# CSC 440

## Assignment 6: Network Flow

Due Monday, April 15$^{th}$ 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. This can also be accomplished by annotating the assignment PDF in a tool of your choosing (GoodNotes is great for this).
- Scan your handwritten solution (you can use a scanning app for a smart-phone, such as Evernote's free Scannable app or the Notes app in iOS) and upload a PDF to Gradescope.
  - ***Please*** do not scan a pencil-written version, especially on yellow lined paper. This ends up ***unreadable***

Your full name: Marceline Kelly

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

$\{(s,u),(s,v)\}$

$\{(u,t),(v,t)\}$

$\{(s,u),(v,t)\}$

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

The minimum capacity is 3.

$\{(s, u), (v, t)\}$

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?

The flow in this graph is $5 + 8 + 5 = 18$. However, it is not a maximum flow, which can be fixed with the following changes:

i. Increase flow on $(s, b)$, $5 \to 8$.
ii. Increase flow on $(b, c)$, $0 \to 3$.
iii. Decrease flow on $(a, c)$, $8 \to 5$.
iv. Increase flow on $(a, d)$, $0 \to 3$.
v. Increase flow on $(d, t)$, $5 \to 8$.

This yields a max flow of $5 + 8 + 8 = 21$.

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

$\{(b, t), (b, c), (s, a), (s, d)\}$

Capacity: 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.

- Give the edges a direction, specifically from the left nodes to the right nodes
- Add a source node on the left side of the graph that flows to every node in the left set
- Add a sink node on the right side of the graph that receives flow from every node in the right set
- Give every edge a capacity of 1
- Find the max flow using Ford-Fulkerson
- Remove $s$, $t$, and any edges not used in the max flow. What remains is the maximum bipartite matching

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

In this graph, the maximum matching is of size 4. It is not a perfect matching; there are multiple nodes who share their only possible match, so a perfect matching is not possible.

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.

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?

In recent years, the metro systems in a lot of American cities have been pushed to their limits. New York City's subway, for instance, was built in 1904 when the city's population was approximately 4 million. Today, that number is over 8 million. It might make sense to build a new subway line to handle increasing ridership, but this is a costly and time-consuming endeavor. Urban planners need to be certain of where this new line will go before starting construction.

This situation can be represented as a max-flow problem. The simplest solution (conceptually speaking) would be to find the route that has the highest average ridership, then create a new line adjacent to that route. Each node represents a station, and each edge, a set of tracks. Since the subway system is bidirectional, each directed edge will have a corresponding edge facing the opposite direction. The capacity of a given connection is how many passengers it can handle at the busiest of times, while the flow is how many passengers actually travel across that connection on an average day. The source and sink nodes could be the ends of any two given lines, depending on which ones are being examined.

Because of the bidirectional nature of this system, there are considerably more edges than nodes in this graph. As such, it would make sense to choose an algorithm whose complexity is dependent more on the number of nodes than the number of edges. Ford-Fulkerson [$O(|E|f)$] is probably not a great choice, since many of these subway lines are quite long, which would generate a large max flow capacity. King, Rao, and Tarjan's [$O(|E||V|\log_{\frac{E}{V\log V}}|V|)$] as well as Orlin's algorithm [$O(|V||E|)$ if $|E| \leq O(V^{\frac{15}{16}-\epsilon})$] are both highly optimized and limited moreso by the number of nodes than the number of edges, so I believe they would be appropriate choices for this problem.