Due 23:59 Nov 6 (Sunday). There are 100 points in this assignment.
Submit your answers (**must be typed**) in pdf file to CourSys
`https://coursys.sfu.ca/2022fa-cmpt-705-x1/`. Submissions received after 23:59 of Nov 6 will get penalty of reducing points: 20 and 50 points deductions for submissions received at $[00:00, 00:10]$ and $(00:10, 00:30]$ of Nov 6, respectively; no points will be given to submissions after 00 : 30 of Nov 6.

1. (Chapter 7 Problem 11 of the text book) 20 points

   The Forward-Edge-Only algorithm computes a flow in a flow networks as follows: it searches for $s-t$ paths in a graph $\tilde{G}_f$ consisting only of arcs e for which $f(e) < c_e$, and it terminates when there is no augmenting path consisting entirely of such arcs (the algorithm may choose a forward arc path arbitrarily, provided it terminates only when there are no forward arc paths). A claim for the Forward-Edge-Only algorithm is that there is a constant $k > 1$ (independent of the particular input flow network), so that on every instance of the Maximum Flow problem, the Forward-Edge-Only algorithm is guaranteed to find a flow of value at least $1/k$ times the maximum flow value (regardless of how it chooses its forward edge paths). Decide whether you think this claim is true or false, and give a proof of either the claim or its negation.

   Answer

   For this algorithm which naively assumes that we can find the maximum flow by just using forward flow in a network, which is similar to Ford-Fulkerson algorithm with the only exception being that it does not consider backward edges. This policy kills the optimality of the algorithm as it is the essence of the Ford-Fulkerson algorithm, and removing it would not guarantee the optimality of the flow.

   Now, we can consider that we have a $N \times N + 2$ graph (2 representing the sink and source nodes). Assume that all of the nodes have 4 connections, two connections to the node above it and below it; and the other two connections being the nodes to its right and its left. One thing we can assume, for simplicity, is that each edge has a capacity of 1. If we were to determine the maximum flow of this network (using algorithms like Ford-Fulkerson), we will have a maximum flow of N. I reached this number, N, by just considering left to right augmented paths, and there are a total N of these paths.

   Now our naive algorithm will also be able to achieve this number in its special form, i.e. our traversal takes us left to right and at no point do we go up a node. If we were to consider a corner case we will have a traversal path of connecting node s, sink, to the bottom most node to its right, remember that node s has an out flow of N, $outdeg(s) = N$, call it node $n_{n,1}$. From this node $n_{n,1}$ if the traversal is to go to a node above it, and then to its right, and doing this till it zigzags its way to the sink node, t through the second last node $n_{1,n}$. This would contribute to 1 outflow. Now onto the next iteration, our forward-edge only algorithm will have no other path at all, as the

capacity of each edge is 1. Now the algorithm cannot search/traverse at all because for every row of nodes has a left to right edge already visited. This means that our total outflow is 1 for this algorithm compared to a maximum flow of N. Hence, our assumption is wrong and we cannot achieve a flow of even $1/k$ because the flow itself is 1.

---

2. (Chapter 7 Problem 17 of the text book) 20 points

   There is a communication network of n nodes carrying data from a source node s to a destination node t. There are k link-disjoint paths from s to t in the network when all links functions formally. Now an attacker disabled a minimum number of links that t is not reachable from s. You are sitting at node s and using `ping` command to find the disabled links (command `ping`(v) tells whether node v is reachable or not from s). Assume that you have the topology information of the network and you can decide which node to `ping` based on the previous results of the `ping` commands. Give an algorithm which uses $O(k \log n)$ `ping` commands to find the disabled links.

Answer

Disjoint paths are such paths from source node s to sink node t which do not share a vertex and/or edge. The problem assumes that we have k edge disjoint paths from s to t when all nodes are in working order. The other thing that this problem assumes is that only a minimum number of nodes have been made unreachable form s. We are given a network topology for this problem, which means that we can run the maximum flow algorithm (Ford-Fulkerson algorithm) on this. This algorithm will output the k edge disjoint paths as being the maximum flow paths. Since k edges have been deleted, because sink node t would be reachable otherwise, exactly one edge/link has been attacked by the attacker along each paths, from the Ford-Fulkerson algorithm. For any of these paths we can have a maximum of n nodes and we will have one edge/link disconnected on each path. If we were to do Binary Search on this, it will take $O(logn)$ `ping` commands to reach the unreachable edge on this path. Now as there are k edge disjoint paths, it will take us a total of $O(klogn)$ `ping` commands to discover the disabled links.

---

3. (Chapter 7 Problem 21 of the text book) 20 points

A test instance for a WiFi network consists of n laptops $\{c_1, .., c_n\}$ and n access points $\{p_1, .., p_n\}$. A test set T for the test instance is a set of pairs (c_i, p_j) with the following properties:

(i) If (c_i, p_j) in T then laptop c_i is within the access range of access point p_j,

(ii) Each laptop appears in at least one pair of T, and

(iii) Each access point appears in at least one pair of T.

Answer the following:

(a) Give an example of a test instance for which there is no test set of size n.

(b) Give a polynomial time algorithm which, given a test instance and integer k, decides whether there is a test set of size at most k.

Answer

(a) For this question, if we have a total of n laptops and n access points and no access point is inaccessible, then we must have a test set of size at least n. For example if we have $n = 2$ and c_1 is accessible to p_1, and c_2 is accessible to p_2 then we can have a test set of size up to 4, i.e. $\{(c_1, p_1), (c_2, p_2), (c_1, p_2), (c_2, p_1)\}$. To prove that we will have at least n number of test pairs, we can say that if we have n laptops and they must be used in the network and we have n access points and all of them must be used, then there must be at least n access pairs.

(b) For this part, we have a set T with an integer k. The algorithm to be developed is asking whether the test set T has the size of at most k. For this we first of all create a graph with nodes from both abscissa and ordinate. We then connect, add an edge between, the pair of nodes which appear in the test set in the form (c_i, p_j). We then check the number of edges in the graph G, are at the most k. If the graph does not have a clique of size k or more, we can say that the test set is not possible in G. If we have a clique of at most k this means that the number of elements in the test set must also be k, and this goes the other way round as well. This proves the correctness of the algorithm. This algorithm runs in $O(n)$.

---

4. (Chapter 7 Problem 31 of the text book) 20 points

   There are n boxes $1, .., n$, each box i has size $s(i)$. A box i can be put inside box j if $s(i) < s(j)$. For any two boxes i and i' with $s(i) < s(j)$ and $s(i') < s(j)$, both i and i' can not be put inside j if i is not put inside i' or i' is not put inside i. But for a sequence of boxes $i_1, i_2, .., i_k$ with $s(i_1) < s(i_2) < \cdots < s(i_k)$, i_1 can be put inside i_2, then i_2 put inside i_3,.., and finally i_{k-1} put inside i_k. In this case, all boxes $i_1, .., i_{k-1}$ are inside box i_k and only i_k is visible. The nesting arrangement for a set of n boxes to put one box inside another such that the number of visible boxes is minimized. Give a polynomial time algorithm which, given a set of n boxes $1, .., n$ and $s(1), .., s(n)$, solves the nesting arrangement problem.

Answer

Intuitively, this problem can be handed by sorting the boxes based on their sizes and just looping through the boxes and checking if the size of the next box is larger than the current box, boxes can also have equal sizes. After this we can just put boxes into each other while decreasing the number of boxes visible, which was equal to n in the beginning. If the sizes of two adjacent boxes are equal we just move forward to the next box while checking the same current box. This will give us a total of $O(nlogn)$ running time if we consider we had to sort the array, otherwise it would take $O(n)$ running time. This can be rectified by using a hash map which does not require sorting and will put the smaller boxes in larger boxes in $O(n)$ running time.

The other solution for this would be if we treat it as airplane scheduling problem, which is just like Maximum flow problem. Just as in Airplane scheduling problem, when one plane/crew can take a flight right after their previous flight. This can be thought of as nestling a box inside another one. We can have the source and sink nodes as an arbitrary nodes from which the rest of the graph starts and terminates. Nestling of boxes can be considered as the flow of water from the pipes, meaning the edges represent the box which needs to go inside the other box. This would mean that each box has two nodes u and v, and if we have a sequence $s, u_{i1}, v_{i1}, u_{i2}, v_{i2}, .., u_{in}, v_{in}, t$, this would mean that box $i1$ goes in $i2$. We can assume that each edge has a capacity of 1. This would mean that there can be m distinct paths (augments) which, when solved would result in m visible boxes with each box nested in other bigger box.

---

5. 20 points

Figure 1 gives a flow network G and a function $f : E(G) \to R^+$. The capacity of each arc appears as a label next to the arc, the value assignd to each arc by f is in the box next to the arc.

(a) Is f a flow on G? If yes, why? and give the residual graph G_f w.r.t. f. If no, why?
(b) Implement Ford-Fulkerson Algorithm for the maximum flow problem and show the result of your implementation for G.

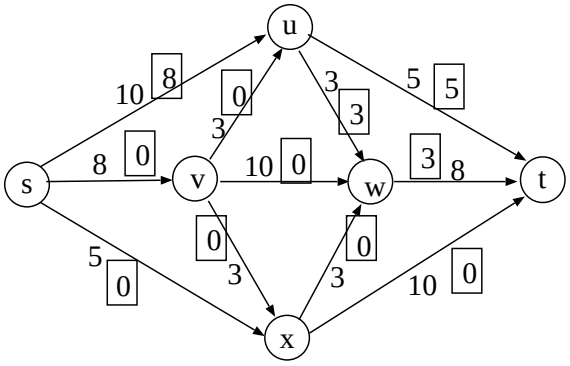

Figure 1: Figure for question 5.

Answer

(a) Yes, it is a flow but not a maximum flow (as to be expected). It is a flow because there are 8 units flowing out of the source node ($outdeg(s) = 8$) and all those 8 units are flowing into the sink node ($indeg(t) = 8$). Furthermore, there is no overflow condition in the graph, at any directed edge, which cements the claim that it is indeed a flow bar the maximum status. The actual graph G_f is as follows:

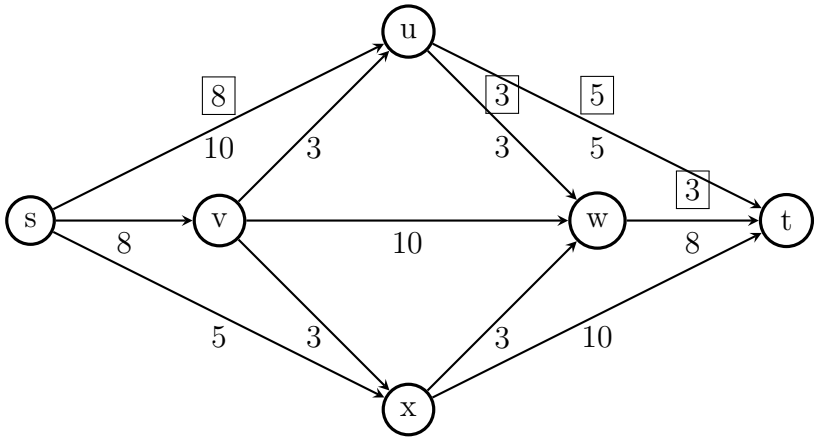

Figure 2: Original Graph without unnecessary flow values

With this its residual graph becomes:

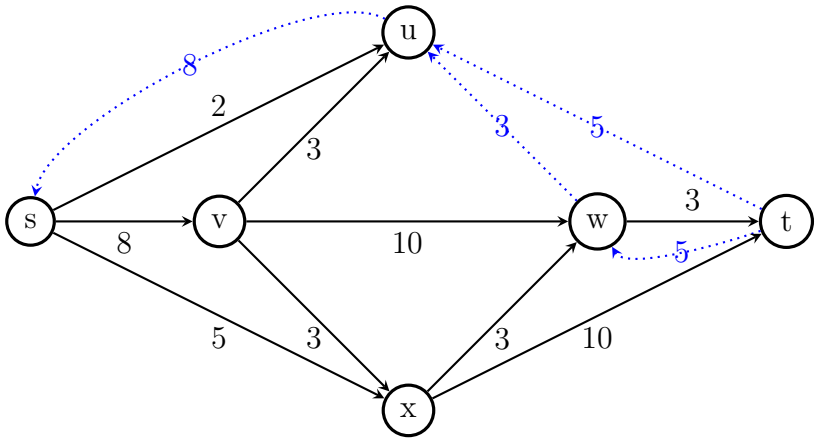

Figure 3: Residual graph of G_f

(b) I assume that the graph for this will be clear of any prior flows. Removing the terms in the squares. That graph will look like this:

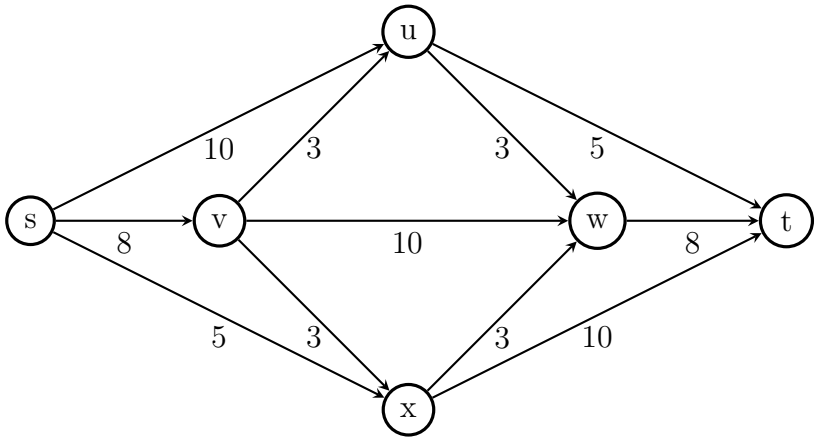

Figure 4: Original Graph without any prior flow values

Now onward, I will just show the flow of the graph without explaining each step after each diagram, as it is self explanatory. The Ford-Fulkerson algorithm is used for determining the maximum flow of the graph in this question. The order of traversal does not matter for this algorithm, as in the original paper of the algorithm there was no specific traversal method and was left open to the developer to figure out the best traversal method for this algorithm. Ford-Fulkerson algorithm just determines the max flow of a graph. In the following figures, the number on top of the edge, in the box, is

the flow of that particular edge and the number on the bottom of the edge is the total capacity of that edge. Each of the figures represent each of the possible traversals, given that the edge does not overflow and/or out flow of any node n being more than its inflow: $outdeg(n) > indeg(n)$.

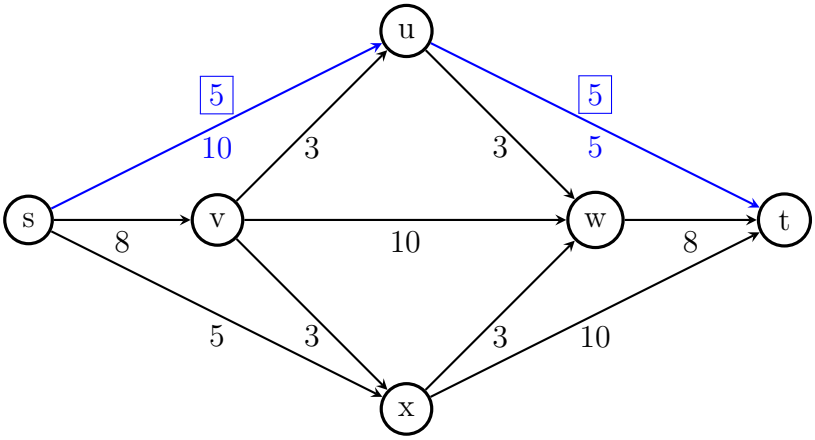

Figure 5: First augmentation

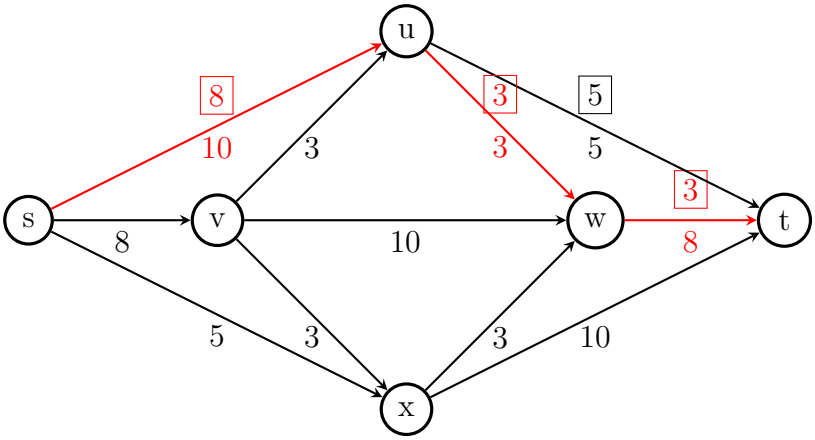

Figure 6: Second augmentation

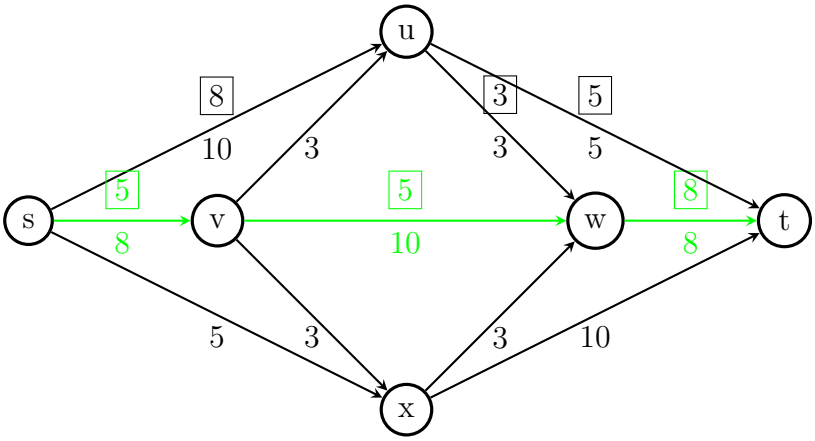

Figure 7: Third augmentation

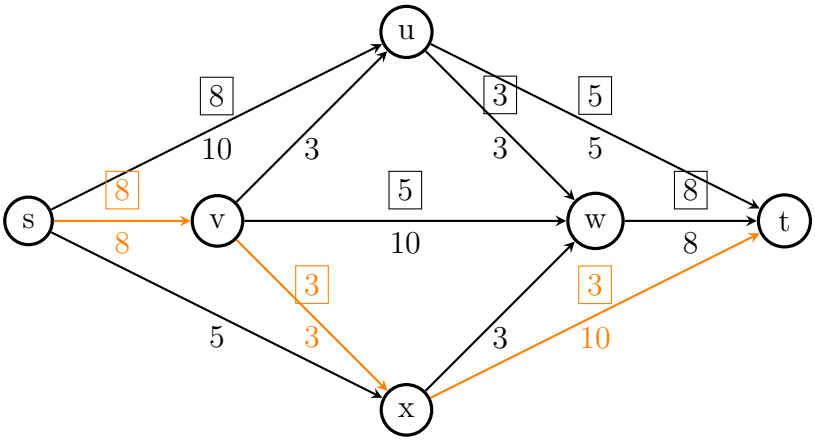

Figure 8: Fourth augmentation

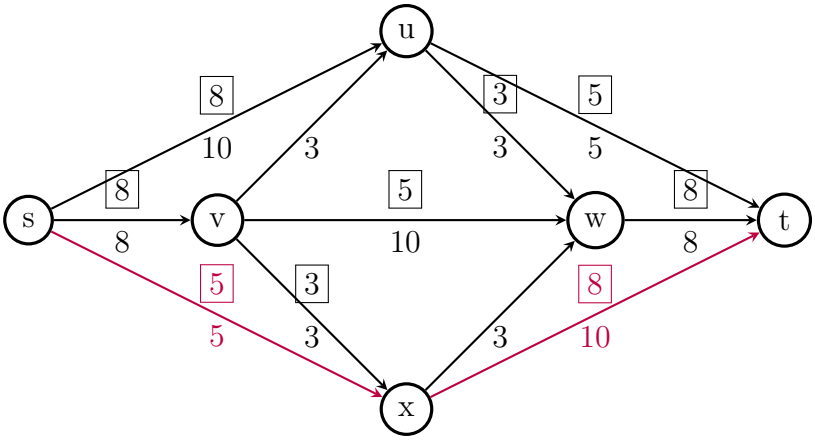

Figure 9: Final Graph