Maximum Flow

input: a *flow network* $(G = (V, E), \mathsf{cap}, s, t)$, consisting of
digraph $G = (V, E)$; edge capacities $\mathsf{cap} : E \to \mathbb{R}_+$; *source* $s \in V$ and *sink* $t \in V$
output: a *maximum s-t flow* f in G

Fix any input $(G = (V, E), \mathsf{cap}, s, t)$. What is an s-t flow? Imagine that there is a water source at s. Water must be sent from s, through the edges and vertices of the graph, to sink t. Each edge $(u, w) \in E$ can carry at most $\mathsf{cap}(u, w)$ units of water per second. For each vertex v other than s or t, all water entering v must leave v. The goal is to maximize the rate at which water is sent.

Formally, an s-t flow f in G is a function $f : E \to \mathbb{R}_+$, assigning a flow $f(u, w) \geq 0$ to each edge $(u, w) \in E$, subject to the following constraints. For each edge $(u, w) \in E$, the flow $f(u, w)$ on the edge must be at most the capacity $\mathsf{cap}(u, w)$. (These are called the *capacity* constraints.) For each vertex $v \in V \setminus \{s, t\}$ other than the source or sink, the flow on the edges into v must be equal to the flow on the edges leaving v. (These are called the *conservation* constraints.)

Any $f : E \to \mathbb{R}_+$ meeting these constraints is a *flow*. The *value* of the flow, denoted $|f|$, is the net flow out of the source. (As we shall see, this must equal the net flow into the sink.) The goal is to find a flow f of maximum value.

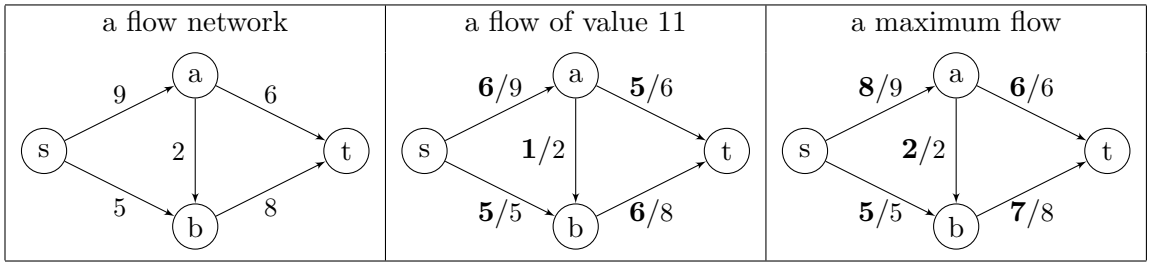
Formally, the problem is the following optimization problem. Let $\mathsf{in}(v) = \{u \in V : (u, v) \in E\}$ denote the in-neighbors of v. Let $\mathsf{out}(v) = \{w \in V : (v, w) \in E\}$ denote the out-neighbors. Then

$$\text{maximize } |f| = \sum_{w \in \mathsf{out}(s)} f(s, w) - \sum_{u \in \mathsf{in}(s)} f(u, s) \quad \text{subject to:}$$

$$\big(\forall (u, w) \in E\big) \quad 0 \leq f(u, w) \leq \mathsf{cap}(u, w) \quad \text{— } \textit{capacity constraints}$$

$$\big(\forall v \in V \setminus \{s, t\}\big) \quad \sum_{w \in \mathsf{out}(v)} f(v, w) = \sum_{u \in \mathsf{in}(v)} f(u, v) \quad \text{— } \textit{conservation constraints}$$

Here is one example of a flow network, a flow in it, and a maximum flow in it:



Flows and cuts. Consider any cut (S, T) (so $S \subseteq V$ and $T = V \setminus S$). Let $E(S, T)$ denote the edges crossing the cut from S to T, so $E(S, T) = \{(u, w) \in E : u \in S, w \in T\} = E \cap (S \times T)$. Define the *capacity* of the cut, $\mathsf{cap}(S, T)$, to be the total capacity of those edges. For any flow f, define the *flow from S to T*, $f(S, T)$, to be the total flow on those edges. That is

$$\mathsf{cap}(S, T) = \sum_{(u,w) \in E(S,T)} \mathsf{cap}(u, w), \quad \text{and} \quad f(S, T) = \sum_{(u,w) \in E(S,T)} f(u, w).$$

E.g. $\mathsf{cap}(\{s\}, \{a, b, t\}) = 9 + 5 = 14$, $\mathsf{cap}(\{b\}, \{s, a, t\}) = 8$, and $\mathsf{cap}(\{s, a, b\}, \{t\}) = 6 + 8 = 14$.

An s-t *cut* is one that separates s and t (that is, $s \in S$ and $t \in T$).

Lemma 1. *Let f and (S,T) be any flow and s-t cut. Then $|f| = f(S,T) - f(T,S)$.*

That is, the value of the flow equals the flow from S to T minus the flow from T to S.

In the example, for the cut $S = \{s, b\}$, the flow f in the third panel has $f(\{s,b\},\{a,t\}) - f(\{a,t\},\{s,b\}) = (8+7) - 2 = 13 = |f|$.

You can find the proof of Lemma 1 at the end of this note. The proof shows the equation by summing the conservation constraints over all vertices $v \in S \setminus \{s\}$.

The next lemma is a direct consequence of Lemma 1.

Lemma 2. *Let f and (S,T) be any flow and s-t cut. Then $|f| \le \mathsf{cap}(S,T)$.*

That is, the value of any s-t flow is at most the capacity of any s-t cut.

Proof. Lemma 1 implies $|f| = f(S,T) - f(T,S) \le f(S,T) \le \mathsf{cap}(S,T)$.

(The two inequalities hold because $f(S,T) \ge 0$, and f satisfies the capacity constraints.) □

In the example above, the s-t cut $S = \{s, a\}$ has capacity 13. So, by the lemma, every s-t flow has value at most 13. The flow in the third panel achieves value 13, so must be a maximum flow.

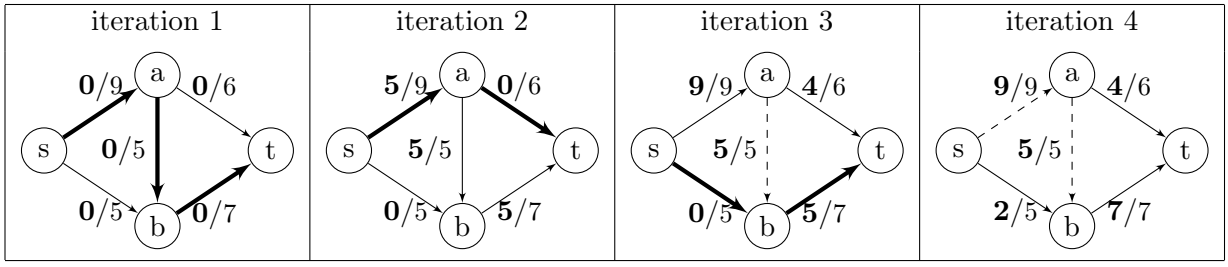
A greedy algorithm that doesn't work. Consider the following greedy algorithm.

```
greedy-flow(G = (V, E), cap, s, t)
1. initialize f[u, w] = 0 for all (u, w) ∈ E
2. for any path p in G, define δ(p) = min{cap(u, w) − f[u, w] : (u, w) ∈ p} (p's residual capacity)
3. while there exists an s-t path p with positive residual capacity δ(p) > 0:
4.1. let p be any such path
4.2. for each edge (u, w) ∈ p: increase f[u, w] by δ(p)        (augment the flow along path p)
5. return f
```

In Step 3, the path p with positive residual capacity is called an *augmenting* path. To find one, delete the *saturated* edges — those with $f[u,w] = \mathsf{cap}(u,w)$. The possible augmenting paths (the s-t paths with positive residual capacity) will be the s-t paths in the resulting graph. Such a path can be found (if it exists) in linear time using depth-first or breadth-first search from s.

The algorithm maintains the invariant that *f is a flow* (that is, f satisfies the capacity and conservation constraints). The invariant holds initially because each edge has flow zero. In each iteration, in Step 4.2, increasing the flow by $\delta(p)$ along the s-t path p maintains the invariant, because it maintains the capacity constraints and the conservation constraints. (Verify!) So, if the algorithm terminates, it returns a flow.

Unfortunately, the algorithm doesn't work. It can get stuck before it finds a maximum flow:



The residual graph G_f. In the remainder of the note, we assume that in G no pair $u, w \in V$ of vertices has edges in both directions ($|\{(u,w),(w,u)\} \cap E| \leq 1$). (This assumption is just to simplify the notation. It is easy to reduce the general case to this one by "splitting" edges as necessary.)

Intuitively, the greedy algorithm failed because it pushed too much flow along edge (a,b). To fix it, we introduce a mechanism for allowing it to "cancel" previously sent flow. Namely, for every edge $(u,w) \in E$, if the current flow f sends positive flow $f(u,w) > 0$ on the edge, we add an artificial *back edge* (w,u), giving it capacity $f(u,w)$. (We also remove all saturated edges.)

The resulting graph is the *residual graph* $G_f = (V, E_f)$ for f, where

$$E_f = \{(u,w) \in E : f(u,w) < \mathsf{cap}(u,w)\} \ \cup \ \{(u,w) : (w,u) \in E,\ f(w,u) > 0\}.$$

The edge capacities, called residual capacities are

$$\mathsf{cap}_f(u,w) = \begin{cases} \mathsf{cap}(u,w) - f(u,w) & \text{if } (u,w) \in E \text{ and } f(u,w) < \mathsf{cap}(u,w) \\ f(w,u) & \text{if } (w,u) \in E \text{ and } f(w,u) > 0. \end{cases}$$

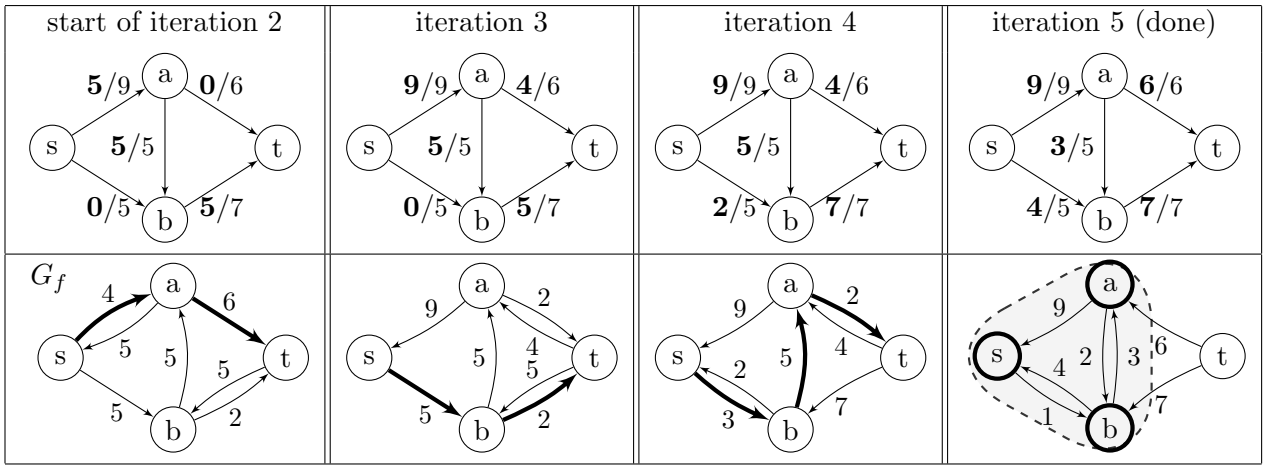
Modify the algorithm to maintain the residual graph G_f, and, in each iteration, to use any s-t path in G_f as the augmenting path. (Note that all residual capacities are positive.)

This gives the Ford-Fulkerson algorithm:

```
Ford-Fulkerson(G = (V, E), cap, s, t)
1. initialize f[u, w] = 0 for all (u, w) ∈ E
2. for any s-t path p in G_f, define δ(p) = min{cap_f(u, w) : (u, w) ∈ p}   (p's residual capacity)
3. while there is an s-t path in G_f:
4.1. let p be any such path                                                 (note δ(p) > 0)
4.2. for each edge (u, w) ∈ p:                                              (augment flow along augmenting path p)
4.3.1. if (u, w) ∈ E: increase f[u, w] by δ(p)
4.3.2. else: decrease f[w, u] by δ(p)                                       (cancel δ(p) units of flow on (w, u))
5. return f
```

Observation 1. *The Ford-Fulkerson algorithm maintains the invariant that* f is a flow in G.

The algorithm maintains the invariant because, in each iteration, augmenting flow along the path p preserves the conservation and capacity constraints. (To see that this still holds when flow is canceled, try some examples.) Here are the first four iterations of the modified algorithm on the previous example. The first row shows the flow f in the graph G at the start of the iteration. The second row shows the residual graph and the augmenting path p in that iteration.



Theorem 1 (Ford & Fulkerson)**.** *If the algorithm terminates, it returns a maximum s-t flow f.*

Proof. Assume the algorithm terminates and returns f. Let S contain the set of vertices reachable from the source s in the residual graph G_f. Let $T = V \setminus S$. The sink t is not in S, because at termination there is no s-t path in the residual graph. So (S, T) is an s-t cut.

We use the following observation, which follows from the definition of S: *in G_f there is no edge (u, w) from S to T (that is, with $u \in S$ and $w \in T$).* (Otherwise, since u is reachable from s in G_f, w would also be reachable from s, so would be in S.)

Consider any edge $(u, w) \in E$ from S to T in the original graph G. This edge is not present in the residual graph G_f (by the observation above). This implies that edge (u, w) must be saturated: $f[u, w] = \mathsf{cap}(u, w)$. Summing over all edges from S to T in E, we get $f(S, T) = \mathsf{cap}(S, T)$.

Consider any edge $(u, w) \in E$ from T to S in the original graph G. The reverse edge (w, u) (S to T) is not present in the residual graph (by the same observation). By definition of G_f, this implies that edge (u, w) has no flow: $f(u, w) = 0$. Summing over all edges from T to S in E, we get $f(T, S) = 0$.

By Lemma 1 and the previous two paragraphs, $|f| = f(S, T) - f(T, S) = f(S, T) = \mathsf{cap}(S, T)$. That is, the value of f equals the capacity of the cut. By Lemma 2, no flow has value more than $\mathsf{cap}(S, T)$. So f is a maximum flow. □

We conclude with two important consequences of the above analysis.

Theorem 2 (max-flow/min-cut)**.** *In any flow network, the maximum s-t flow value equals the minimum capacity of any s-t cut.*

Proof. Let v^* denote the maximum value of any s-t flow. (It is well defined, as the set of flows is closed under limit.) Let f be a flow of value v^*. Let S contain the vertices reachable from s in G_f. Following the reasoning in the previous proof, (S, T) is an s-t cut with capacity equal to $|f|$. By Lemma 2, every cut has capacity at least $|f| = v^*$. So $\mathsf{cap}(S, T) = v^*$ is the minimum possible. □

For many applications, the edge capacities are integers, and we want an *integer-valued* flow, where the flow $f(u, w)$ on each edge (u, w) is an integer.

Theorem 3. *If the edge capacities are integers, the algorithm returns an integer-valued flow.*

Proof. Assume the edge capacities are integers. The algorithm maintains the invariant that f is integer-valued, because if f is integer valued and each capacity is an integer, then each capacity in the residual graph is a positive integer, so $\delta(p)$ is a positive integer in each iteration. And the algorithm terminates, because in each iteration the value of the flow increases by at least 1, but the value never exceeds the capacity of any s-t cut. □

External resources on Max Flow

- CLRS Chapter 26. Dasgupta et al. Chapter 7.2. Kleinberg & Tardos Chapter 7.
- Jeff Edmond's Lectures 23 and 24:
  http://jeffe.cs.illinois.edu/teaching/algorithms/notes/23-maxflow.pdf
  http://jeffe.cs.illinois.edu/teaching/algorithms/notes/24-maxflowapps.pdf

- MIT lecture videos
  - https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-design-and-analysis-of-algorithms-spring-2015/lecture-videos/lecture-13-incremental-improvement-max-flow-min-cut/

Proof of Lemma 1. Fix any flow network $(G = (V, E), \mathsf{cap}, s, t)$. Let f be any s-t flow.

Proof of Lemma 1. We prove by induction on $|S|$ that $|f| = f(S,T) - f(T,S)$ for any s-t cut (S,T).

The base cases is $|S| = 1$. The only s-t-cut (S,T) with $|S| = 1$ is $S = \{s\}$, and for that cut the equation holds by definition of $|f|$. For the inductive step, consider any s-t-cut (S,T) with $|S| \geq 2$. Let $v \in S \setminus \{s\}$ be any vertex other than s in S. Let $S' = S \setminus \{v\}$ and $T' = T \cup \{v\}$.

Assume inductively that $|f| = f(S',T') - f(T',S')$.

To show $|f| = f(S,T) - f(T,S)$, we show $f(S,T) - f(T,S) = f(S',T') - f(T',S')$.

The left-hand side minus the right-hand side of the desired equation is

$$
\begin{aligned}
& f(S,T) \;-\; f(T,S) \;-\; f(S',T') \;+\; f(T',S') \\
&= f(S' \cup \{v\}, T) - f(T, S' \cup \{v\}) - f(S', T \cup \{v\}) + f(T \cup \{v\}, S') && \textit{(as } S = S' \cup \{v\},\ T' = T \cup \{v\}\textit{)} \\
&= [f(S',T) + f(\{v\},T)] - [f(T,S') - f(T,\{v\})] && \textit{(using e.g. } f(S' \cup \{v\},T) \\
&\quad - [f(S',T) + f(S',\{v\})] + [f(T,S') + f(\{v\},S')] && \qquad = f(S',T) + f(\{v\},T)\textit{)} \\
&= [f(\{v\},S') + f(\{v\},T)] \;-\; [f(S',\{v\}) + f(T,\{v\})] && \textit{(cancelling terms and rearranging)} \\
&= \; f(\{v\}, V \setminus \{v\}) \;-\; f(V \setminus \{v\}, \{v\}) && \textit{(using } S' \cup T = V \setminus \{v\} \textit{ twice)} \\
&= \sum_{w \in \mathsf{out}(v)} f(v,w) \;-\; \sum_{u \in \mathsf{in}(v)} f(u,v) \\
&= \; 0 && \textit{(by conservation constraint for } v\textit{)}
\end{aligned}
$$

□