u and v in G_f using **DFS/BFS** which takes $\mathbf{O(n + m)}$ time. All the edges where such path does not exist are all the downward critical edges. The running time of this part will be $\mathbf{O(m(n + m))}$ because we are checking for each edge in G if it exist in the all possible path from s to t of the residual network using DFS/BFS.
Correctness: Assuming that if we can reduce the capacity of such edges by δ without affecting the max flow then we should be able to re-route this δ which is not possible as there does not exist a path between (u,v) in G_f which is contradiction to our finding such edges.
Complexity: The complexity of the algorithm can be given by

$$
\begin{aligned}
T(n) &= O(n^3) + O(m(n+m)) \\
&= O(n^3) + O(m^2) && \text{In worst case } m = O(n^2) \\
&\boxed{\therefore\ \mathbf{T(n) = O(n^4)}}
\end{aligned}
$$

Solution 2. Part 1. This problem is similar to finding the minimum degree of a vertex in a graph because removing that number of edges will disconnect it from the graph.

We can construct a directed graph G' from the given undirected graph G by creating two edges (u, v) and (v, u) for each edge (u, v) in G and assigning capacity to all these edges as 1. $f(u, v)$ be the maximum flow in graph G' from source u to sink v. Select any random vertex u from G' which will work as source and will be fixed and after that compute $f(u, v)\ \forall\ v \neq u$ that is other vertices will be considered sinks. Thus the edge connectivity of the undirected graph will be $c = \min f(u, v)$ among all calculated f, only $n - 1$ pairs are enough where u is any fixed vertex and $v \neq u$. So the problem can be reduced to finding max flow in $n - 1$ networks by considering a different sink each time and the directed graph will be of n vertices *i.e* $O(n)$ and $2m$ edges *i.e* $O(m)$.

Lets assume c' is the edge connectivity of graph G', so removal of c' number of edges will disconnect the graph in two non-empty sets V_1 and V_2 such that $u \in V_1$ and $v \in V_2$. By Max-flow Min-cut theorem, maximum flow must be bounded by the capacity of a minimum cut. Therefore

$$c \leq f(u, v) \leq c'$$

But $c < c'$ contradicts the definition on edge connectivity, as c' is the edge connectivity of graph G'. Therefore $c = c'$ which implies that above algorithm would return the edge connectivity.

Part 2. Consider the algorithm mentioned above, where we select any random vertex u from the network and run Ford-Fulkerson $n - 1$ times for all pair $(u, v)\ \forall\ v \neq u$. Now as all the edges have a weight of 1 therefore the calculation of max flow will take $O(m(n - 1))$ time as the max flow in this case will be upper bounded by n-1 (case where s is connected to all the other edges and t is connected by all the other edges including s as well). As we have to find n-1 such max flows, the time complexity of the algorithm can be given by

$$\begin{aligned} T(n) &= O(m(n-1)(n-1)) \\ &= O(m \cdot n^2) && \text{In worst case } m = O(n^2) \\ &\boxed{\therefore\ \mathbf{T(n) = O(n^4)}} \end{aligned}$$