CS 473: Algorithms, Spring 2018
HW 7 (due Wednesday, March 28th at 8pm)

This homework contains three problems. **Read the instructions for submitting homework on the course webpage**.

Collaboration Policy: For this home work, each student can work in a group with up to three members. Only one solution for each group needs to be submitted. Follow the submission instructions carefully.

For problems that use maximum flows as a black box, a full-credit solution requires the following.

- A complete description of the relevant flow network, specifying the set of vertices, the set of edges (being careful about direction), the source and target vertices s and t, and the capacity of every edge. (If the flow network is part of the original input, just say that.)
- A description of the algorithm to construct this flow network from the stated input. This could be as simple as "We can construct the flow network in $O(n^3)$ time by brute force."
- A description of the algorithm to extract the answer to the stated problem from the maximum flow. This could be as simple as "Return TRUE if the maximum flow value is at least 42 and False otherwise."
- A proof that your reduction is correct. This proof will almost always have two components. For example, if your algorithm returns a boolean, you should prove that its TRUE answers are correct and that its FALSE answers are correct. If your algorithm returns a number, you should prove that number is neither too large nor too small.
- The running time of the overall algorithm, expressed as a function of the original input parameters, not just the number of vertices and edges in your flow network.
- You may assume that maximum flows can be computed in $O(VE)$ time. Do *not* regurgitate the maximum flow algorithm itself.

Reductions to other flow-based algorithms described in class or in the notes (for example: edge-disjoint paths, maximum bipartite matching, minimum-cost circulation) or to other standard graph problems (for example: reachability, minimum spanning tree, shortest paths) have similar requirements.

---

0. **Not to submit:** Please do Problems 1 and 2 from `https://courses.engr.illinois.edu/cs473/sp2017/homework/hw6.pdf`, and Problem 1 from `https://courses.engr.illinois.edu/CS473/fa2016/Homework/hw6.pdf`.

1. Consider a directed network $G = (V, E)$ with capacity $c(e)$ on edge $e \in E$, and a feasible s-t flow $f : E \to \mathbb{R}^+$.

   We say that flow f is *acyclic* if the subgraph of directed edges with positive flow contains no directed cycles. Show that given any feasible flow f in G, there is an a feasible acyclic flow of the same value (This implies that some maximum flow is acyclic).

2. Let $G = (V, E)$ a directed unit-capacity graph, i.e., $c(e) = 1$ for each $e \in E$.

   - Given an integer $k > 0$, show that G has k edge disjoint paths from s to t if and only if there is an s-t flow of value k in G. (This shows that the maximum number of edge disjoint paths from s to t is exactly the value of the maximum s-t flow.)

     [Hint: Use Q1, and the fact any flow can be converted to an acyclic integer flow.]
   - Describe an algorithm to find maximum number of *vertex-disjoint* paths from s to t [Hint: reduce the problem to finding edge-disjoint paths].

     No proof of correctness necessary but we recommend a brief justifaction. And make sure you have a clear and understandable algorithm.

3. Let $G = (V, E)$ be a flow network with integer edge capacities. We have seen algorithms that compute *a* minimum s-t cut. For both problems below assume that you only have black box access to an algorithm that given G and nodes s, t outputs a maximum (acyclic) flow $f : E \to \mathbb{R}^+$ from s to t.

   - Given G and s, t and an integer k describe an algorithm that checks whether G has a minimum cut with at most k edges.
   - **Not to submit:** Given G and s, t describe an algorithm that decides whether G has at least two distinct minimum s-t cuts. Alternatively, does G has a *unique* minimum s-t cut?

   No proof of correctness necessary but we recommend a brief justifaction. And make sure you have a clear and understandable algorithm.

The remaining problems are for self study. Do *NOT* submit for grading.

- Klenberg-Tardos Chapter 7 is an excellent source on network flow and has many nice problems starting with basic ones to advanced ones. There are several nice problems on reductions to network flow.
- Problem 7.11 in KT asks you an interesting question on whether the simple greedy algorithm that does not use the residual graph can achieve a decent approximation. It cannot but figuring out the counter example is quite useful.
- Problem 7.14, the "evacuation" problem.
- Suppose G is bipartite *regular* graph. That is, all vertices have the same degree r. Prove that G has a perfect matching. Prove that this is not necessarily true in a non-bipartite graph.