Jeffrey Martinez

CSC 440

Assignment 6: Network Flow

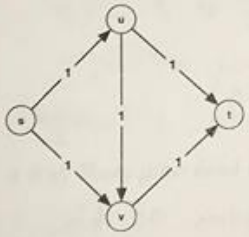
Due Monday, April 16^{th} by 11PM

You may work in small groups on this assignment, but the work you hand in must be your own. You may submit this in one of two ways:

- Write it up electronically (e.g. LaTeX) and upload a PDF to Gradescope.
- Scan your handwritten solution (you can use a scanning app for a smartphone, such as Evernote's free Scannable app or the Notes app in iOS 11) and upload a PDF to Gradescope.
- Solutions handed in on paper **will not be accepted**

Your full name: Jeffrey Martinez

1. (10 points) List all the minimum cuts in the following flow network (the edge capacities are the numbers on each edge)



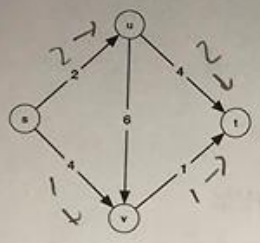
S-U, S-V

V-t, V-t

S-V, Vt

2 cuts each

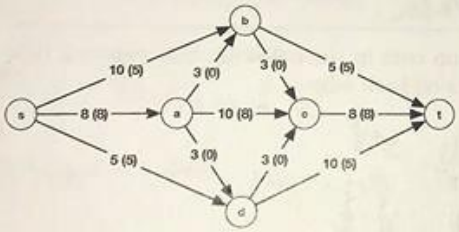
2. (10 points) What is the minimum capacity of an (s, t) cut in the following network?



s → u → t ⇒ 2 capacity.
s → v → t ⇒ 1 capacity

2 + 1 = (3)

3. (20 points) In the following network, a flow has been computed. The first number on each edge represents capacity, and the number in parentheses represents the flow on that edge.



a. (10 points) What is the value of the flow that has been computed? Is it a maximum (s, t) flow?

8+8+5 = 18 ⇒ value of flow

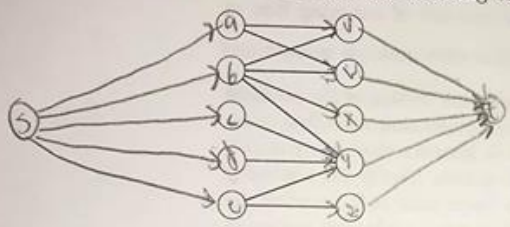
Not a max(s,t) flow because it's not the minimum cut

b. (10 points) Find a minimum (s, t) cut on the network, state which edges are cut, and state its capacity.

s-d, a-d, b-t, c-t = 4 cuts

Capacity of the minimum cut is 2

4. (25 points) At the beginning of the semester, you implemented an algorithm for finding a perfect matching based on a preference list on a complete bipartite graph. Here, consider a different problem: given a bipartite graph that isn't necessarily complete, determine the maximal matching (and if that matching is perfect). Note that here there are no preference lists; there are only vertices and edges. So, we are not worried about *stability*. Consider the following bipartite graph.



a. (15 points) How might you use the Ford-Fulkerson Maximum Flow algorithm to determine whether or not there is a perfect matching? Hints: flow networks need edge capacities, so you'll need to add some. And, flow networks use directed edges, so you'll need to make this a directed graph. You may need some other changes as well.

set node s on left to connect to the left 5 nodes and node t to connect to right 5 nodes then a Ford-Fulkerson algorithm can be used by making every path each have a capacity of 1 and flow towards the right,

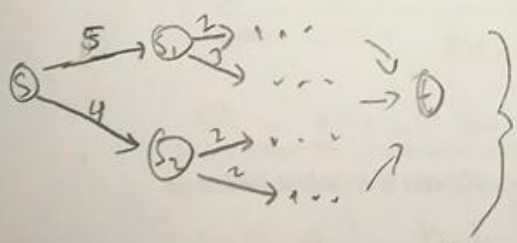
b. (10 points) On the example graph shown, what is the size of a maximum matching? Is it a perfect matching?

The max matching is 4 since there are 2 nodes, c and d, that both connect to one node, both being y, since only one of these two matches (c-y or d-y) can work, then this graph is not a perfect match

5. (25 points) Suppose we generalize the *max-flow* problem to have multiple source vertices $s_1, \ldots s_k \in V$ and sink vertices $t_1, \ldots, t_l \in V$. Assume that no vertex is both a source and a sink, the source vertices have no incoming edges, and sink vertices have no outgoing edges, and that all edge capacities are still integral. A flow is still defined as a nonnegative integer f_e for each edge $e \in E$ such that capacity constraints are obeyed on every edge, and conservation constraints hold at all vertices that are neither sources nor sinks. The value of a flow is the total amount of outgoing flow at the sources $\sum_{i=1}^{k} \sum_{e \in \delta^+(s_i)} f_e$. Prove that *max-flow* with multiple sources and sinks can be reduced to the single-source single-sink version of the problem. Specifically, given an instance of multi-source multi-sink *max-flow*, show how to (i) produce a single-source single-sink instance that lets you (ii) recover a maximum flow of the original multi-source. Sketch out a proof of correctness, and that your algorithms run in linear time (not counting the time required to solve *max-flow* on the single-source single-sink instance). Hint: consider adding additional vertices or edges to the graph.

Set a node S on left, a node t on right, and connect to the sorce vertices and sink vertices respectively. Then for all k, a path $S \to S_k \to t$ takes the total flow of each path S_k takes.

Eg.:



Gets the max-flow for a single source single-sink,

∴ The max flow is produced the same as the original multi source because the path $S \to S_k$ takes the same flow as the path between $S_k \to t$

Space for problem 5

6. (10 points) Describe a real-world problem, not yet discussed in class, that is amenable to *max-flow* or *min-cut*. How would you represent the problem in terms of max-flow or min-cut? Given the various asymptotic complexities of the algorithms for *max-flow* discussed so far, which algorithm might be most appropriate, and why?

A real world problem would be a supply (salt) and demand (distrib
between plows and towns that need the roads to be salted by
the plows. Source s has a path to each garage, $g_1, g_2, g_3 \ldots g_n$
w/ the path to each factory be the amount of salt, S, that
the town, t_n, needs from g_n; for all n where n is a real number.
Each town, $t_1, t_2 \ldots t_n$, is then connected to t_n to provide
a max xflow and min-cut. Ford-Fulkerson is more appropriate
based on the design of a single source, making it run in linear time