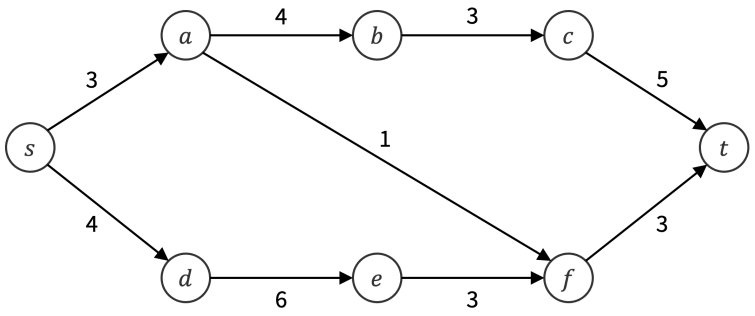
Section 7: Solutions

The goal for this week's section is to use max-flow and min-cut to model various problems. There are a few common tricks that allow us to cleverly solve problems.

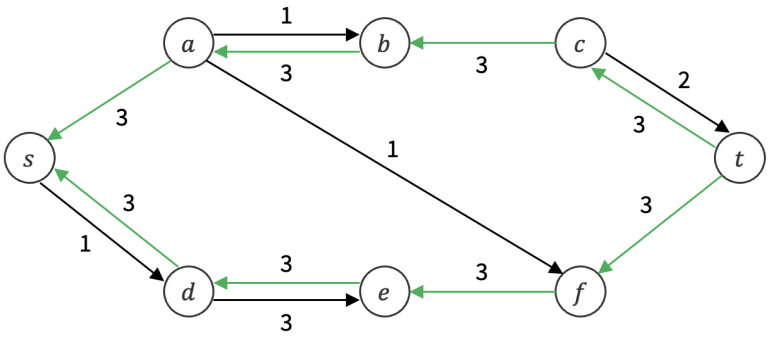
1. Flow algorithms practice

Using Ford–Fulkerson with BFS, find the maximum s-t flow in the graph G below, the corresponding residual graph, and minimum cut.



Solution:

The final residual graph is:



The maximum flow is 6.

The minimum $s-t$ cut is $(\{s,d,e\},\{a,b,c,f,t\})$, which also attains a value of 6.

2. Reservoir balancing

You have a set of overfilled reservoirs $O = \{o_1, \ldots, o_k\}$ and a set of underfilled reservoirs $U = \{u_1, \ldots, u_\ell\}$, and want to move 10,000 gallons of water from reservoirs in O to reservoirs in U. You only care about the total amount of water moved, not each individual reservoir. You have a directed graph $G = (V, E)$ describing the one-way pipes connecting the reservoirs, where $O \subseteq V$ and $U \subseteq V$. This graph may include intermediate reservoirs, whose water levels should not change through your solution. Each pipe $e \in E$ has an integer maximum rate of flow $c(e)$ in gallons per minute. Find a method to move the water in the shortest amount of time.

(a) Write a summary of the problem.

Solution:

Input: Directed graph $G = (V, E)$ with maximum flow rates $c(e)$, sets $O, U \subseteq V$.

Output: Time/method to push 10,000 gallons from O to U while respecting maximum flow rates.

For network flow problems, there are three common tricks that you can apply to solve problems.

- If you want "multiple sources/sinks," add dummy vertices.
- If you want "vertex capacities," split vertices into two.
- If you want "unconstrained capacity," just set capacity to infinity.

(b) Think about the three tricks. Use them to preprocess our input into something suitable for a network flows algorithm.

Solution:

Create dummy vertices s and t, as well as edges (s, o_i) and (u_i, t) for all $o_i \in O$ and $u_i \in U$. Give these new edges infinite capacity, and leave the rest of the graph alone.

(c) After running a max flow algorithm, what do you get? What postprocessing is needed to get the solution?

Solution:

We get the maximum flow r from s to t as well as the flow $f : E' \to R$ that achieves it, where E' is the edge set of our new graph.

Denote $f|_E$ the restriction of f to the original edge set E, and our solution will be to push water at rates according to $f|_E$ for $10,000/r$ minutes.

In a network flow problem, the main claim in your proof will probably be:

"The maximum flow in the graph that I constructed is equal to the maximum solution in the original problem."

It should be intuitive, but to write thing formally, the strategy is to prove two things:

1. We can turn any solution of our network into a solution of the original problem of same quality.
2. We can turn any solution of the original problem into a solution of our network of same quality.

Then, your output works because (1), and no other solution of the original problem is better, by applying (2) and the fact that we found the max flow of our network.

(d) Prove your solution is correct.

Solution:

First, we show that the maximum flow in the graph that we constructed is equal to the maximum simultaneous flow rate from all vertices in O to all vertices in U.

- To turn any solution of our network into a solution of the original problem of same quality: Simply restrict to the original edge set as we did in part (c) (note that this is a valid flow for the original problem). By flow conservation, the flow out of s is equal to the sum of all flow leaving O, so the quality is the same.
- To turn any solution of the original problem into a solution of our network of same quality: Set $f(s, o_i)$ to be the sum of all flow out of o_i (valid because infinite capacity). Then, the flow out of s is the sum of all flow leaving O, thus same quality.

Lastly, it remains to show that the best strategy for filling the reservoirs must have the form "run a fixed strategy with flow rate r for $10,000/r$ minutes" (i.e. no changes over time). Suppose a strategy changed over time. Then, we can improve it by taking the strategy at the point in time where flow rate is fastest, and use that the whole time, so it was not the best.

(e) Which flow algorithm is best for this problem? Then analyze your running time. (Assume $|V| = n$, $|E| = m$, and $n \leq m$.)

Solution:

Because we have no control over how large the capacities are, Ford–Fulkerson with BFS and the Edmonds–Karp bound is best for this problem.

The graph we constructed has $|V'| = n + 2$ vertices and $|E'| = m + |O| + |U| < m + n \leq 2m$ edges. Thus, the running time is $O(|V'||E'|^2) = O(nm^2)$.

3. Traffic modeling

In most cities, traffic congestion happens only at intersections—segments without intersections are free-flowing. That is why you often see more lanes (such as turning lanes) appearing at intersections.

An extremely rough model is that the capacity of an intersection (the total number of vehicles per hour that flow through the intersection in any direction) is proportional to the number of traffic lanes at the intersection. You are given the road network of a city as a graph $G = (V, E)$ (consisting of directed edges, i.e. one-way streets), as well as the number of lanes $c(v)$ at each intersection. Suppose each lane adds a capacity of 300 vehicles per hour, and there are no intersections with more than 12 lanes. Given an origin $s \in V$ and destination $t \in V$ (which also do have limited capacity), compute how many vehicles per hour can move from s to t.

(a) Write a summary of the problem.

Solution:

Input: Directed graph $G = (V, E)$ with vertex capacities $c(v) \leq 12$, and $s, t \in V$
Output: Maximum flow rate from s to t

(b) Think about the three tricks. Use them to preprocess our input into something suitable for a network flows algorithm.

Solution:

First, put infinite capacity on all edges in the original graph.

Then, split every vertex v into v_{in} and v_{out}, and change all (u, v) into (u_{out}, v_{in}), (as well as (v, w) into (v_{out}, w_{in})). Put capacity $c(v)$ on the new edge (v_{in}, v_{out}).

Lastly, we will compute the flow between $s' = s_{in}$ and $t' = t_{out}$.

(c) After running a max flow algorithm, what do you get? What postprocessing is needed to get the solution?

Solution:

We get the maximum total flow rate r, and we should return $300r$.

(d) Prove your solution is correct.

Solution:

To sketch the argument:

- Turn solution of our network into a solution of original problem of same quality:

For every edge (u_{out}, v_{in}), give flow $f(u_{out}, v_{in})$ to original edge (u, v). By flow conservation at v_{in} and $c(v_{in}, v_{out}) = c(v)$, the original vertex v has at most $c(v)$ flow through it, so it is a valid solution. By flow conservation, flow out of s_{in} is the same as flow out of s_{out}, which by construction is the flow out of s, so the quality is the same.

- Turn solution of original problem into a solution of our network of same quality:

  Set $f(u_{out}, v_{in})$ to be the traffic flow from u to v, and set $f(v_{in}, v_{out})$ as necessary to maintain flow conservation. Edges of type (u_{out}, v_{in}) have infinite capacity, so they are fine, and edges of type (v_{in}, v_{out}) have capacity $c(v)$, so they are fine by vertex capacities. The quality is the same, since by construction we set the the flow out of s_{in} to be equal to the flow out of s_{out}, which was the flow out of s.

(e) Which flow algorithm is best for this problem? Then analyze your running time. (Assume $|V| = n$, $|E| = m$, and $n \leq m$.)

Solution:

Because all (non-infinite) capacities are at most 12, Ford–Fulkerson with BFS and the original Ford–Fulkerson analysis is best for this problem.

The graph we constructed has: $|V'| = 2n$ vertices and $|E'| = m + n \leq 2m$ edges. Since $|V'||E'|C = 48nm$, the running time is $O(|V'||E'|C) = O(nm)$.

The following problems will not be covered in section, but may be useful to think about.
We recommend trying them by yourself first. Solutions will be posted in the evening.

4. Where in the grid is Carmen Sandiego?

Carmen Sandiego is currently in an $n \times n$ grid at location s (the location s is some ordered pair). From any square she can move to the four adjacent squares (up, down, left, right). Her goal is to get to any square on any "edge" of the grid, from which she can escape. To prevent her escape, you will place ACME agents on various locations on the grid. Depending on various environmental factors, the locations may require different numbers of agents to block Carmen from progressing. You are given a function $c(u)$ that denotes the number of agents needed to block Carmen at location u. Carmen may be blocked at her starting location. You have k agents to place. Determine whether or not there exists a way to place agents so that you can catch Carmen, and if so, how to place the agents.

(a) Preprocess the input to describe a graph you can run a max-flow algorithm on. Be sure to mention edge directions and capacities.

Solution:

Construct a directed graph $G = (V, E)$ representing the grid, where each grid edge corresponds to two directed edges, one for each direction. Create a dummy sink t. Add an edge from every "edge" vertex to t. Set the capacities of all edges discussed so far to ∞.

Now, for each vertex u of G excluding t, subdivide it into two vertices u_{in} and u_{out}, and redirect all edges entering u to now enter u_{in}, as well as all edges leaving u to now exit u_{out}. Add an edge $(u_{\text{in}}, u_{\text{out}})$ of weight $c(u)$. We find the max flow from $s' = s_{\text{in}}$ to t.

(Note that this graph has anti-parallel edges. Some implementations of Ford–Fulkerson may require no anti-parallel edges, so that residual graph may use "backwards" edges to denote that amount of flow currently being pushed. If this is the case, we can break up one of the directed edges with a new vertex in between, i.e. if we have edges (a, b) and (b, a), delete (a, b) and insert edges (a, c) and (c, b), where c is a new vertex and these new edges have the same capacity as (a, b).)

(b) Postprocess the result of a max flow algorithm to tell whether an assignment is possible or not. If an assignment is possible, how do you read it from the result of the maximum flow algorithm?

Solution:

If the value of the max flow is more than k, we say that there is no way to prevent Carmen's escape. If the max flow is at most k, then we find a min cut (using BFS/DFS on the final residual graph starting from s). Place $c(u)$ ACME agents at all u such that the edge $(u_{\text{in}}, u_{\text{out}})$ is across the cut.

(c) Prove your solution is correct.

Solution:

The proof is a combination of the two proofs above from section, along with the max flow/min cut theorem, which equates the maximum number of agents that can flow from s to t and the minimum number of agents across any cut of the graph separating s and t. (In your homework solutions, you should write the proof out in a bit more detail, as in the previous two problems.)

(d) Which flow algorithm is best for this problem? Then analyze your running time.

Solution:

Since we have no control over capacities $c(u)$ or k, it is reasonable to use Ford–Fulkerson with BFS and Edmonds–Karp bound.

The graph that we construct has $|V'| = n^2 + 1$ vertices and $|E'| = O(n^2)$ edges. Thus, Edmonds–Karp bound gives $O(|V'||E'|^2) = O(n^6)$ time.

However, regardless of the size of capacities $c(u)$, a Ford–Fulkerson-type bound can give $O(n^4 k)$, which is better if k is small. This is because every iteration of Ford–Fulkerson increases the flow by at least 1, and we can stop running the algorithm as soon as the max flow is greater than k (to output no).