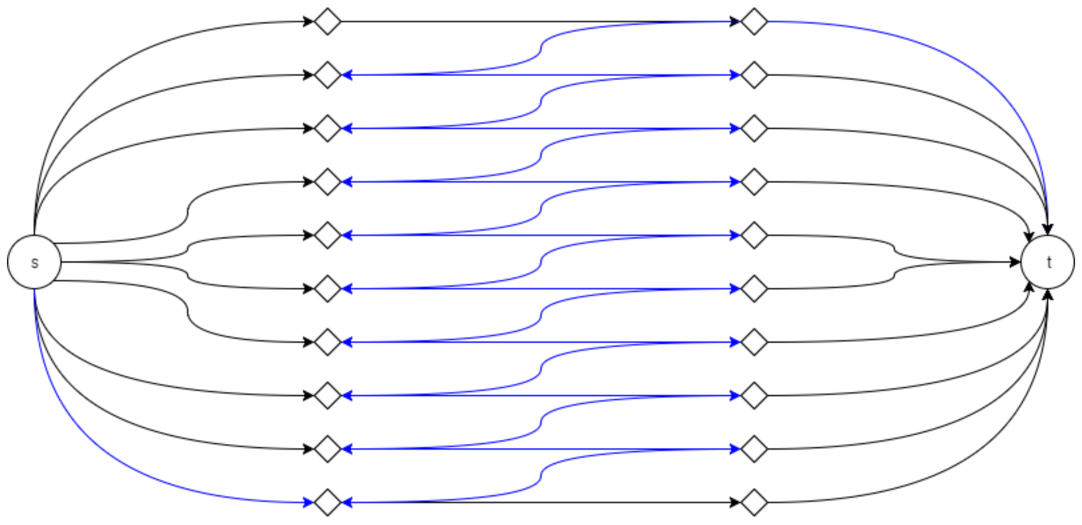
Cody Hubbard - 004843389

CS180 HW 7

7.11
Solution:
FALSE: Proof by counterexample



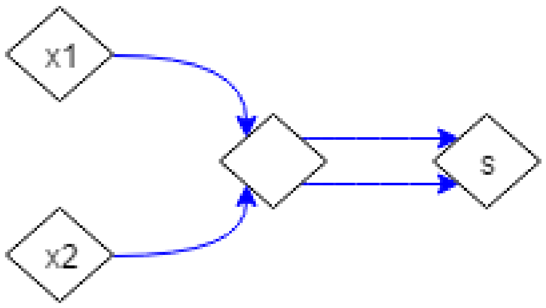
Suppose we have a graph as above with a single sink t, a single source s, and 20 other vertecies connecting them. Also supose that each edge shown in the graph has a maximum flow of only1. This graph will have max flow of 10 in this case. However the algorithm supplied in this problem will find a maximum flow of 1 from s to t along the blue path. Thus the graph found a flow which is $\frac{1}{10}$ the true maximum flow of my graph. This will be true for any such graph with $2 \cdot v$ number of verticies and thus by induction on the number of vertices the algorithm will always find a flow of $\frac{1}{v}$ the maximum flow, which is not an “absolute” constant and thus the claim is false.

7.14

Solution:

(a) To check if a set of evacuation routes exits make a network flow problem out of the given data. Make each node in X be connected to some sort of source node s with directed edges of capacity 1. Make each node in S be connected to some sort of sink node t withdirected edges of capacity equal to the number of elements in X. Lastly make sure all edges which connect X to S are also of capacity 1. Then all you must do is calcualte max flow using something like the Ford–Fulkerson algorithm and if the max-flow is equal to the number of elements in X a route exsists. This should give a result in polynomial time as long as the Ford–Fulkerson algorithm runs in polynomial time.

(b) To check if the evacuation routes exists with this new condition you still need to make a network flow problem out of the data using the same logic as (a). Then instead of calculating max flow repeatedly run the Ford–Fulkerson algorithm (using each node $x \in X$ as a starting node) to find unique paths from each node in X to nodes in S. However each time a path is found remove the nodes used by that path from the graph, as well as any edges which shouldnt logically exist with those nodes gone. If a unique path can be found for each node in X under this condition then a solution exists. This should give a result in polynomial time as long as the Ford–Fulkerson algorithm runs in polynomial time.



In the above graph let $nodes(x1, x2) \in X$ and $node(s) \in S$. The asnwer is clearly yes for (a) as all nodes in X have a unique routes to the only node in S which do not share any edges. However then answer to (b) would be clearly false as they must both go through the middle node, violating the congestion condition.

7.17
Solution:
We are given that there are k paths from s to t which have been cut. We can find these paths by repeatedly running the Ford–Fulkerson algorithm on our original unharmed graph, and then sperate all the found paths. We know there were only k edges destroyed and we know that the destroyed edges formed a minimum cut and thus we can assume that each if the k path is missing at most one 1 edge. We can easily search these paths using a binary search algorithm with our pings. That is, ping the midpoint of our current path and if returns reachable then ping the midpoint node of the $\frac{1}{2}$ upper nodes, if ping returns unreachable ping the midpoint node of the $\frac{1}{2}$ lower nodes; continue this until we find the first node that returns unreachable and the first node which returns reachable, the edge which connects these nodes is the destroyed edge on our current path. Lastly after we have found all the destroyed edges using this method we can store them and then find them in our unharmed original graph using another searching algorithm which works on unsorted data, say breadth-first or depth-first search without the use of ANY pings. All these nodes that come before the cut that becomes before these edges and return it as the set of reachable nodes, and the set of nodes that come after said cut can be returned as the set of unreachable nodes. Since we use a binary search algorithm $O(logn)$, where n is the number of nodes in a path on k paths the complexity of the algorithm is $O(k \cdot logn)$ as required.

7.29

Solution:

To solve this we need to make a graph out of the data. Let all $\{1, ..., n\}$ software application be $\{v_1, ..., v_n\}$ nodes in the graph. For every $(v_i.v_j)$ $i, j < n$ pair of nodes create an edge between them if an x_{ij} expense exists, give the edge this x_{ij} value as its capacity or weight. Lastly make a special node, labeled $PROFIT$, and connect each of the v_i nodes to the $PROFIT$ node with an edge of weight b_i, where b_i is the profit gained by porting over to the new system. Creating a graph like this reduces solving the problem of maximizing $(benefits - expenses)$ into simply finiding the minimum cut of the graph which we know can be done within the polynomial time constraint using something like Karger's algorithm.