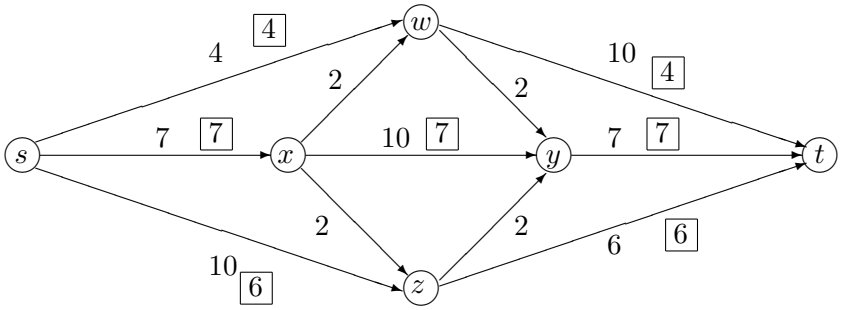
Max Flow, Min Cut, and Matchings (Solution)

1. The figure below shows a flow network on which an s-t flow is shown. The capacity of each edge appears as a label next to the edge, and the numbers in boxes give the amount of flow sent on each edge. (Edges without boxed numbers have no flow being sent on them.)

   (a) What is the value of this flow?

   (b) Is this a maximum s-t flow in this graph? If not, find a maximum s-t flow.

   (c) Find a minimum s-t cut. (Specify which vertices belong to the sets of the cut.)

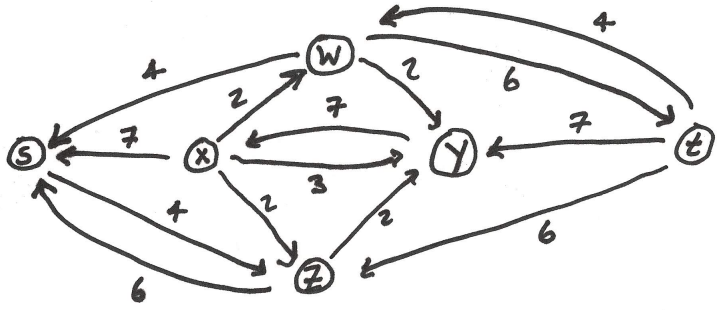


Solution:

(a) The value of this flow is 17. (The total amount of flow passing from s to t.)

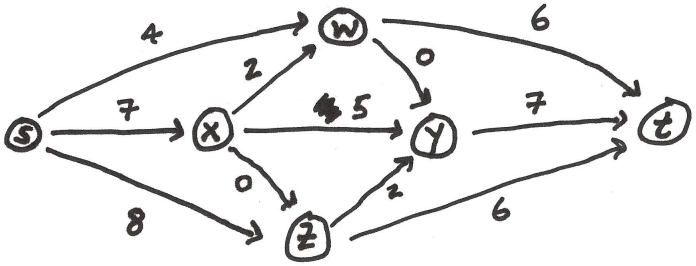
(b) No, this isn't a maximum flow. We can find a maximum flow by looking at the residual network. In this case, from the given flow, we can get a max flow with only a single augmenting path, as shown in the picture below.

First residual network for flow given

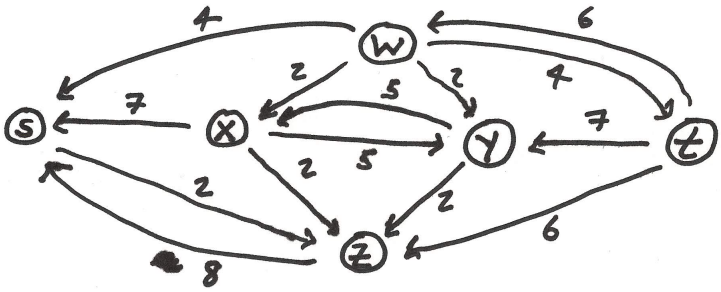


Augment flow along path s, z, y, x, w, t. Capacity of this path is 2, so send 2 units of flow on this path.

New flow values (shown on original graph)



New residual network



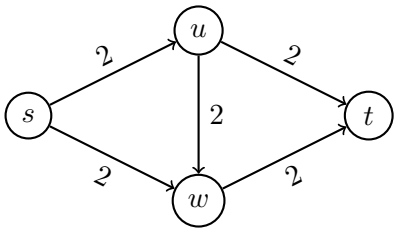
So the maximum flow has value 19.

(c) As explained in class, we can find a minimum cut using the residual network for the final maximum flow. Starting at s, find all vertices that are reachable from s in the residual network. This

forms one set in the minimum cut. The other vertices form the other set part of the cut. So a minimum cut in this case are the two sets $\{s, z\}$ and $\{x, w, y, t\}$.

You can check that the capacity of this cut (i.e. the total capacity of the edges from $\{s, z\}$ to $\{x, w, y, t\}$, consisting of the directed edges $(s, w), (s, x), (z, y)$, and (z, t)) is 19, as the Max Flow/Min Cut Theorem guarantees.

2. Find *all* minimum s-t cuts in the following graph. The capacity of each edge appears as a label next to the edge.



Solution: First, there are exactly four possible cuts in the graph. These are (recalling that the cut must separate s and t):

(a) $\{s\}$, $\{u, w, t\}$,
(b) $\{s, u\}$, $\{w, t\}$,
(c) $\{s, w\}$, $\{u, t\}$, and
(d) $\{s, u, w\}$, $\{t\}$.

It's a simple matter to determine the capacity of each of these cuts, and find those that are minimum cuts. In the order given above, the capacities of the cuts are 4, 6, 4, and 4. So we see that there are three minimum cuts in this graph, namely

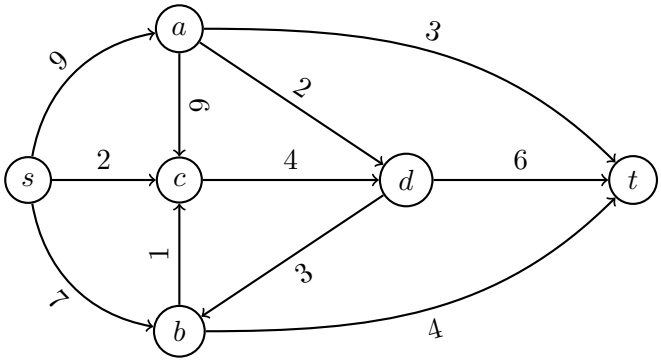
(a) $\{s\}$, $\{u, w, t\}$,
(b) $\{s, w\}$, $\{u, t\}$, and
(c) $\{s, u, w\}$, $\{t\}$.

3. Consider the flow network H below with source s and sink t. The edge capacities are the numbers given near each edge.

(a) Find a maximum flow in this network.

Once you have done this, draw a copy of the original network H and clearly indicate the flow on each edge of H in your maximum flow.

(b) Find a minimum s-t cut in the network, i.e. name the two (non-empty) sets of vertices that define a minimum cut.



Solution:

Starting from 0 flow (i.e. flow of 0 on each edge), we find augmenting paths.

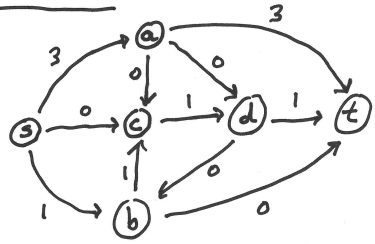
We can find some that are non-overlapping and augment along all simultaneously.

For example, we can take

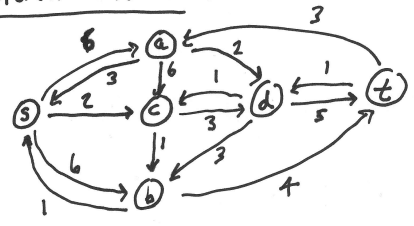
s, a, t augment 3 units

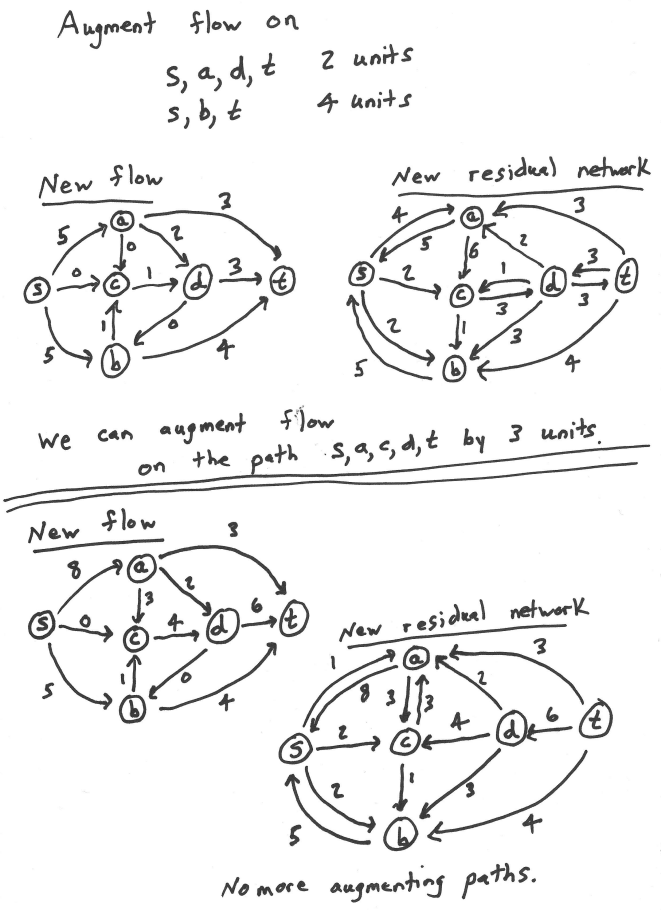
s, b, c, d, t augment 1 unit

New flow (flows on edges)



Residual network (residual capacities on edges)





Thus, the max flow has value 13. Note that the flow shown is not unique, i.e. there are other possible maximum flows in the network, but they will all have the same value of 13.

(b) As in problem (1), start from s and find which vertices can be reached in the final residual network for one set of the cut, and the remaining vertices form the other set. So one minimum cut is $\{s, a, b, c\}$ $\{d, t\}$. The (forwards) edges across this cut are $(a, d), (a, t), (c, d)$, and (b, t). The capacities of those edges are, respectively, $2, 3, 4$, and 4, the sum of which is 13 (which it should be according to the Max Flow/Min Cut Theorem).

Note, also in this case that the minimum cut isn't unique. For example, the cut consisting of the two sets $\{s, a, b, c, d\}$ and $\{t\}$ also has capacity 13.

4. Network flows come up in dealing with natural disasters and other crises, since treatment of injured people and/or evacuation of large numbers of people without overloading hospitals and roads is necessary.

So consider the following scenario: Due to large-scale flooding in a region, paramedics have identified a set of n injured people distributed across the region who need to be rushed to hospitals. There are k hospitals in the region, and each of the n people needs to be brought to a hospital that is within a half-hour's drive to their current location. (So different patients will be able to be served by different hospitals depending upon the patients' locations.)

However, overloading one hospital with too many patients at the same time is undesirable, so we would like to distribute the patients as evenly as possible across all the hospitals. So the paramedics (or a centralised service advising the paramedics) would like to work out whether they can choose a hospital for each of the injured people in such a way that each hospital receives at most $\lceil \frac{n}{k} \rceil$ patients.

Describe a procedure that takes the given information about the patients' locations (hence specifying which hospital each patient could go to) and determines whether a balanced allocation of patients is possible (i.e. each hospital receives at most $\lceil \frac{n}{k} \rceil$ patients).

What is the asymptotic running time of your procedure (in terms of n and k)?

Solution: Seeing this is an exercise appearing with regards to maximum flows, minimum cuts, and matchings, we want to identify some appropriate instance of one of those problems here. Each patient must be sent to a hospital, so on first glance this might come across as a matching problem. However, several patients can be sent to one hospital, so it's not the one-to-one type of pairing that is necessary for a matching instance. But we can solve this problem using a maximum flow algorithm, provided we define the flow network properly. So how do we do that?

Firstly, take the vertices of a (directed) graph to be the set of patients and hospitals. For each patient, we add a directed edge from that patient to each hospital to which she/he can be evacuated. (This depends upon the location of the patient in relation to the hospitals in terms of the driving time to the hospital, but we are given the set of hospitals that each patient can reach, or it's assumed that we can easily figure that out.)

Then, to turn this directed graph into a flow network, we add a source vertex s and connect s to each patient node. And we add a sink vertex t and connect each hospital to t. What are the capacities on the edges of this network? For each edge from s to a patient, take the capacity to be 1, and similarly for each edge from a patient to each hospital she/he is joined to, make the capacity of that edge also equal

to 1. In order to handle the "balance" constraint, for each edge from a hospital to t, take the capacity of that edge to equal $\lceil \frac{n}{k} \rceil$ (this is an integer, recall the definition that the brackets means the "ceiling" of the number inside). (Why do these capacities on all the edges work to do the task we want? If there is a maximum flow of value n, why does this give an evacuation schedule for all n patients?)

This leaves us with a flow network with integer capacities on the edges. So we can use the Edmonds-Karp network flow algorithm to find a maximum flow. If the value of the maximum flow turns out to be n, then we now there is a procedure that allows us to evacuate all the patients *and* maintain the balance amongst the hospitals (once again, why?).

If the value of the maximum flow is less than n, then there is no balanced evacuation procedure (and, I hope, the paramedics try some other procedure to get all the patients to some set of hospitals).

Building the graph takes time $O(nk)$. (Each of the n people can be joined to at most k hospitals, and adding the additional edges to s and t takes time $O(n)$ and $O(k)$ respectively, so, overall the whole flow network takes time $O(nk)$ to construct.) Using the Edmonds-Karp variation of the maximum flow problem, we find the maximum flow in time $O((n+k+2)(nk+n+k)^2)) \in O((n+k)n^2k^2)$. (Recall that the Edmonds-Karp maximum flow algorithm takes time $O(ve^2)$, where v is the number of vertices, and e is the number of edges. In this case we have $v = n+k+2$ and $e \in O(nk+n+k)$.)

5. Suppose you live with $n-1$ other people in an off-campus cooperative apartment. Over the next n nights, each of you is supposed to cook dinner for the entire group exactly once, so that someone different cooks on each night.

   Due to scheduling constraints (concerts, sports, etc), each person is unable to cook on certain nights, so deciding on who is cooking on each night appears to be a tricky task. Suppose we label the people in the flat $\{p_1, p_2, \ldots, p_n\}$ and the nights $\{d_1, d_2, \ldots, d_n\}$.

   Then for each person p_i, there is a set of nights $S_i \subseteq \{d_1, d_2, \ldots, d_n\}$ where p_i is <u>unable</u> to cook.

   A *feasible dinner schedule* is an assignment of each person in the flat to a different night, so that each person cooks on exactly one night, there is someone cooking on each night, and if p_i cooks on night d_j then $d_j \notin S_j$.

   Describe an algorithm to determine if there is a feasible dinner schedule or not.

   What is the running time of your procedure?

Solution: Based on the description of the problem, I hope you concluded that this is an instance of a matching problem. (If so, you're correct.) What would you do? You can first build a bipartite graph. The two sets of vertices are taken to be the set of people, and the set of nights. If a person p_i is able to cook dinner on a night d_j, then you add the edge (p_i, d_j) to the bipartite graph.

Once you build this bipartite graph, then you can use a maximum flow algorithm to find a maximum matching in the way we talked about in class. Direct all edges from "people" to "nights", add a source vertex s, with edges from s to each person, and a sink t with edges from each night to t. Set the capacity of each edge to be equal to one. Then find a maximum flow.

There's a feasible dinner schedule if and only if there is a maximum flow with value n. Otherwise there is no feasible dinner schedule. If there's a maximum flow with value n, then examine the edges in the maximum flow that are used that join people to nights. There will be exactly n of these edges, and due to the way that the graph was constructed (i.e. the fact that all capacities were equal to 1), each person will be joined to a different night, and that gives your feasible dinner schedule.

Constructing the bipartite graph takes time at most $O(n^2)$ (there's at most n^2 edges in the bipartite graph as each of the n people can be joined to at most n nights). Adding the additional edges to make the directed flow network takes time $O(n)$. The Edmonds-Karp maximum flow algorithm takes time at most $O(n^3)$. (Using the crude upper bound of n^2 here on the number of edges in the bipartite graph, it could be better.) So overall, finding a feasible dinner schedule, or concluding there isn't one can be done in time $O(n^3)$.

6. Suppose you are given a flow network with unit capacity edges, i.e. you have a directed graph $G = (V, E)$, a source vertex s and a sink vertex t, and each edge has capacity 1. You are also given an integer parameter k.

   Your goal is to delete a set of k edges from G in order to reduce the maximum s-t flow as much as possible. In other words, you want to find a subset of edges $F \subseteq E$ consisting of *exactly* k edges to delete from G, giving a new flow network G', and the maximum s-t flow in G' is as small as possible.

   Describe an efficient algorithm for performing this task.

   Solution: First find a maximum flow in G. This also allows you to find a minimum cut (A, B) where $s \in A$ and $t \in B$. (As has been outlined before in class, consider the residual network associated

with a maximum flow, then find all vertices reachable from s in that residual network. That is one set in a minimum cut, the remaining set of vertices form the other set in the cut.)

Consider the set of edges E' that cross the cut (A, B), i.e. all of the forwards edges that start in A and end in B. If E' has k or more edges in it, then the set F can consist of any k edges of E'. If E' has fewer than k edges, then for the set F, we can take *all* of the edges of E', together with any other edges of G, so that F has exactly k edges in the overall set.

We claim that the set F described does the job, and you should convince yourself why that is the case. (Hint: What is the capacity of the cut (A, B) in the graph G' obtained from G by deleting the edges in the set F?)

As before, using the Edmonds-Karp maximum flow algorithm, this procedure will work in time $O(nm^2)$ where n is the number of vertices of G and m is the number of edges. (Finding the maximum flow is the time-consuming part here. Then identifying the cut and the edges crossing the cut can be performed in time $O(nm)$ in the residual graph (by BFS, say) and in time $O(m)$, respectively.)