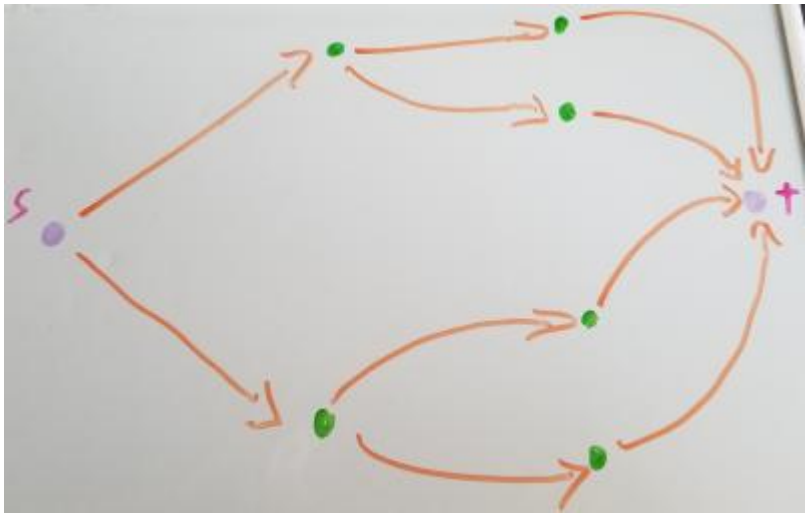
CS 330, Fall 2020, Homework 12
Due Thursday, December 10, 2020, 11:59 pm Eastern Time

Contributors: None

ANSWER

1.

1)



2)

(a) V' will be a set of vertices in G'. This set will have size $|E| + 2$, and each vertex in the set, will correspond to an edge in E. The two additional vertices will be s' and t', and they will be added at each end of the graph.

(b) E' will be the set of edges in G'. This set will have size $k + |V|$ where k equals the amount of edges coming out of s in G plus the amount of edges going into t in G (these edges will be used to connect the new s' and t' to the graph).

(c) s' and t' will be the two new vertices added at the end of each graph, thus not having an equivalent in G but rather being a needed addition in the transformation.

(d) Since our transformation above basically switched vertices in G for edges in G', and edges in G for edges in G', all we have to do now is invert it. To do this, we can check for whatever edge is in p'_x, and put the vertex it corresponds to into p_x, and repeat this for all edges in every path p'. The only downside to this is that we'd need to maintain a data structure that allows us to quickly figure out what edge in G' corresponds to what vertex in G.

3) Since we need to run through every edge every vertex to create G', our runtime for that part would be $O(|V| + |E|)$, the final part of re-assembling would again, in the worst case scenario, have to run through $O(|V| + |E|)$.

   However the algorithm from class runs in the same time as the Ford-Fulkerson Algorithm, or $O(maxFlow * E)$, so in the worst case our algorithm would run in $O(max((|V| + |E|), (maxFlow * E)))$. *(Should be noted that although it's likely that* $O(maxFlow * E)$ *will usually be bigger than* $O(|V| + |E|)$*, it still could be otherwise.)*

2.

1) "Alana shows how, supposing an algorithm A for DeadCode did exist, one can design an algorithm B for Halting that uses A as a subroutine."

2) In order to prove that this problem is an NP problem, we must prove that we can devise some sort of algorithm A, that takes in a input s, and a certificate t, and checks whether or not the input is a valid response, responding *yes* if it is and *no* if it isn't. Furthermore, the algorithm must run in polynomial time.

   The certificate t for this problem would be the list of capacities c, the minimum flow value k, and the two pairs of nodes (s_1, t_1) and (s_2, t_2). Meanwhile the input s would be the pair of flows (f_1, f_2).

   As for the verification algorithm A, it would begin by checking if the values of f_1 and f_2 were greater than or equal to k, if not, the algorithm would immediately return *no*. Otherwise it would check if s_1 and t_1 are in f_1, and if s_2 and t_2 are in f_2; if they aren't then it would immediately return *no*. Finally, the algorithm would go throw each of the edges in list of capacities e, and check whether $f_1(e) + f_2(e) \leq c(e)$, and if the equation doesn't hold for any of the edges, it would immediately return *no*. However, if the input managed to pass all these tests, the algorithm would return *yes*.

   Although it might seem like the algorithm wouldn't be efficient, it would actually be rather quick, the check for the minimum flow and for the presence of the source and end nodes in the flow can be done in $O(|f|)$, while the check for the edge capacities would take at most $O(|E|)$, since $|f| \leq |E|$, then the algorithm A would run in $O(|E|)$, thus running in polynomial time.