COS 330: Great Ideas in Theoretical Computer Science

Fall 2026

Precept 2

Go With The Flow

RECAP SESSION

Today is a **recap session**: a short lecture on material you have mostly seen before (in COS 226), collected in one place and stated carefully, to prepare you for next week's lectures. We will not assume you remember any of it – every definition we need is stated below. Some of the definitions might be stated in a slightly different way than you have seen before.

The last part of precept is a problem to work on, alone or in small groups. Solutions will be shared after precept.

The Maxflow Problem

A *flow network* is a directed graph $G = (V, E)$ together with a *capacity* $c(e) > 0$ on every edge e, a distinguished *source* vertex s, and a distinguished *sink* (or target) vertex t. We write $V = |V|$ and $E = |E|$ when talking about running times, and we assume for convenience that no edge enters s and no edge leaves t.

Definition 1 (s–t flow)**.** An *s–t flow* is an assignment $f(e) \geq 0$ of a nonnegative amount of flow to each edge, subject to two constraints:

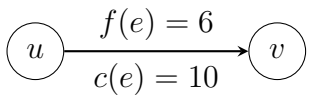
- **Capacity constraint:** $0 \leq f(e) \leq c(e)$ for every edge e.
- **Flow conservation:** for every vertex $v \notin \{s, t\}$, the total flow entering v equals the total flow leaving v.

The *value* of a flow f, written $|f|$, is the total flow into t, or the total flow out of s (these are equal by flow conservation).

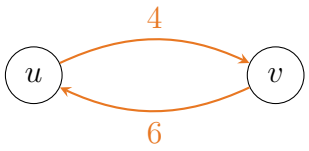
The *maxflow problem* asks for a flow of maximum value. Our goal is to give an efficient algorithm to solve it. The algorithm is the natural greedy idea (push flow along a path from s to t, repeat) plus one crucial fix: we must be allowed to *undo* flow we have already sent. With that in mind, we make the following definition:

Definition 2 (Residual network)**.** Given a flow f in G, the *residual network* G_f has the same vertices, and for each edge $e = (u, v)$ of G:

- a *forward* edge (u, v) with residual capacity $c(e) - f(e)$, present when $f(e) < c(e)$ (the edge is not full); and
- a *backward* edge (v, u) with residual capacity $f(e)$, present when $f(e) > 0$ (the edge is not empty).



in G



in G_f

Definition 3 (Augmenting path)**.** An *augmenting path* is a directed path from s to t in the residual network G_f. Its *bottleneck capacity* is the minimum residual capacity of its edges.

Unwinding the definition: an augmenting path is a path from s to t in the underlying *undirected* graph in which every forward edge is not full, and every backward edge is not empty. Sending Δ units along it – adding Δ on forward edges and subtracting Δ on backward edges – keeps all capacity constraints satisfied (that is what the bottleneck is for) and keeps flow conservation satisfied at every internal vertex (each such vertex has one incoming and one outgoing path edge, and the update changes both by the same amount). It increases the value of the flow by Δ.

Ford–Fulkerson (computes a maxflow)

- Initialize $f(e) = 0$ for every edge e.
- While there exists an augmenting path P in G_f:
  - let Δ be the bottleneck capacity of P;
  - add Δ to $f(e)$ on each forward edge of P, and subtract Δ from $f(e)$ on each backward edge of P.
- Return f.

Finding an augmenting path is just a graph search: run BFS or DFS from s in G_f, which takes $O(V + E)$ time. Two questions remain: does the algorithm return a maxflow when it stops, and how many iterations does it take?

The Mincut Problem

As you might remember, there is a very powerful theorem that says that the maxflow problem is equivalent to another problem known as the *mincut problem*. To prove the correctness of Ford–Fulkerson, it is helpful to first define the mincut problem and prove the equivalence.

Definition 1 (s–t cut)**.** An *s–t cut* is a partition of V into two disjoint sets A and B with $s \in A$ and $t \in B$. The *capacity* of the cut (A, B) is the sum of the capacities of the edges directed from A to B:

$$\text{cap}(A, B) = \sum_{\substack{e=(u,v)\in E \\ u\in A,\, v\in B}} c(e).$$

Note carefully that edges directed from B to A do **not** count towards the capacity of the cut. The *mincut problem* asks for a cut of minimum capacity.

Lemma 2 (Flow-value lemma)**.** Let f be a flow and (A, B) any cut. The *net flow across* (A, B), defined as the total flow on edges directed from A to B minus the total flow on edges directed from B to A, equals $|f|$.

Proof. Applying the definition, we can write the net flow across (A, B) as:

$$\sum_{\substack{e=(u,v)\in E \\ u\in A,\, v\in B}} f(e) \; - \sum_{\substack{e=(v,u)\in E \\ v\in B,\, u\in A}} f(e).$$

Now observe that the above is equal to:

$$\sum_{u\in A} \left(\sum_{\substack{e=(u,v)\in E \\ v\in V}} f(e) - \sum_{\substack{e=(v,u)\in E \\ v\in V}} f(e) \right).$$

The second expression is equal to the first one, but its inner sums extend over all edges incident to u, instead of only those with their other endpoint in B. To see why this is true, note that an edge with both endpoints in A is counted in both inner sums – positively at its tail and negatively at its head – and so it cancels out.

But now notice that by flow conservation the inner sum is 0 for every $u \in A \setminus \{s\}$, and so the above expression reduces to just the term for $u = s$, which is exactly the value of the flow $|f|$. □

Using the flow-value lemma, we can prove the following important fact:

Lemma 3 (Maxflow $\leq$ mincut)**.** For any flow f and any cut (A, B), we have $|f| \leq \text{cap}(A, B)$. In particular, the value of a maximum flow is at most the capacity of a minimum cut.

Proof. By the flow-value lemma, $|f|$ is the net flow across (A, B). Dropping the (≥ 0) flow on edges from B to A only increases the quantity, so $|f|$ is at most the total flow on edges from A to B. By the capacity constraints, that total is at most $\text{cap}(A, B)$. □

This lemma already gives us something very useful: **any** cut is a certificate of an upper bound on **every** flow. So if you ever exhibit a flow f and a cut (A, B) with $|f| = \text{cap}(A, B)$, you have proved at once that f is a maxflow and that (A, B) is a mincut – no further argument needed. Keep this in mind: it is the single most useful trick in this topic, and it is exactly what the correctness proof below produces.

Correctness of Ford–Fulkerson

Why should the flow we end up with be the best possible one? The key is that when Ford–Fulkerson gets stuck, the set of vertices it can still reach in the residual network *is* a cut, and it is a cut of exactly the right capacity.

Theorem 1 (Correctness of Ford–Fulkerson)**.** If Ford–Fulkerson terminates, the flow it returns is a maximum flow.

Proof. Let f be the flow the algorithm returns. It is a valid flow: we start from the all-zero flow, and, as we saw above, each augmentation preserves both the capacity constraints and flow conservation.

The algorithm stops only when there is no augmenting path with respect to f, that is, when t is not reachable from s in G_f. So let

$$A = \{v \in V : v \text{ is reachable from } s \text{ in } G_f\}, \qquad B = V \setminus A.$$

Then $s \in A$, and $t \in B$ precisely because there is no augmenting path, so (A, B) is a cut.

We claim that f saturates this cut. Consider any edge $e = (u, v)$ of G with $u \in A$ and $v \in B$. If e were not full, G_f would contain the forward edge (u, v), and v would be reachable from s – so every such edge is full: $f(e) = c(e)$. Similarly, consider any edge $e = (v, u)$ with $v \in B$ and $u \in A$. If e carried positive flow, G_f would contain the backward edge (u, v), and again v would be reachable – so every such edge is empty: $f(e) = 0$. By the flow-value lemma,

$$|f| = \underbrace{\sum_{A \to B} f(e)}_{= \text{cap}(A,B)} - \underbrace{\sum_{B \to A} f(e)}_{= 0} = \text{cap}(A, B). \tag{1}$$

We are done: for any flow f', the maxflow $\leq$ mincut lemma gives $|f'| \leq \text{cap}(A, B) = |f|$. So no flow has value larger than f, which is to say that f is a maximum flow. □

Of course, this says nothing at all unless the algorithm does terminate. It does, as long as the capacities are integers.

Theorem 2 (Running time)**.** On a network with integer capacities whose maximum flow has value F, Ford–Fulkerson terminates after at most F augmentations, and runs in time $O(EF)$.

Proof. Since the capacities are integers, so is every residual capacity, and hence so is the bottleneck capacity of any augmenting path. Being positive, it is at least 1, so each augmentation increases the value of the current flow by at least 1. The value starts at 0 and never exceeds F, so there are at most F augmentations.

Each augmentation costs a single graph search from s in G_f, plus the update along the path, which is $O(V + E) = O(E)$ time.[a] □

[a] As usual we may assume every vertex lies on some s–t path, so that $V = O(E)$; otherwise the bound reads $O((V + E)F)$.

Note that this running time is *not* polynomial in the size of the input: F can be as large as the sum of the capacities, and capacities are written in binary. So this is a pseudo-polynomial time algorithm (see lecture 2). Choosing augmenting paths more cleverly fixes this, and that is a topic for lecture.

The Maxflow–Mincut Theorem

The proof of Theorem 1 did more work than we asked of it. It did not just show that f is a maximum flow: it also handed us a cut whose capacity equals $|f|$, and by the very same inequality that cut must be a minimum one. That is the maxflow–mincut theorem, and we get it for free.

Theorem 1 (Maxflow–mincut)**.** The value of a maximum flow equals the capacity of a minimum cut.

Proof. Let f be a maximum flow. There is no augmenting path with respect to f: if there were one, with bottleneck capacity $\Delta > 0$, augmenting along it would produce a valid flow of value $|f| + \Delta > |f|$, contradicting maximality. So the construction in the proof of Theorem 1 applies verbatim to f, and (1) gives a cut (A, B) with $\mathrm{cap}(A, B) = |f|$.

By the maxflow $\leq$ mincut lemma, every cut has capacity at least $|f|$. Hence (A, B) is a minimum cut, and its capacity equals the value of a maximum flow. □

The cut built in the proof of Theorem 1 is worth remembering as an *algorithm*, not just as a proof step: given a maxflow f, you can compute a mincut in $O(V + E)$ time by running one graph search from s in G_f and letting A be the set of vertices you reach. We will use this in the problem below.

Problem

Problem 1

Let $G = (V, E)$ be a flow network with source s, sink t, and **integer** capacities $c(e)$. You are given, as input:

- the network G (with its capacities);
- a maximum flow f of G (that is, the value $f(e)$ on every edge – you do not have to compute it); and
- a particular edge $e^\star \in E$.

Let G' be the network obtained from G by increasing the capacity of $e^\star$ by 10, leaving everything else unchanged:

$$c'(e^\star) = c(e^\star) + 10, \qquad c'(e) = c(e) \text{ for } e \neq e^\star.$$

Design an algorithm that computes a maximum flow of G' in $O(V + E)$ time.

(a) Show that f is still a valid flow in G', and that the value of a maximum flow of G' is at most $|f| + 10$.

(b) Give an $O(V + E)$-time algorithm to compute a maximum flow of G', and prove that it is correct and that it runs in the stated time.

Problem Solving Tips

You are not going to beat $O(V+E)$ by computing a maxflow from scratch, so the only sensible plan is to *start from* f and repair it. You already know an algorithm that takes a valid flow and improves it one step at a time, and you know from part (a) how much improving is left to do.