Assignment 2 500468777

Problem 1

Part A

We are given the residual graph in adjacency list form. We run a depth first search from node s to node t. If such an s-t path exists, it is an augmenting path in the flow. Thus, if an s-t path exists then return false as the current flow is not the maximum, otherwise return true as the current flow is the max flow for the network.

Part B

For flow F, network G and residual network R, the statement "Flow f is a max flow if and only if there are no augmenting paths in R" was proved in page 52 of lecture "Flows and Networks I".

Thus, for the algorithm above, if there is no augmenting path in the residual graph, the current flow is a max flow.

Therefore, if the depth first search does not return a path, then we return true as the current flow is the max flow for the network. Otherwise, we return false as the current flow is not the max flow for the network.

Part C

There are 3 parts to the algorithm:

1. Get the residual network
2. Run a depth first search
3. Return true if step 2 doesn't return a path, otherwise return false

We are given the residual graph for the network, and thus step 1 can be ignored.
The depth first search algorithm takes O(n+m) time where n is the number of vertices in the graph and m is the number of edges.
Step three is a comparison and can be done in constant time.

Thus, the algorithm runs in O(n+m) time.

Problem 2

Part A

There is not necessarily a expandable pipe in every network, consider the following:

Increasing edge (s,A) by 1 does not increase the max flow as it is limited by edge (A,t)
Increasing edge (A,T) by 1 also does not increase the max flow as it is limited by edge (s,A)
Thus no expandable edge exists

Part B

There is always a limiting pipe in any network N.

To prove this, first we find a min cut (a partition into sets A and B) in the network N, the capacity of which is equal to the max flow. As the network is connected, there will always be at least one edge between the sets A and B of the min cut. Decreasing the value of any such edge will decrease the capacity of the min cut by 1 and hence decrease the value of the max flow by 1. Therefore, any edge between sets in a min cut is a limiting edge. Hence, as there will always be at least one edge between the sets of a min cut, there will always be a limiting edge.

Part C

Firstly initialize an empty list called expandable_pipes.

Then, for every edge in the residual graph, if there is not an edge in the other direction, we create one and set the value to 0. This can be done by running a DFS from t-s in the residual graph and creating an edge in the opposite direction if one doesnt exist.

Then run a depth first search on the residual graph which is modified to do the following:

1. Keep track of a boolean value, expanded, which is initially set to false.
2. If an s-t path is found, the search is not terminated, instead it continues until each possible path has been checked.

In the depth first search, at each node c, do the following for each edge e = (c, b) out of c in the network graph:
If the forward edge has flow 0, increase the value to 1 and then continue the DFS at node b. If an s-t path is found with total flow greater than 1, add edge e to expandable_pipes. Then set the flow through e back to 0 and set expanded to false.

Otherwise, if there is a forward edge, continue the DFS as normal.
Once this has completed, return expandable_nodes.

Part D

As we are given a max flow, we know there is no augmenting path through the residual graph. Additionally, at least one edge is at capacity as if none were, there would be an augmenting path and f would not be a max flow. Further, if an edge is not at capacity, it is not expandable as it is not the bottleneck of the residual graph. Finally, if an edge is at capacity, there will be no forward edge in the residual graph, so one is created with value 0.

The algorithm above expands one pipe at a time and checks if doing so creates an augmenting path in the updated residual graph. If such a path exists, then increasing the capacity of edge e will increase the max flow (as an augmenting path exists), hence edge e is an expandable pipe. The boolean expanded ensures that more than one edge cannot be expanded at a time.

Therefore, the algorithm traverses all possible paths in the graph, with each edge considered for expansion. As an edge is only added to expandable edges if expanding it increases the max flow, and each edge is checked, by returning expandable_edges, the algorithm correctly finds all expandable edges in the graph.

Part E

A depth first search on a graph G=(V, E) takes $O(|V| + |E|)$. In an undirected connected graph, if there are n vertices and m edges, $m \geq n-1$. However, for a directed graph, there can be an edge in either direction and thus $m \geq 2(2n-1)$. Thus, $O(m) = O(n)$ and therefore a basic DFS on a residual graph can be done in $O(2m) = O(m)$ time.

However, in the algorithm above, two depth first searches are run. Thus, the algorithm runs in $O(2 * 2m) = O(m)$ time as required.

Part F

First, find the total capacity of the network, T by summing the capacities of each edge in the given array. Then, set P to be the s-t path found by running a breadth first search on the network. Define len(P) = the number of edges in the path, or the number of nodes - 1 as P is linear. Set L = len(p). Then return true if the T / L rounded down ≥ k otherwise false.

Part G

As a BFS searches all the nodes at the present depth before increasing the depth, a BFS partitions the graph into sets representing the depth of each node from the

starting point. As the first s-t path found is returned, it is the shortest s-t path in the graph. Thus, P is the shortest path in the network.

The most efficient use of total capacity T through any path Q is equal to T / len(Q) rounded down, as this maximizes the bottleneck through Q by minimizing unused capacity.

That is:

$$min\{x_i : \Sigma_{i=1}^{n} x_i \leq T \; and \; x_i, T, n \in \mathbb{Z}\}$$

Where each x_i is an edge is maximized when

$$x_1 = x_2 = \ldots = x_n = \lfloor T/n \rfloor$$

As assigning the remaining capacity, r cannot increase the minimum value as r < n. Further, assigning capacity from any x_i to x_j will decrease the minimum, as one of them will have to decrease.

Consequently, as for m > n > 1:

$$\frac{T}{n} > \frac{T}{m}$$

Using P, the shortest path in the network, is the most efficient distribution of available capacity to route the flow of ale from start to finish as the number of edges is minimized. To maximize the bottleneck through P, we assign the capacity of each edge in P to be $\lfloor T/L \rfloor$ (rounded down as the flow is integral and thus each edge must be an integer), as this minimizes unused capacity. Further, rerouting some capacity from P to another path Q to use the remaining capacity will require at least as many nodes as P and thus the decrease in the flow through P will be greater than or equal to the additional flow through Q, and thus, it can only detriment the total flow.

Hence, the bottleneck in the flow, $B = \lfloor T/L \rfloor$, represents the most efficient flow of ale possible.

Thus, there exists a path with max flow greater than or equal to k if and only if $B = \lfloor T/L \rfloor \geq k$.

Part H

For a network with n nodes and m edges:

- Summing the capacities of each edge takes O(m) time
- Running the BFS takes O(n + m) = O(m) time (proof in part d)
- Finding the length of this path takes O(m) time
- The return value is a comparison and takes constant time

Thus the overall time complexity is O(m).