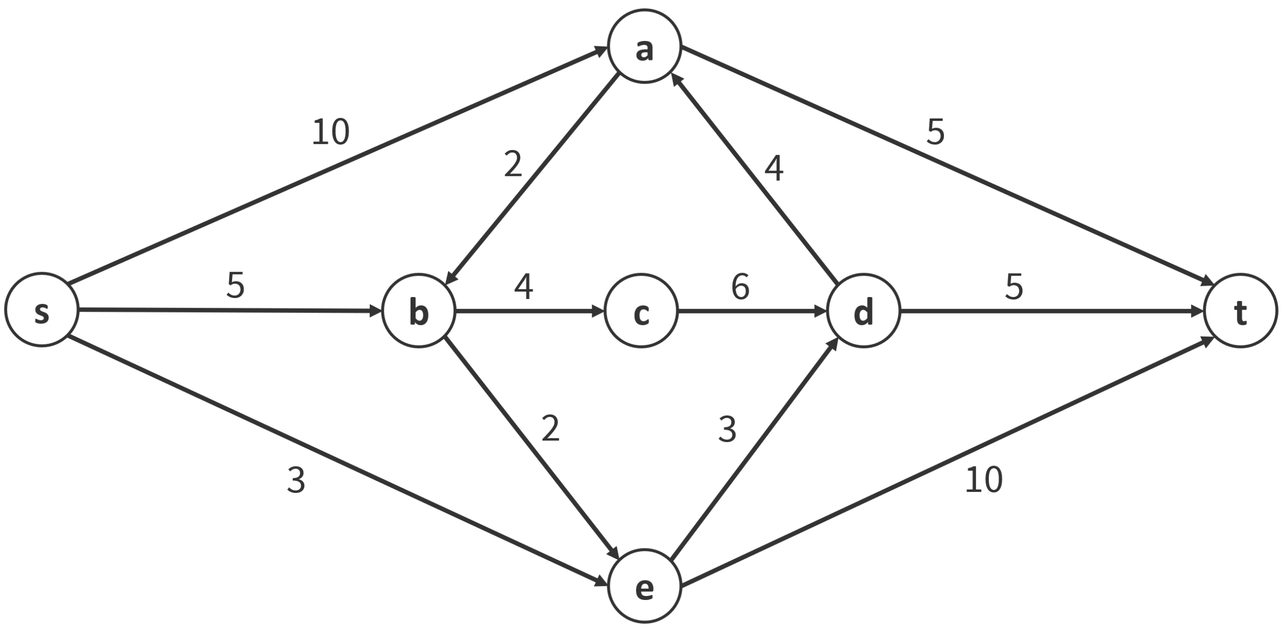
Section 7: Solutions

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 that at first glance look like they are *not quite* the same as the "standard" problems. You might figure out the trick the first time, or you might not. But either way you should remember these tricks! If you've seen it once, it's much easier to use later, and these tricks are common in modeling problems with flows and cuts.

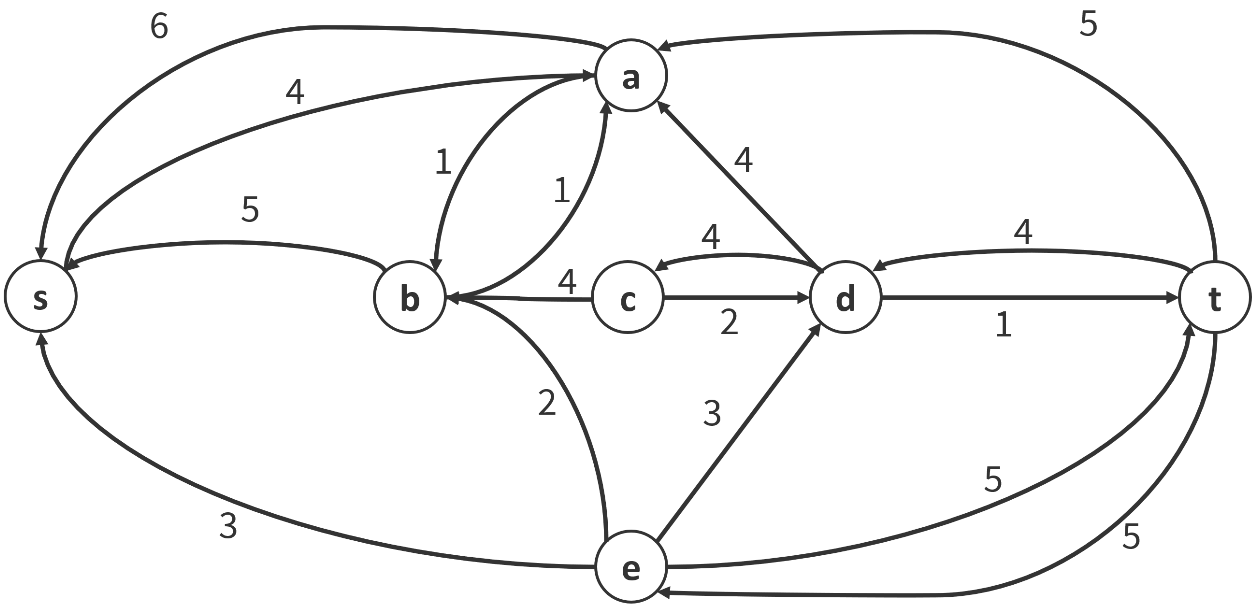
1. Go With the Flow

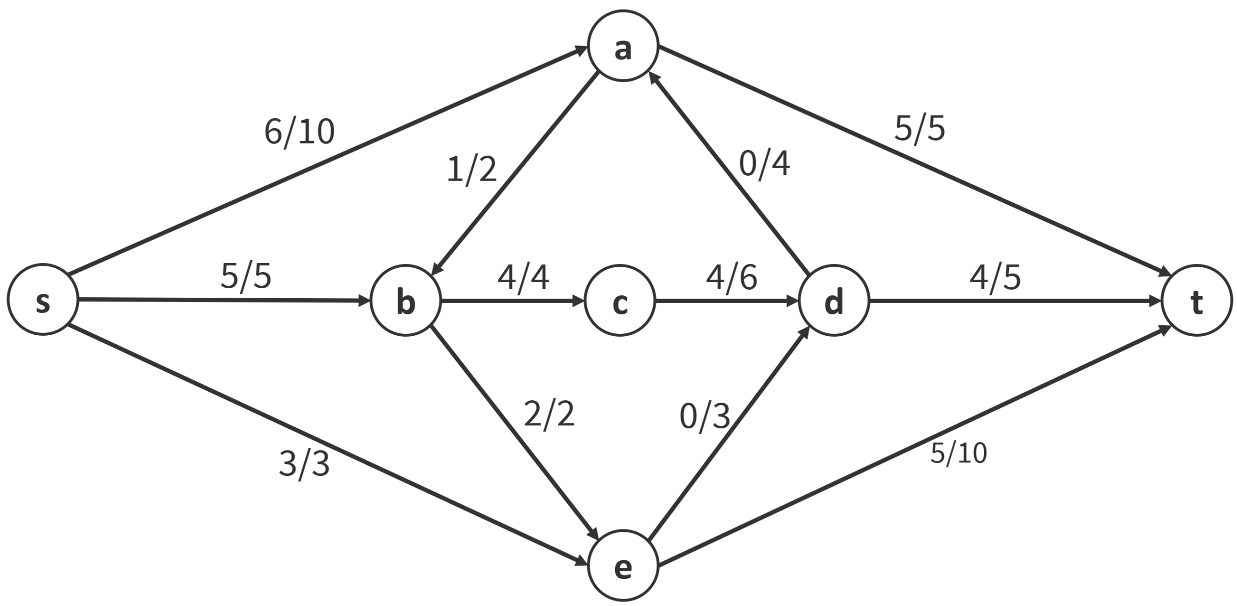
Using Ford-Fulkerson, find the maximum $s-t$ flow in the graph G below, the corresponding residual graph, and list out the corresponding minimum cut.



Solution:

The maximum flow is 14.
The $s-t$ cut is $(\{s,a,b\}$,$\{c,d,e,t\})$.





2. You're not a dummy...

You have three overfilled reservoirs and two underfilled reservoirs. You want to (as quickly as possible) move a total of 10,000 gallons of water from the overfilled reservoirs to the underfilled ones. You do not care how much comes from each of the three individual reservoirs (as long as the total is 10,000 gallons) nor how much arrives at each of the underfilled ones (again, as long as the total is 10,000 gallons). You have a map of (one-way) pipes connecting the reservoirs (in the form of a directed graph); each pipe has a maximum capacity in gallons per minute. You wish to find the way to route the water and the amount of time that will be required.

2.1. Read The Problem Carefully

Answer the usual quick-check questions:

- Are there any technical terms in the problem you don't know? Are there any words that look like normal words, but are actually technical terms?

- What is the input type?
- What is the output type?

Solution:

- Terms:
  One-way pipes: directed edges on our graph
  Maximum capacity: the maximum amount of flow in gallons per minute for each pipe
- Input: a graph with reservoirs as vertices and one-way pipes as edges.
- Output: a flow network indicating the path the water should flow, with the max flow (fastest way to move the water from overfull reservoirs to underfull reservoirs)

2.2. Make a basic model

This sounds like a flow problem. From what you know so far, what would the flow model be? What parts of the problem have you represented successfully? What is still missing? **Solution:**

We use the map as our graph, with the pipes being directed edges and the capacities of the pipes being the edge capacities. But we don't have a single source or sink – we have three possible sources and two possible sinks.

We'll also have to convert our flow, which will be in gallons/minute to a time, but that will just be doing some division. The main problem is the multiple sources and sinks.

2.3. Brainstorm: How can you fix the missing piece?

Find a clever trick to represent the missing piece. The goal here is to do a reduction. By the end of this step, you should have a "standard" flow problem. **Solution:**

Since we don't care **which** reservoirs water is coming from or to, we can treat each of them as "one unit." Add an additional vertex to represent this "combined source" and add an infinite capacity edge from the new source to each of the three source reservoirs. Similarly add an additional vertex for the "combined sink" with infinite capacity edges from the target reservoirs to the new vertex. The combined vertices will be the source and sink for the max-flow.

These combined vertices are called "dummy" vertices. They don't represent anything "real" in the problem, the way the other vertices and edges do. But they let us make this different-looking problem into a standard flow problem. Dummy vertices are a common trick in flow and path-finding problems.

Our algorithm is then just to run any max-flow algorithm on our graph, and then calculate $\frac{10{,}000}{f}$ where f is the value of the maximum-flow.

2.4. Correctness and running time

Explain why your algorithm is correct. For flow problems, the proof is usually just explaining how you've represented each part of the problem, and relying on the correctness of the flow algorithm. **Solution:**

We respect the capacities of the pipes, as they are encoded as edge capacities; each source and sink reservoir will behave as a source/sink via the added edge, and we don't limit the flow because the added edges are infinite capacity. Finally, we will get a maximum-flow by the correctness of the algorithm. We can move 10,000 gallons in $\frac{10000}{f}$ minutes. And we cannot move that much water any faster, as some flow in that time would have greater movement than the maximum flow.

3. Split Personality

You have been given a map of the water cleaning system for the city of Seattle. Water enters from a marked vertex, and flows through pipes (directed edges with specified capacities), through processing facilities, and back out to nature (marked as a specified sink vertex). The processing facilities are vertices in your graph. As the facilities process the water, they **also** have maximum capacities, which may be less than the total capacity entering or leaving the vertex. You wish to find the amount of water that can flow through this network while respecting both the facility and pipe capacities.

3.1. Read The Problem Carefully

Answer the usual quick-check questions:

- Are there any technical terms in the problem you don't know? Are there any words that look like normal words, but are actually technical terms?
- What is the input type?
- What is the output type?

Solution:

- Terms: Maximum capacity of facilities: another capacity in addition to normal edge capacities
- Input: a graph with cleaning facilities as vertices and pipes as edges.
- Output: the maximum amount of water that can flow through the cleaning network

3.2. Make a basic model

This sounds like a flow problem. From what you know so far, what would the most direct flow model be? What parts of the problem have you represented successfully? What is still missing? **Solution:**

Take the map as our graph, with the source, sink, edges and capacities as marked. Each facility will start as a vertex. We have so far failed to represent the flow limits in each processing plant (i.e., the vertices).

3.3. Brainstorm: How can you fix the missing piece?

Find a clever trick to represent the missing piece. The goal here is to do a reduction. By the end of this step, you should have a "standard" flow problem. **Solution:**

We can only put capacities on edges, not vertices...but we could add an edge! We want to make sure there is only so much water flowing through a vertex, so make an edge to represent that. For each vertex v, split it into two vertices v_{in} and v_{out}. Every edge (u, v) is replaced by (u, v_{in}) and every edge (v, w) is replaced by (v_{out}, w). Finally, we add an edge $(v_{\text{in}}, v_{\text{out}})$ with capacity equal to the capacity given in the problem for that vertex.

3.4. Correctness and running time

Explain why your algorithm is correct. For flow problems, the proof is usually just explaining how you've represented each part of the problem, and relying on the correctness of the flow algorithm. **Solution:**

Our algorithm will find a maximum-flow in the altered graph. Note that a flow in our altered graph will be valid; we still encode all edge capacities, we interpret the flow on $(v_{\text{in}}, v_{\text{out}})$ as the flow going through facility v. Since v_{in} has only one outgoing edge and v_{out} has only one incoming edge, accurately represents the flow on that edge represents the total amount flowing through v at any point, and the capacity is enforced.

More problems!

4. Where in the Grid is Carmen Sandiego?

Carmen Sandiego is currently in an $n \times n$ grid. 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. You wish to prevent her escape; you will place ACME agents on various locations on the grid. Each grid location u is labeled with a weight $w(u)$ which denotes the number of agents needed to block Carmen from that location. If that many agents are placed on that location, Carmen cannot pass through. You have k agents to place; determine whether you can catch Carmen (and if so how to place the agents).

(a) Describe a graph you can run a max-flow algorithm on. Be sure to mention edge directions and capacities. **Solution:**

Let $G = (V, E)$ be the original grid graph and v be the location of Carmen. Create a dummy source s and a dummy sink t. Add an edge from s to v, and an edge from every "edge" vertex to t. For each edge $e = (u, v) \in E$, replace it with both directed edges, i.e., (u, v) and (v, u). Set the capacities of all edges discussed so far to ∞.

Now, for each vertex u of G, subdivide it into two vertices u_{in} and u_{out}. All edges entering u now enter u_{in}; all edges leaving u enter u_{out}. Add an edge $(u_{\text{in}}, u_{\text{out}})$ of weight $w(u)$.

(b) Describe how you'll 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:**

Run a max-flow algorithm with s as source and t as sink. Find the value of the maximum flow. If the value of the max flow is more than k, then there is no cut of value k so we cannot prevent Carmen's escape. This is because an allocation of k agents means we can reduce the weight over a finite cut by at most k. So a minimum cut of more than k means there is no possible allocation that reduces the value over the cut to zero. On the other hand, if the max flow is at most k, then we find a min-cut (using the standard process); this cut must use only finite weight edges (as k is finite). Thus the only cut edges must be of the form $(u_{\text{in}}, u_{\text{out}})$ for vertices u. All such vertices u should get $w(u)$ agents assigned to them.

(c) Briefly justify correctness. We aren't expecting a formal proof here, but you should have a sentence or two for each of the restrictions given in the problem. **Solution:**

Any set of vertices that separate v from the border of the grid in the original graph will separate s from t (and vice versa) by construction. Because we subdivided each vertex (and made edges infinite capacity), the only finite cuts in the modified graph correspond to the weights on the vertices of the original graph. Thus any finite cut in the modified graph corresponds to a set of vertices to remove that separate v from the edges in the original graph.

By the correctness of the max-flow algorithm, and the max-flow-min-cut theorem, we find the minimum cut, which is the minimum way to separate Carmen from her escape.

(d) Describe the runtime of the algorithm. Briefly justify why the running time is the value that you state. **Solution:**

$\mathcal{O}\left(n^2 k\right)$. We have an upper bound of $2n^2 + n^2$ edges, corresponding to the grid edges plus the added vertex edges. We can also assume that for each vertex u in our graph, $w(u) \leq k$ or else we just return False. Hence the max flow in this case is at most k so using Ford-Fulkerson we get the runtime above.