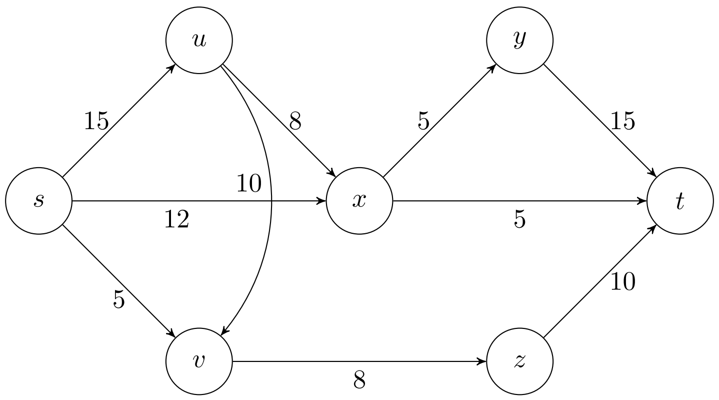
Weekly Assignment 6:
More Flow algorithms

1. Run the Ford-Fulkerson algorithm on the following flow network. For each iteration, pick the shortest augmenting path from s to t (i.e., run the Edmonds-Karp version) and give the residual network. Finally, give the value of the maximum flow and a minimum cut (S, T).



2. Suppose that, in addition to edge capacities, a flow network has *vertex capacities*. That is each vertex v has a limit $l(v)$ on how much flow can pass though v. Show how to transform a flow network $G = (V, E)$ with vertex capacities into an equivalent flow network $G' = (V', E')$ without vertex capacities, such that a maximum flow in G' has the same value as a maximum flow in G. Explain your solution.

3. An edge of a network is called a *bottleneck edge*[1] if increasing its capacity results in an increase in the maximum flow. Give an efficient algorithm to identify all bottleneck edges in a flow network. Argue why your algorithm is correct.

   Hint: Start by running the usual network flow algorithm, and then examine the residual graph.

4. Suppose you and your friend Tijmen live, together with $n-2$ other students, at the popular Hoogeveldt SSHN-complex. Over the next n nights, each of you is supposed to cook dinner for the whole group exactly once, so that someone cooks on each of the nights.

   Of course, everyone has scheduling conflicts with some of the nights (e.g. sports, concerts, etc.), so deciding who should cook on which night becomes a tricky task. For convenience, label the students $\{p_1, \ldots, p_n\}$, the nights $\{d_1, \ldots, d_n\}$, and for student p_i, there's a set of nights $S_i \subseteq \{d_1, \ldots, d_n\}$ when they are *not* able to cook.

   A *feasible dinner schedule* is an assignment of each student to a different night, so that each student cooks on exactly one night, and if p_i cooks on night d_j then $d_j \notin S_i$.

   (a) Describe a bipartite graph G such that G has a perfect matching if and only if there is a feasible dinner schedule for the co-op.

[1]Note that this is distinct from *bottleneck capacity* as defined in the lectures.

(b) Your friend Tijmen takes on the task of trying to construct a feasible dinner schedule. After great effort, he constructs what he claims is a feasible schedule and then heads of to class for the day.

Unfortunately, when you look at the schedule he created, you notice a big problem. $n-2$ students are assigned to different nights on which they are available: no problem there. But for the other two students p_i and p_j, and the other two days, d_k and d_l, you discover that he has accidentally assigned both p_i and p_j to cook on night d_k, and assigned no one to cook on night d_l.

You want to fix Tijmen's mistake but without having to recompute everything from scratch. Show that it is possible, using his "almost correct" schedule, to decide in only $O(n^2)$ time whether there exists a feasible dinner schedule. (If one exists, you should also output it.)

5. You are helping the medical consulting firm *Doctors Without Weekends* set up a system for arranging the work schedules of doctors in a large hospital. For each of the next n days, the hospital has determined the number of doctors they want on hand; thus, on day i, they have a requirement that *exactly* p_i doctors be present.

There are k doctors, and each is asked to provide a list of days on which he or she is willing to work. Thus, doctor j provides a set L_j of days on which he or she is willing to work.

The system produced by the consulting firm should take these lists, and try to return to each doctor j a list L'_j with the following properties.

(A) L'_j is a subset L_j, so that doctor j only works on days he or she finds acceptable.

(B) If we consider the whole set of lists $L'_1, \ldots, L'_k$, it causes exactly p_i doctors to be present on day i, for $i = 1, 2, \ldots, n$.

(a) Describe a polynomial-time algorithm that implements this system. Specifically, give a polynomial-time algorithm that takes the numbers $p_1, p_2, \ldots, p_n$, and the lists $L_1, \ldots, L_k$, and does one of the following two things.

1. Return lists $L'_1, L'_2, \ldots, L'_k$ satisfying properties (A) and (B); or
2. Report (correctly) that there is no set of lists $L'_1, L'_2, \ldots, L'_k$ that satisfies both properties (A) and (B).

Argue why your algorithm is correct, and briefly analyze the running time.

(b) The hospital finds that the doctors tend to submit lists that are much too restrictive, and so it often happens that the system reports (correctly, but unfortunately) that no acceptable set of lists $L'_1, L'_2, \ldots, L'_k$ exists.

Thus, the hospital relaxes the requirements as follows. They add a new parameter $c > 0$, and the system now should try to return to each doctor j a list L'_j with the following properties.

(A*) L'_j contains at most c days that do not appear on the list L_j.

(B) *(Same as before.)* If we consider the whole set of lists $L'_1, \ldots, L'_k$, it causes exactly p_i doctors to be present on day i, for $i = 1, 2, \ldots, n$.

Describe a polynomial-time algorithm that implements this revised system. It should take the numbers $p_1, p_2, \ldots, p_n$, the lists $L_1, \ldots, L_k$, and the parameter $c > 0$, and do one of the following two things.

- Return lists $L'_1, L'_2, \ldots, L'_k$ satisfying properties (A*) and (B); or
- Report (correctly) that there is no set of lists $L'_1, L'_2, \ldots, L'_k$ that satisfies both properties (A*) and (B).

For this question, you need only describe the algorithm; you do not need to explicitly write a proof of correctness or running time analysis. (However, your algorithm must be correct, and run in polynomial time.)