CSC 440

Assignment 6: Network Flow

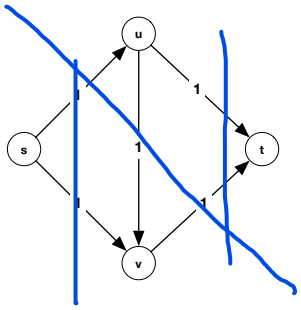
Due Wednesday, April 20^{th} by 11:59PM

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**

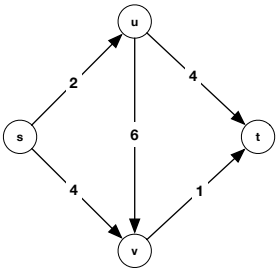
Your full name:	Jason Bachman

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



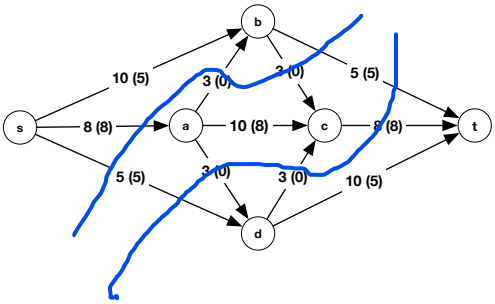
There are 3 min cuts as shown on the diagram.

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



The minimum capacity of an s-t cut in this diagram is 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.



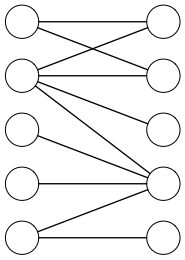
a. (10 points) What is the value of the flow that has been computed? Is it a maximum (s, t) flow?

computed flow is 18. Not a maximum. (a,b) = 8, (b,c) = 3
(a,d) = 3 (d,t) = 8 max flow is 21

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

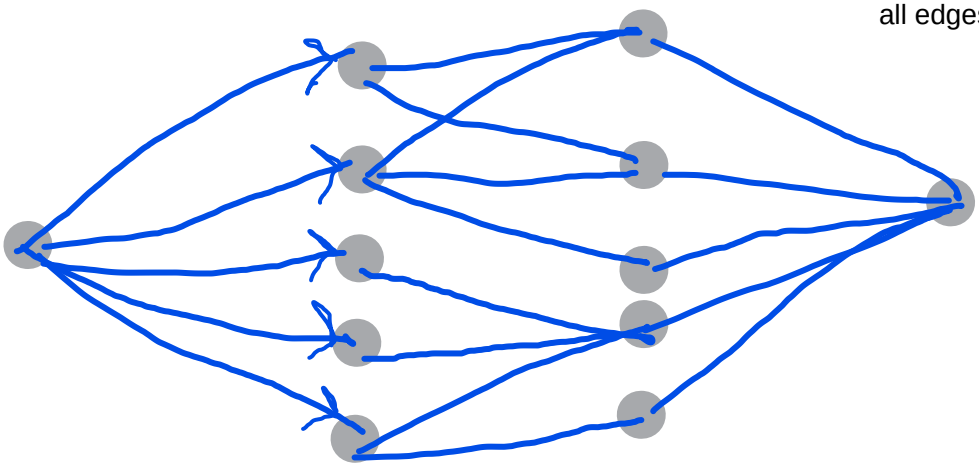
The minimum cut of the figure cuts edges (shown above) the capacity is 21

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.

all edges are capacity of 1

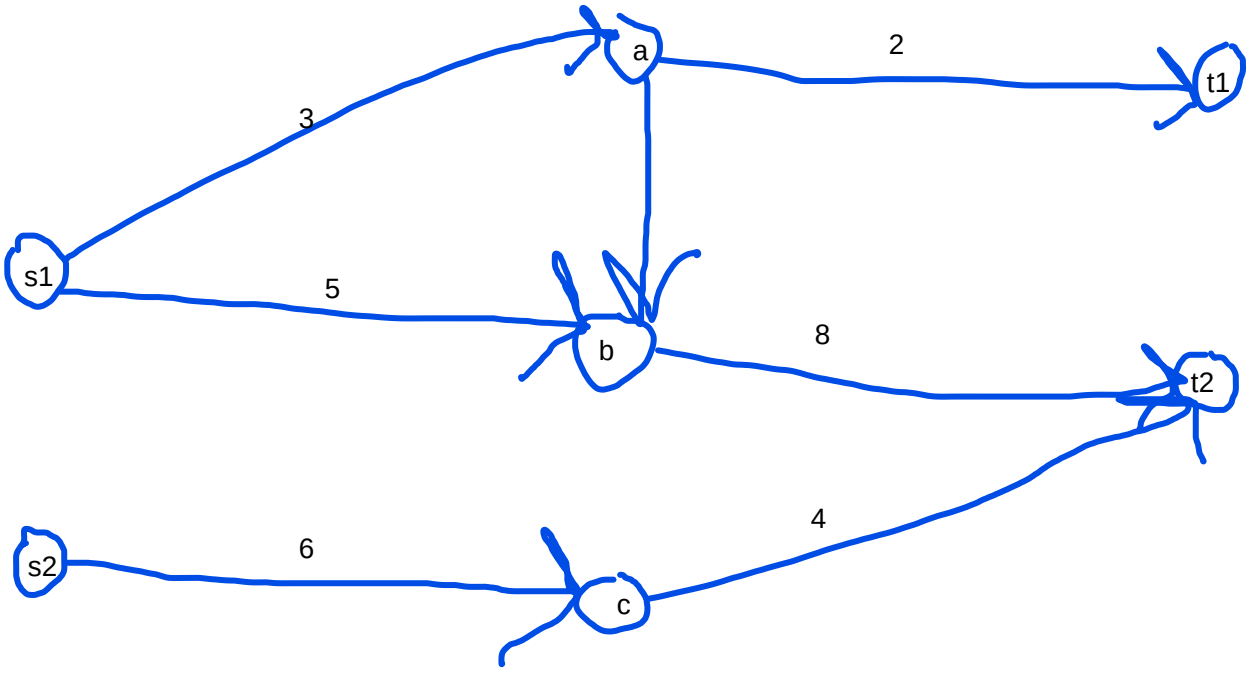


b. (10 points) On the example graph shown, what is the size of a maximum matching? Is it a perfect matching?

max size of a matching is 4. it is not a perfect matching. a perfect matching would have a flow = number of nodes on each side.

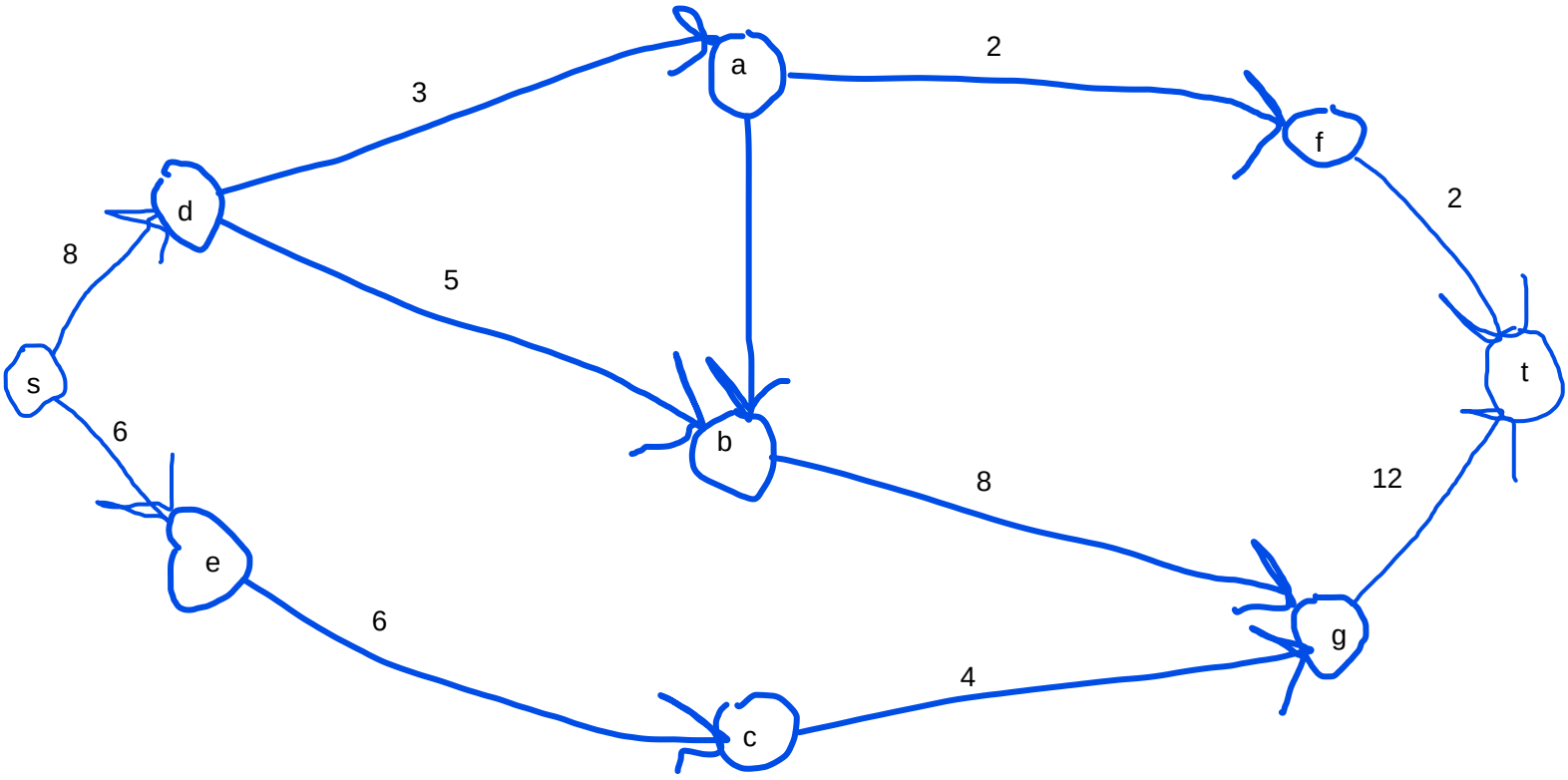
5. (25 points) Suppose we generalize the *max-flow* problem to have multiple source vertices $s_1, ...s_k \in V$ and sink vertices $t_1, ..., t_l \in V$. Assume that no vertex is both a source and a sink, the source vertices have no incoming edges, and sink vertices have no outgoing edges, and that all 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.

create a new source and replace all other sinks with vertices and link them back to the remaining sink with an edge == to their previous output.
do the same with all sources.
linear time



Space for problem 5

s1 becomes d s2 becomes e and t1 becomes f and t2 becomes g



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 amenable to max flow min cut would be planes and airports. To maximize the amount of planes that are able to travel from one large airport to another while making layovers at smaller airports with limited capacities. The source would be large airport 1 and the sink large airport 2. The vertices between them would be the smaller airports all with limited amounts of planes that they can handle (edges). The max flow and min cut would allow you to find out how many planes you can get from one to the last. The algorithm most appropriate for this example would be Dinic's algorithm because it is dependent on the number of vertices^2 and edges for its time taken and the amount of edges and vertices is low. Compared to Ford-Fulkerson's is dependent on flow rate of which the air traffic has an immense amount of. As well as compared to Edmond-Karps algorithm it is better because the amount of vertices is much less than the number of edges as planed can go to only a few airports but all airports can go to all others creating more edges