CS 341: Algorithms

Lecture 17: Max flow = Min cut

Slides due to Éric Schost and based on lecture notes by many other CS341 instructors

David R. Cheriton School of Computer Science, University of Waterloo

Winter 2026

Sir Tony Hoare

- Age 92
- Inventor of quicksort/quickselect
- Inventor of “null”
- Turing Award (1980)
- Concurrency and correctness

Cuts

Cuts

Definition

- a **cut** is a partition of the vertices into sets $\boldsymbol{A}$ and $\boldsymbol{B = V - A}$, with $s \in A$ and $t \in B$.
- the **capacity** of the cut is

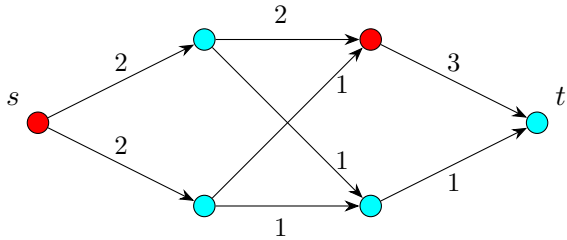
$$c(A) = \sum_{e:A \to B} c(e)$$

  (does not depend on any flow, only on the graph and its capacities)
- if f is a flow, the **outgoing** and **incoming** flows of the cut are

$$v_{\text{out}}(f, A) = \sum_{e:A \to B} f(e), \quad v_{\text{in}}(f, A) = \sum_{e:B \to A} f(e)$$

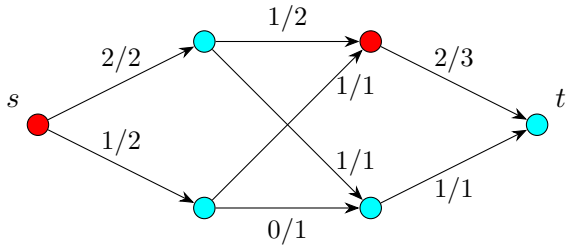
- **net flow** is $v_{\text{net}}(f, A) = v_{\text{out}}(f, A) - v_{\text{in}}(f, A)$

Examples



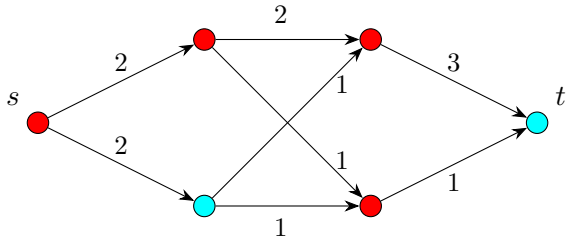
- A is in red and B in light blue,
- capacity is $2 + 2 + 3 = 7$,

Examples



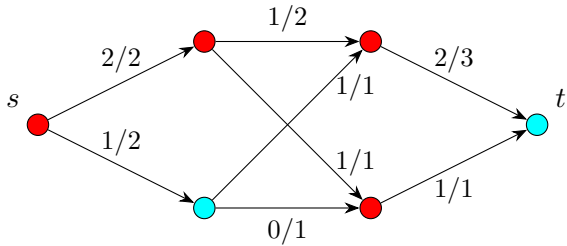
- A is in red and B in light blue,
- capacity is $2 + 2 + 3 = 7$,
- outgoing flow is $2 + 1 + 2 = 5$,
- incoming flow is $1 + 1 = 2$,
- net flow is 3

Examples



- A is in red and B in light blue,
- capacity is $2 + 3 + 1 = 6$,

Examples



- A is in red and B in light blue,
- capacity is $2 + 3 + 1 = 6$,
- outgoing flow is $1 + 2 + 1 = 4$,
- incoming flow is 1,
- net flow is 3

Flows and cuts

Claim

For any flow f and any cut A, we have

$$\mathsf{Val}(f) = v_{\text{out}}(f, A) - v_{\text{in}}(f, A)$$

Proof: induction on A.

- true when $A = \{s\}$, by definition.
- suppose this is true for a cut A, $B = V - A$, we show this is true for the cut $A' = A \cup \{v\}$, $B' = B - \{v\}$, for any vertex $v \in B$ (with $v \neq t$).

What we need to do:

- relate $v_{\text{out}}(f, A)$ to $v_{\text{out}}(f, A')$,
- relate $v_{\text{in}}(f, A)$ to $v_{\text{in}}(f, A')$.

Step 1

$$\begin{aligned} v_{\text{out}}(f, A) &= \sum_{e:A\to B} f(e) \\ &= \sum_{e:A\to v} f(e) \quad + \quad \sum_{e:A\to B'} f(e) \end{aligned}$$

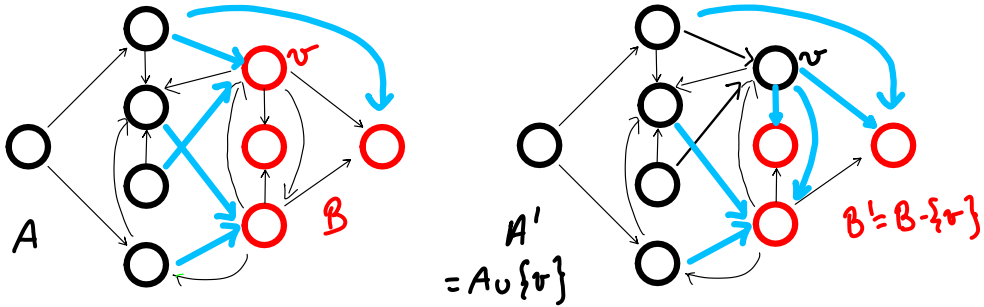
and

$$\begin{aligned} v_{\text{out}}(f, A') &= \sum_{e:A'\to B'} f(e) \\ &= \sum_{e:A\to B'} f(e) \quad + \quad \sum_{e:v\to B'} f(e). \end{aligned}$$

so

$$v_{\text{out}}(f, A') = v_{\text{out}}(f, A) - \sum_{e:A\to v} f(e) + \sum_{e:v\to B'} f(e)$$

Step 1



so

$$v_{\text{out}}(f, A') \quad = \quad v_{\text{out}}(f, A) - \sum_{e:A \to v} f(e) + \sum_{e:v \to B'} f(e)$$

Step 2

$$\begin{aligned} v_{\text{in}}(f, A) &= \sum_{e:B \to A} f(e) \\ &= \sum_{e:v \to A} f(e) \quad + \quad \sum_{e:B' \to A} f(e) \end{aligned}$$

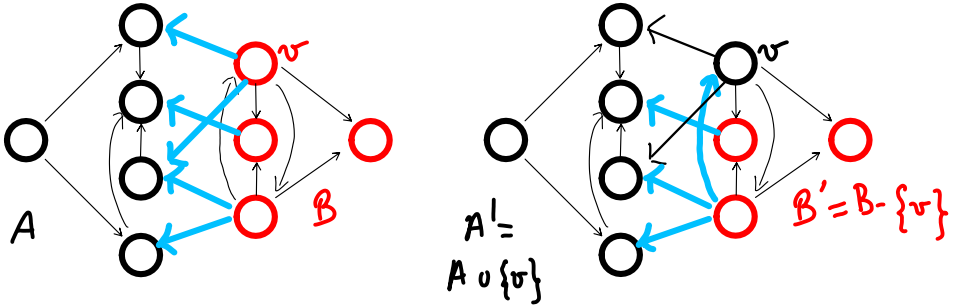
and

$$\begin{aligned} v_{\text{in}}(f, A') &= \sum_{e:B' \to A'} f(e) \\ &= \sum_{e:B' \to A} f(e) \quad + \quad \sum_{e:B' \to v} f(e). \end{aligned}$$

so

$$v_{\text{in}}(f, A') = v_{\text{in}}(f, A) - \sum_{e:v \to A} f(e) + \sum_{e:B' \to v} f(e)$$

Step 2



so

$$v_{\text{in}}(f, A') = v_{\text{in}}(f, A) - \sum_{e:v \to A} f(e) + \sum_{e:B' \to v} f(e)$$

Step 3

Because f is a flow, we have

$$\sum_{e:v\to A} f(e) \;+\; \sum_{e:v\to B'} f(e) \;=\; \sum_{e:B'\to v} f(e) \;+\; \sum_{e:A\to v} f(e)$$

so

$$\begin{aligned} v_{\text{out}}(f,A') &= v_{\text{out}}(f,A) - \sum_{e:A\to v} f(e) + \sum_{e:v\to B'} f(e) \\ &= v_{\text{out}}(f,A) - \sum_{e:v\to A} f(e) + \sum_{e:B'\to v} f(e) \end{aligned}$$

and still

$$v_{\text{in}}(f,A') = v_{\text{in}}(f,A) - \sum_{e:v\to A} f(e) + \sum_{e:B'\to v} f(e)$$

This gives

$$\begin{aligned} v_{\text{out}}(f,A') - v_{\text{in}}(f,A') &= v_{\text{out}}(f,A) - v_{\text{in}}(f,A) \\ &= \mathsf{Val}(f). \end{aligned}$$

Maximum flow and minimal cut

Consequence

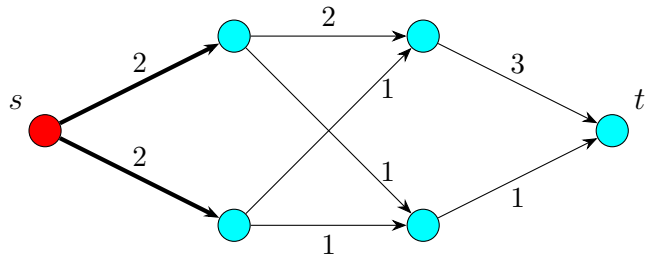
for **any flow** f and **any cut** A, we have $\mathsf{Val}(f) \leq c(A)$.

proof:

$$\begin{aligned} \mathsf{Val}(f) &= v_{\text{out}}(f, A) - v_{\text{in}}(f, A) \\ &\leq v_{\text{out}}(f, A) \\ &\leq c(A) \end{aligned}$$

- so the **maximal value** of a flow $\leq$ **minimal capacity** of a cut
- and if we find **any** flow and cut with equality, they are optimal

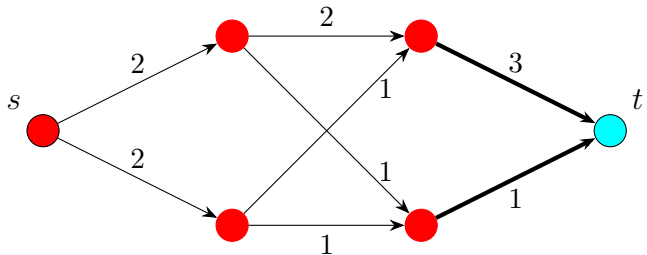
Example 1



Max flow?

- we found 4 in the previous lecture
- with $A = \{s\}$, $c(A) = 4$
- so max flow = min cut = 4

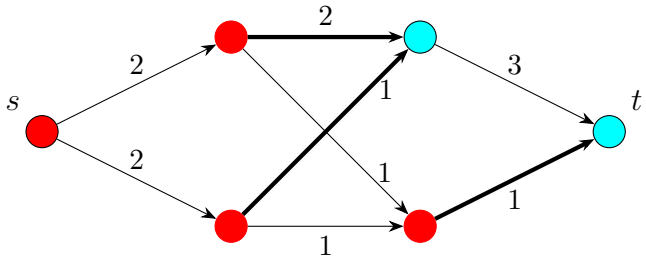
Example 1



Max flow?

- we found 4 in the previous lecture
- with $A = V - \{s\}$, $c(A) = 4$
- so max flow = min cut = 4

Example 1

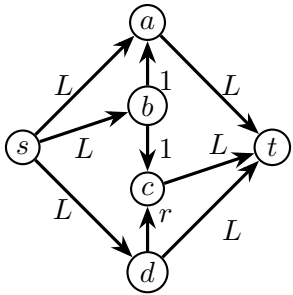


Max flow?

- we found 4 in the previous lecture
- $c(A) = 4$
- so max flow = min cut = 4

Example 2

A bonus example from last lecture: $r = (\sqrt{5} - 1)/2 \simeq 0.618$, $L \geq 1$

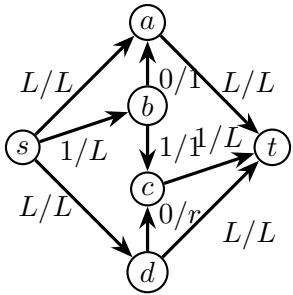


Max flow?

- easy to get $2L + 1$
- with $A = \{s, a, b\}$, $c(A) = 2L + 1$
- so max flow = min cut $= 2L + 1$

Example 2

A bonus example from last lecture: $r = (\sqrt{5} - 1)/2 \simeq 0.618$, $L \geq 1$

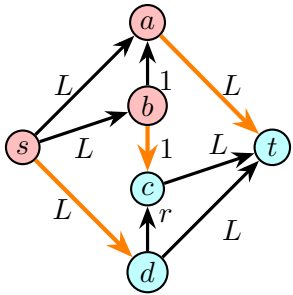


Max flow?

- easy to get $2L + 1$
- with $A = \{s, a, b\}$, $c(A) = 2L + 1$
- so max flow = min cut $= 2L + 1$

Example 2

A bonus example from last lecture: $r = (\sqrt{5} - 1)/2 \simeq 0.618$, $L \geq 1$



Max flow?

- easy to get $2L + 1$
- with $A = \{s, a, b\}$, $c(A) = 2L + 1$
- so max flow = min cut $= 2L + 1$

Max flow = min cut

Claim

no improving path in G_f $\implies$ can find a cut A such that $\mathsf{Val}(f) = c(A)$

Proof (sketch)

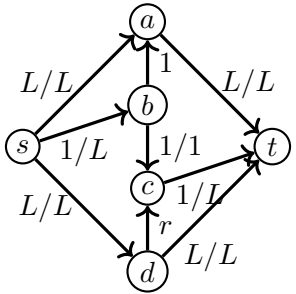
- let A be of vertices **reachable from s** in G_f
- this is a cut (s is in A, no path $s \to t$ in G_f so t is in $B = V - A$)
- needs a bit more work: $\mathsf{Val}(f) = c(A)$

Consequences

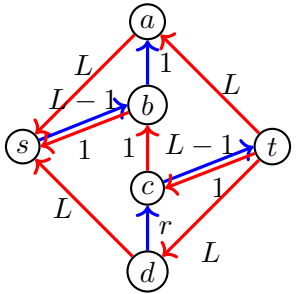
1. max flow = min cut (because no improving path for a max flow)
2. Ford-Fulkerson computes a max flow (because no improving path at the end)
(when/if it terminates)

WHY IS THERE A MAX FLOW?

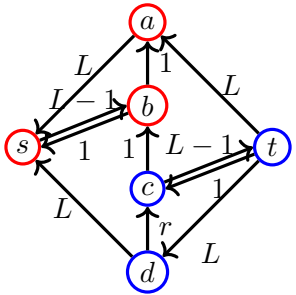
Example (same graph as before, $L > 1$)



Example (same graph as before, $L > 1$)



Example (same graph as before, $L > 1$)



$\boldsymbol{A}$ in red, $\boldsymbol{B} = \boldsymbol{V} - \boldsymbol{A}$ in blue

A few remarks
(time permitting)

Remark 1: Edmonds-Karp

A strategy that refines Ford-Fulkerson: choose a **shortest** path (BFS)

Edmonds-Karp Proof I

Let f be the old flow, γ a shortest augmenting path, and f' the new (augmented) flow.

Theorem

The distances from s do not decrease: for all $v \in V$, $\delta_{f'}(s, v) \geq \delta_f(s, v)$.

Proof:

Suppose to the contrary $\delta_{f'}(s, v) < \delta_f(s, v)$. Choose v with minimal distance $\delta_{f'}(s, v)$, and let (u, v) be the last edge in a shortest path in $G_{f'}$.

- $\delta_{f'}(s, u) \geq \delta_f(s, u)$ because v is minimal.
- (u, v) is a new edge only if (v, u) is on γ.
- Since (u, v) is on a shortest path, $\delta_{f'}(s, v) = \delta_{f'}(s, u) + 1$.
- Since (v, u) is on a shortest path, $\delta_f(s, u) = \delta_f(s, v) + 1$.
- $\implies \delta_{f'}(s, v) \geq \delta_f(s, v) + 2$.

Edmonds-Karp Proof II

Remarks:

- The augmenting paths in Edmonds-Karp get monotonically longer.
- Each improving step deletes at least one edge.
- New edges are created, **but** any shortest path that uses them is at least two steps longer than the current shortest.

Theorem

Edmonds-Karp runs in $O(m^2 n)$ time

Proof

- Each improving step runs in time $O(m)$.
- At distance d, each edge can be critical in at most one augmenting path
- Paths range from length 1 to $n - 1$.

Remark 2: thick paths

A slightly weaker strategy to refine Ford-Fulkerson: choose a path that **maximizes the bottleneck** capacity x.

Key ideas

- finding the thickest path: similar to Dijkstra
  - Dijkstra minimizes $\sum_{e\in\gamma} w(e)$
  - here we maximize $\min_{e\in\gamma} c(e)$
- in G_f, there is a path with $x \geq (M - \mathsf{Val}(f))/2m$, $M =$ max flow so

$$\mathsf{Val}(f') \geq \mathsf{Val}(f) + (M - \mathsf{Val}(f))/2m$$

  (takes some work)
- if capacities are integers, implies we do $O(m \log(M))$ iterations
- total $\boldsymbol{O(m^2 \log(n) \log(M))}$

Remark 3: maximal flow from linear programming

Equations for the max flow problem:

1. create a variable $f_{u,v}$ for each edge (u, v) and the linear constraints

$$f_{u,v} \geq 0, \quad f_{u,v} \leq c(u, v), \quad \sum_{(u,v) \text{ edge}} f_{u,v} = \sum_{(v,w) \text{ edge}} f_{v,w}$$

2. maximize

$$\sum_{(s,v) \text{ edge}} f_{s,v}.$$

- this is an instance of a **linear programming** problem
- max flow / min cut special case of **linear programming duality** (max something = min something else)

  (takes work)