CS 330, Fall 2020, Homework 11
Due Wednesday, November 25, 2020, 11:59 pm Eastern Time

Contributors: None

ANSWER

1.

1.1) Taking only $G(V, E)$, c, s, t and f. We can proceed to make a BFS search through G, using a queue to keep track of which vertices to check next. If we start from s and for each $v \in V$, we will check each edge e that comes out of said vertex v to see if:

- $f'(e) = c(e) - f(e) \leq 0$, and if so we found an edge without any flow available in the maximum flow f, so we won't continue searching in the direction that vertex lead

- If e leads to the sink vertex t, in which case we'd return that f is not the maximum flow

- If none of the above happen, we'd proceed with BFS as per usual

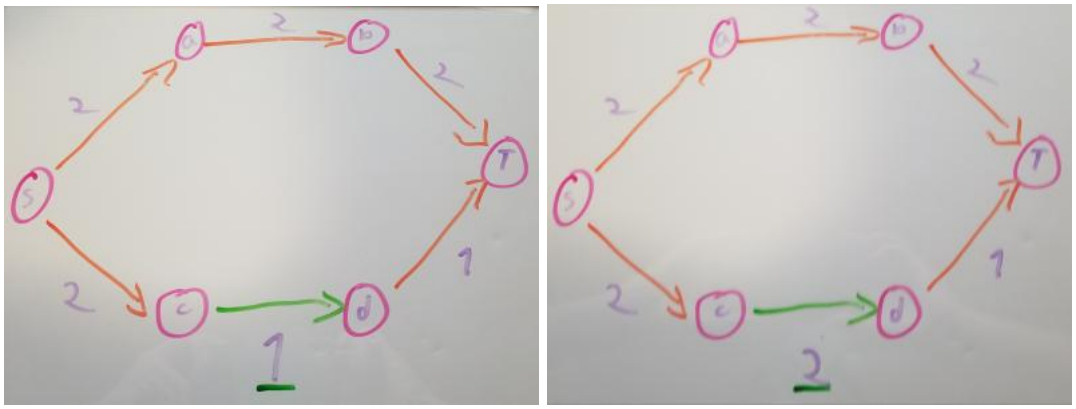
If there are no further vertices to be explored, and we haven't hit one of the two base cases above, then we'd return that f is the max flow. We can keep track of this using a *visited* boolean array, just as you would in a regular BFS.

Since this algorithm will follow the loop structure of the general BFS, we'd end up with $O(m + n)$ approach, with $m = E$ and $n = V$.

The reasoning behind why this algorithm work is actually quite simple, as we went over in lecture, if there are any paths in the Residual Flow Network f' between s and t, then the flow f cannot be the Maximum Flow. Since we have access to the function f that will give you the flow in the Maximum Flow Network for any specific edge e, we can check the residual flow on that edge via $f'(e) = c(e) - f(e)$, and any complete edges in f will have $f'(e) = 0$, while any edges with Residual Flow will have $f'(e) > 0$.

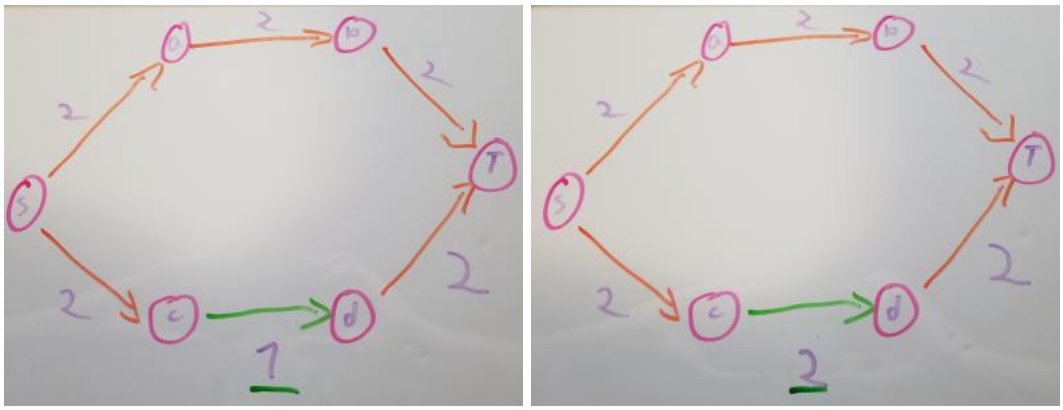
1.2) If we increase the capacity of a single edge $e*$ by $c(e*)_{new} = c(e*)+1$, then the maximum flow $v(f)$ will either remain the same, or be increased by, AT MOST 1.

Below is an example of a Network Flow that does not increase its value, should be noted that the edge that will be increased is the green one, and the value in question is underlined in green:



The first image shows the original Network, with a Maximum Flow of 3, while our green edge $e_g = 1$. The second image however, shows the changed Network, using $e_g = 2$, however, this still does not change the Maximum Flow of 3, since there is still another bottleneck after the edge the increased its value.

Now below is an example of a Network Flow that increases its value by 1, as does the edge:



In this case, we see that there is no bottleneck after the edge that increases its value, which means that, by increasing $e_g = 2$ we remove the existing bottleneck, and increase the Maximum Flow from 3 to 4.

1.3) Taking only $G(E, V)$, f, c, s, t and $e*$ we will start by having our algorithm build the Residual Graph using the already given flow f, as mentioned in lecture, this takes, at most $O(m + n)$ time. Then we can add plus 1 capacity to the edge given to us $c(e*)_{new} = c(e*) + 1$, in linear time.

From here, all we need to do is run BFS along the Residual Graph, trying to find a path that allows for additional flow. Since the Residual Graph already only keeps track of any leftover possible flow, it's easy to go through it and check if there is an entire path from s to t that still allows for further flow by simply checking the residual values of each edge in the graph. Furthermore, we can alter BFS to keep track of the paths it's making, so in case it does find one that allows for further flow, it can return it, otherwise, it will return "Not Found" This will take at most $O(m + n)$.

If we do not find any paths, and the BFS finishes running without finding a path, we can simply have our Algorithm return f, since even with the additional edge, there is no further increase to the possible Maximum Flow. If however BFS returns a path P, then we know, as proven above, that it **must** be a path with Maximum Flow $val(f') = val(f) + 1$, so we can simply run the Augment Algorithm discussed in class along that path, and update f to be $f' = augment(f, c, P)$, as we know from class this will take at most $O(n)$.

With all that said, our entire algorithm will take, at most $O(m + n)$, thus satisfying the requirements.