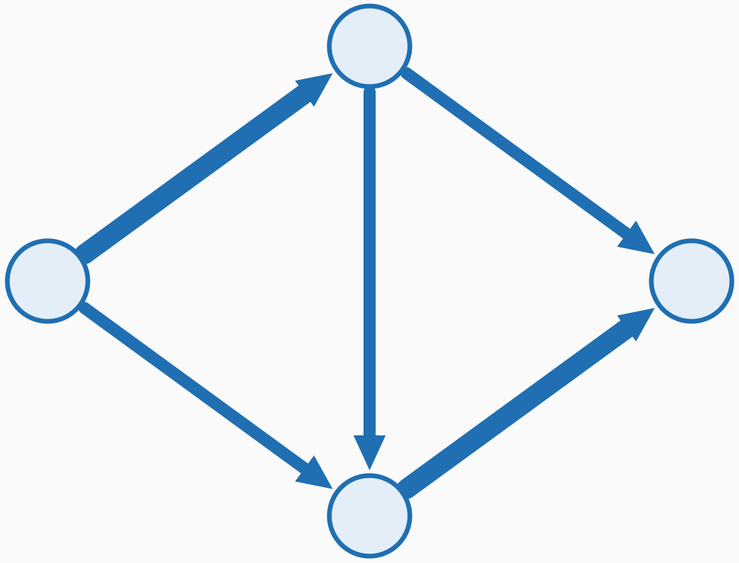
ALGORITHMS AND DATA STRUCTURES · LECTURE 9

Maximum flow

Flow networks and the Ford–Fulkerson algorithm



Heng Guo

School of Informatics, University of Edinburgh

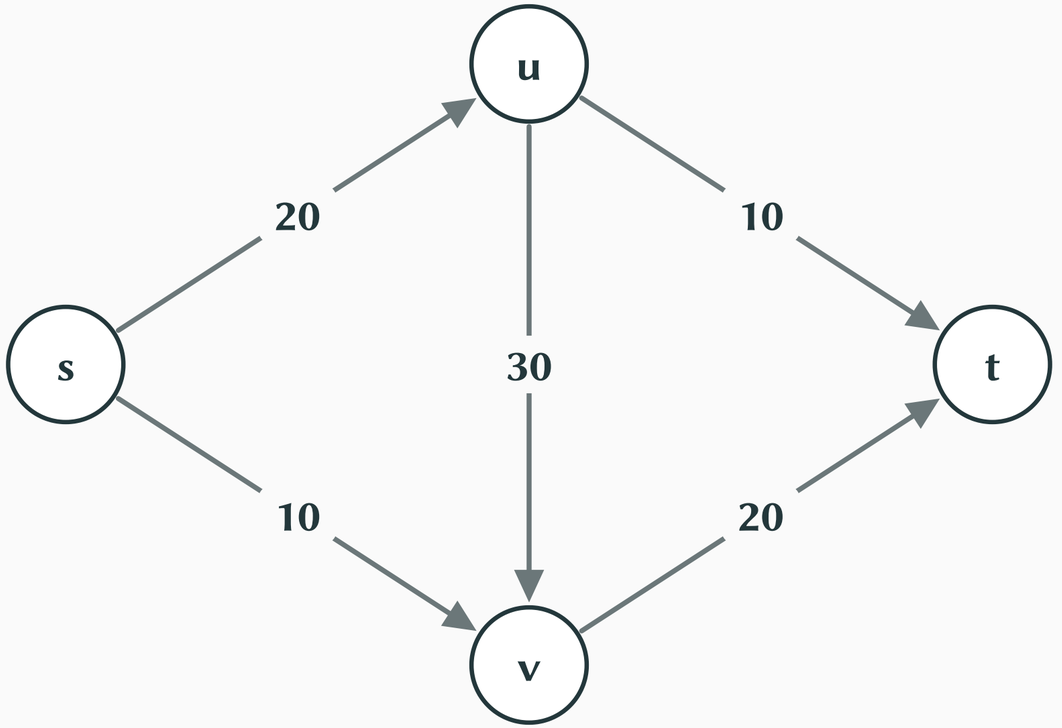
Monday 19 October 2026 · 16:10–17:00

Today: maximum flow

Last lecture: Kruskal's and Prim's algorithms, made fast by priority queues and Union-Find.

New topic: networks whose edges have **capacities**: pipes, roads, data links.

? *How much can we send from* s *to* t*, and along which routes?*



Today: flow networks → a greedy attempt → residual graphs → Ford–Fulkerson → analysis

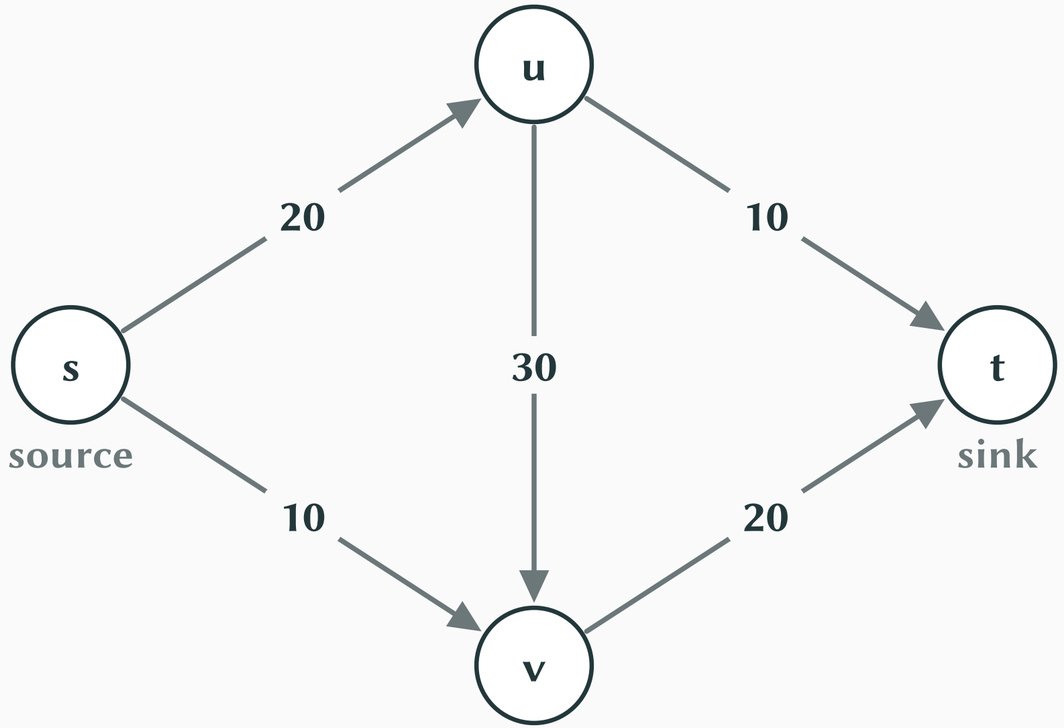
Flow networks

Definition.
A **flow network** is a directed graph $G = (V, E)$ with

a **capacity** $c_e \geqslant 0$ on every edge e,
a **source** s and a **sink** t.

The other nodes, $V \setminus \{s, t\}$, are **internal**.

- No edge enters s and no edge leaves t.
- Every node has an edge, so $n \leqslant 2m$, where $n = |V|$ and $m = |E|$.
- All capacities are **integers**.



Flows: capacity and conservation

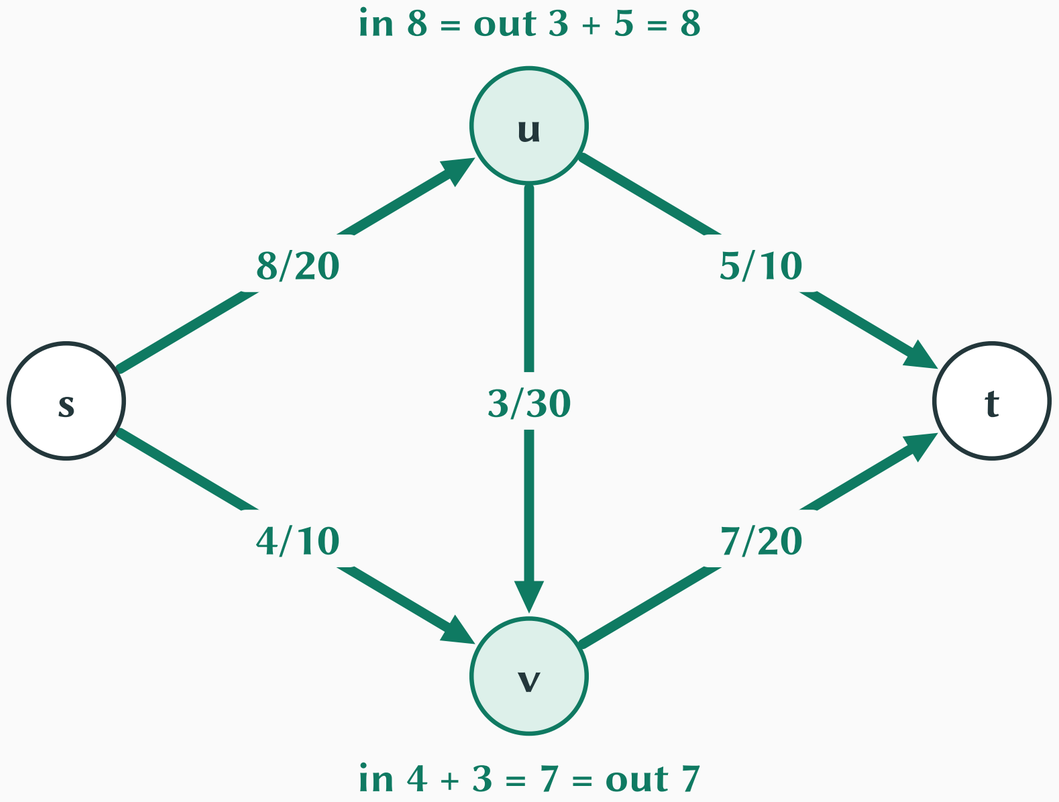
Definition.
A **flow** assigns a number $f(e) \geqslant 0$ to every edge e such that

Capacity: $f(e) \leqslant c_e$ for every edge e,

Conservation: $f^{in}(v) = f^{out}(v)$ for every internal node v.

$f^{in}(v)$ is the total flow into v, $f^{out}(v)$ the total flow out of v.

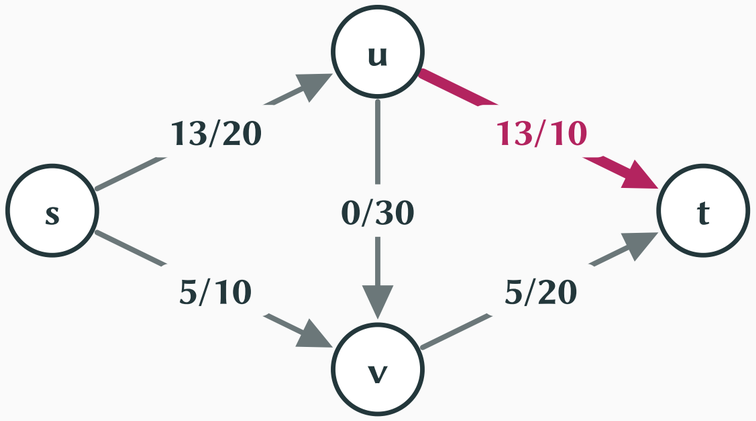
The source creates flow, the sink absorbs it; nothing is lost or created in between.



Is it a flow?

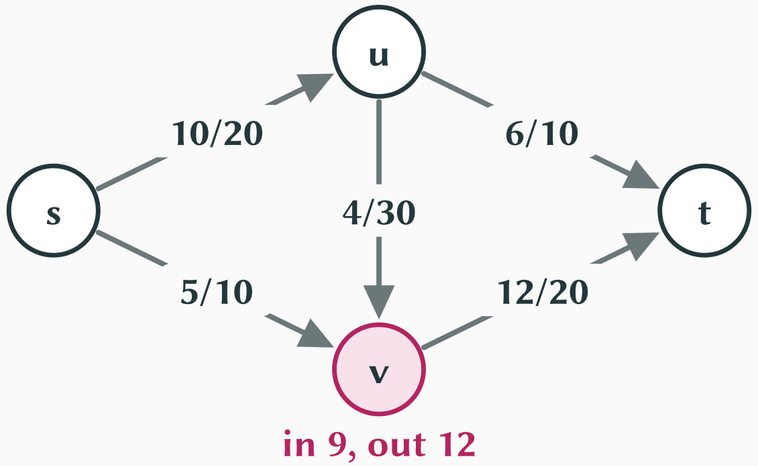
? *Which of these labellings are flows?*

(a)



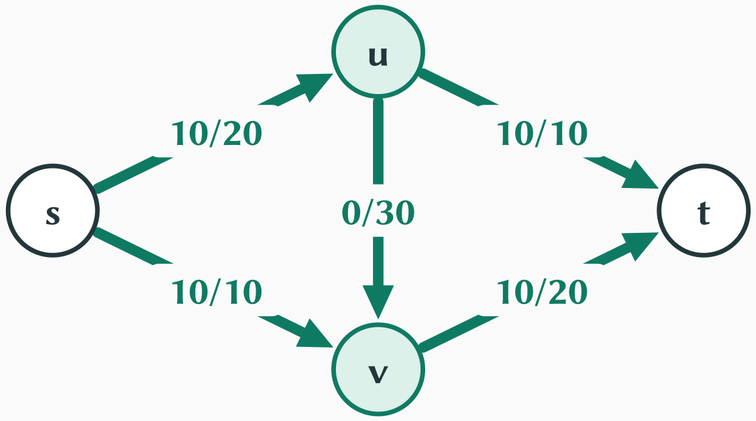
No: capacity violated on (u, t): $13 > 10$.

(b)



No: at v the flow in is 9, the flow out is 12.

(c)



Yes: a flow of value 20.

The value of a flow

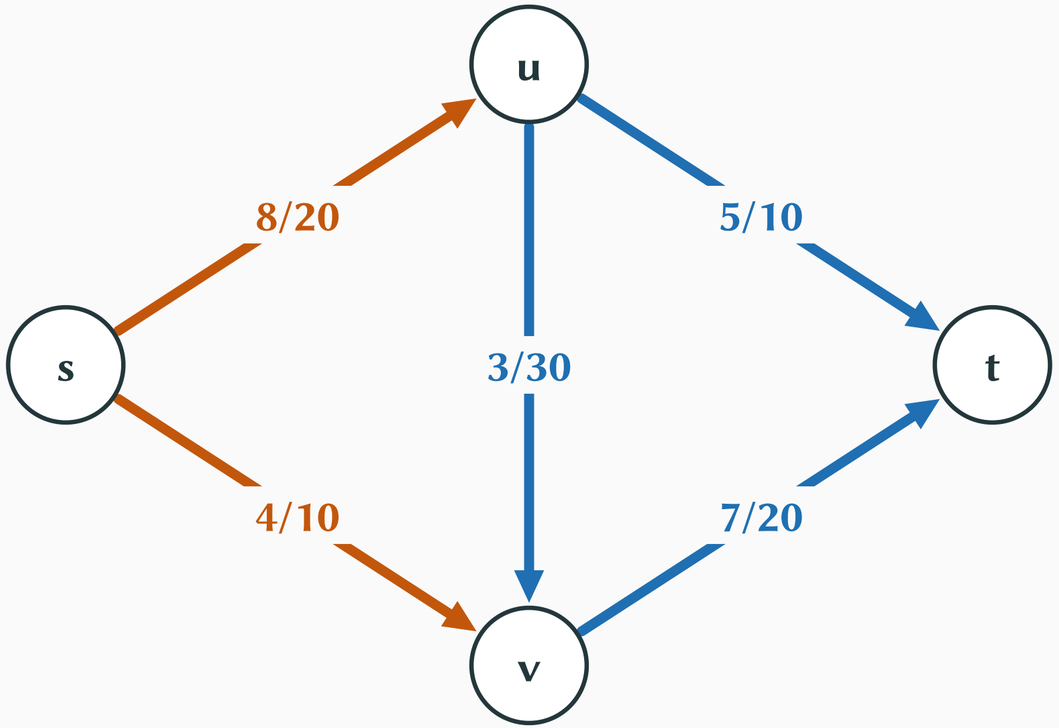
Value. $\mathrm{val}(f) = f^{\mathrm{out}}(s)$,
the total flow leaving the source.

Here $\mathrm{val}(f) = 8 + 4 = 12$.

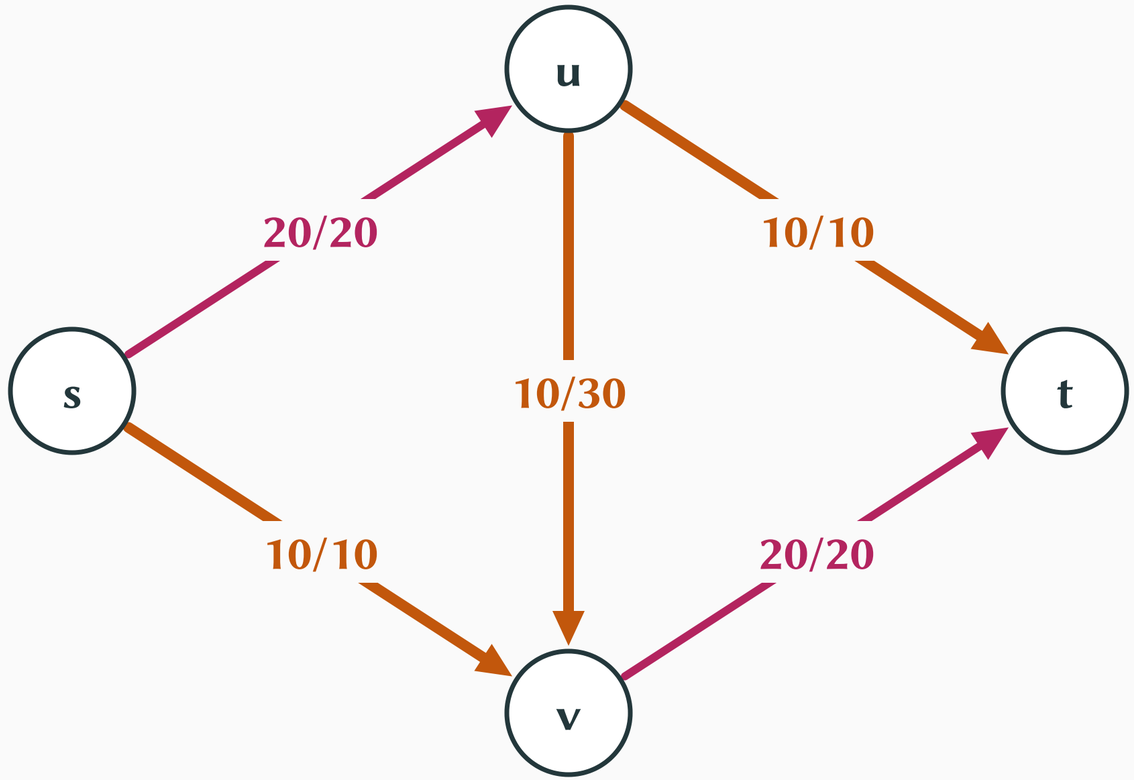
The same amount arrives at t: $5 + 7 = 12$.

Problem. Given a flow network,
find a flow of **maximum value**.

Every flow has $\mathrm{val}(f) \leqslant C$, the total capacity out of s. Here $C = 20 + 10 = 30$.



A first attempt: push greedily



$\mathrm{val}(f) = 30$

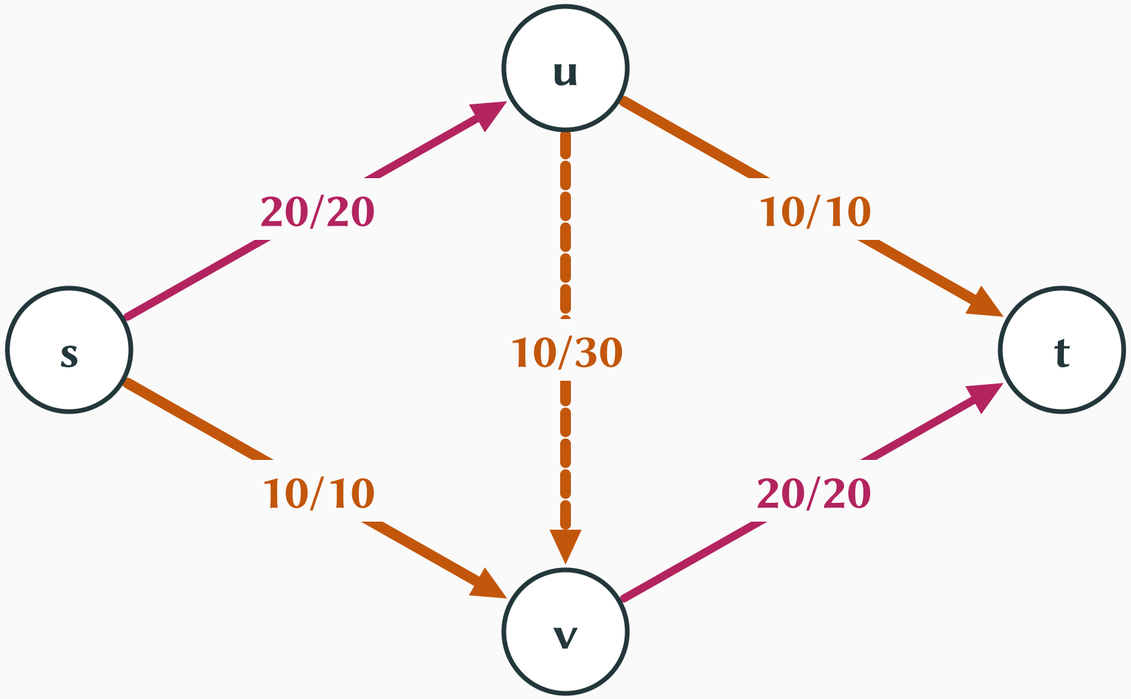
Greedy: start with $f = 0$. Repeatedly push as much as possible along an s–t path with spare capacity.

Path s, u, v, t with spare capacities $20, 30, 20$: push 20.

No such path is left: from s we reach only v, and (v, t) is full. **Stuck at 20.**

> But a flow of value 30 exists! The 20 units on (u, v) were a mistake.

Undoing flow



Stuck at 20. Idea: **undo** some flow.

Take back 10 of the 20 units on (u, v): now u has 10 too many, v has 10 too few.

Repair: send 10 along (u, t) and 10 along (s, v): a flow again, of value 30.

Net effect: the path $s \to v \Rightarrow u \to t$, going **backward** along (u, v).

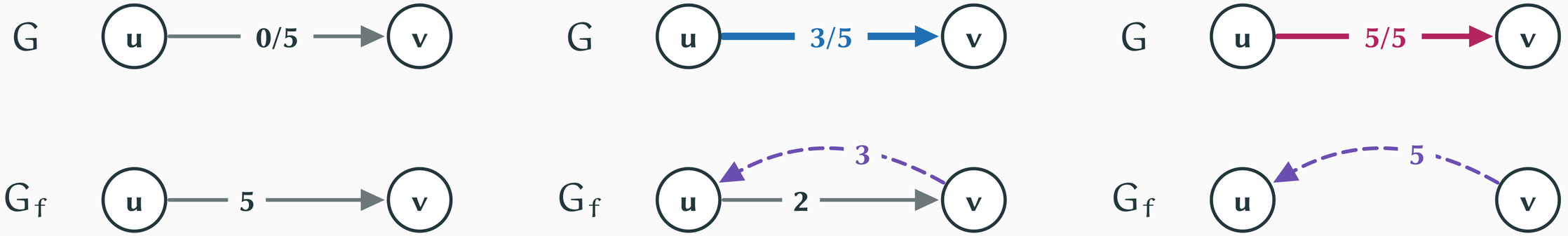
? *How do we find such paths systematically?*

The residual graph

Definition. The **residual graph** G_f of a flow f has the same nodes as G.
Every edge $e = (u, v)$ of G gives

- a **forward edge** (u, v) of capacity $c_e - f(e)$, if positive: room for more flow,
- a **backward edge** (v, u) of capacity $f(e)$, if positive: flow we can take back.

The edges of G_f are also called **arcs**.



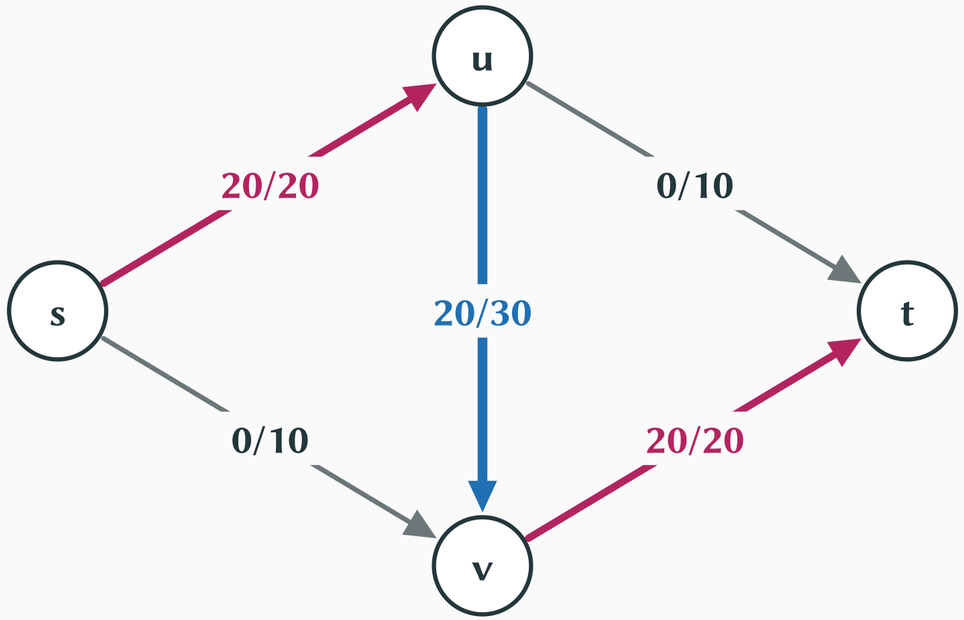
empty: forward edge only

partly full: both edges

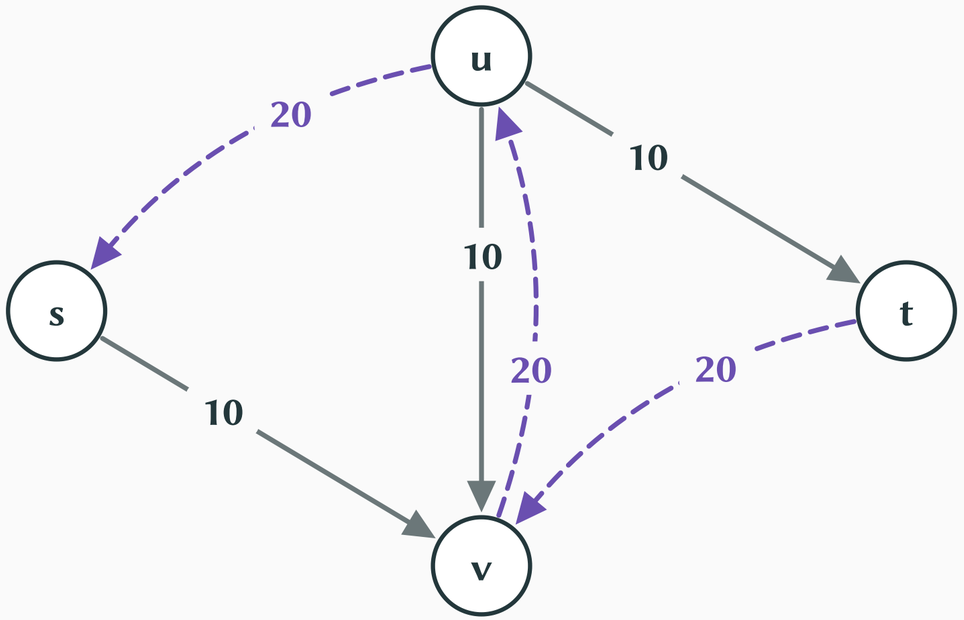
full: backward edge only

The residual graph of a flow

G with a flow f



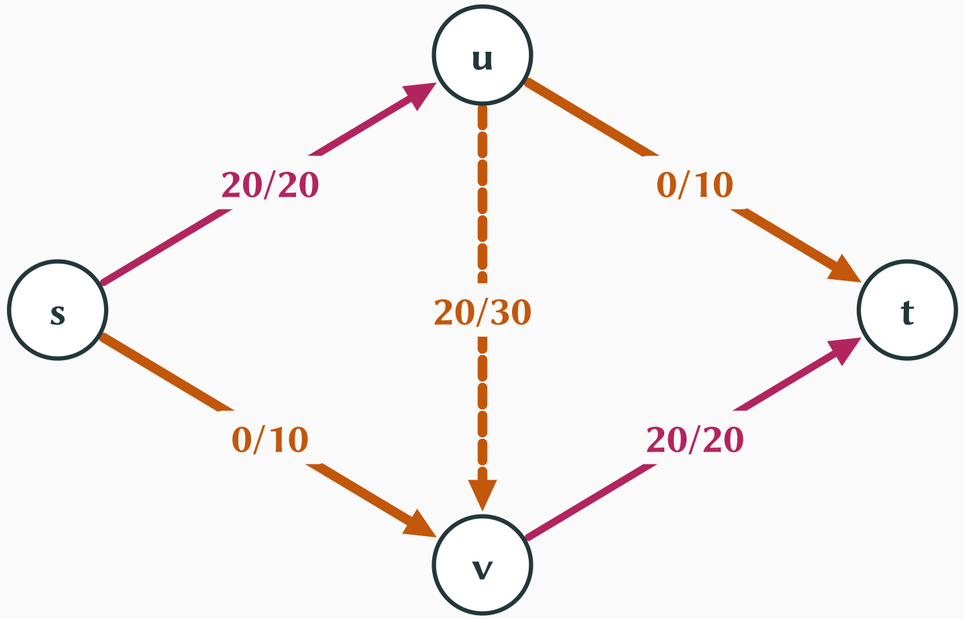
the residual graph G_f



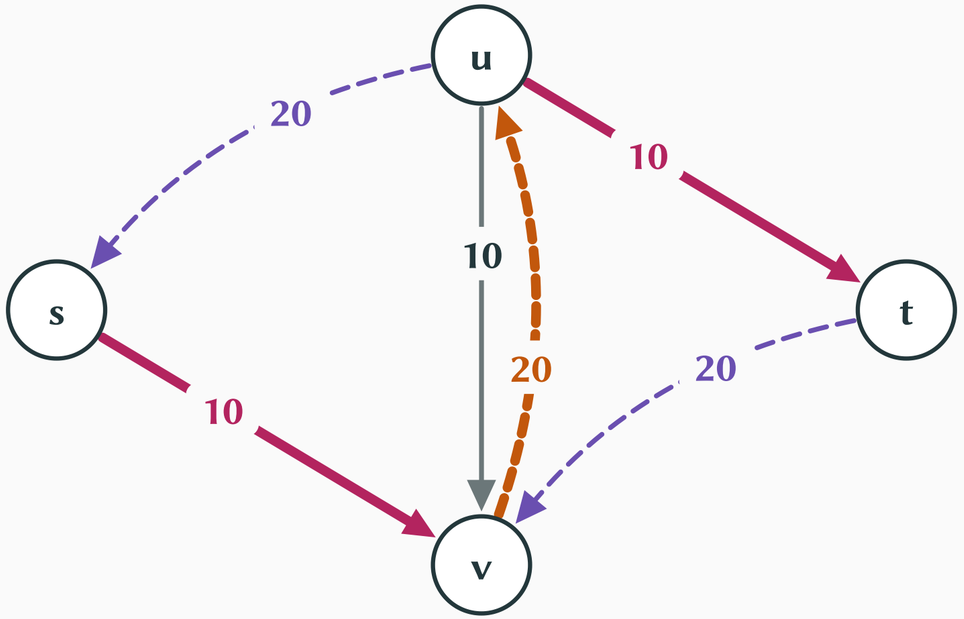
Backward edges are dashed purple. Edges of capacity 0 do not exist.

Augmenting path and bottleneck

G with the flow f



the residual graph G_f



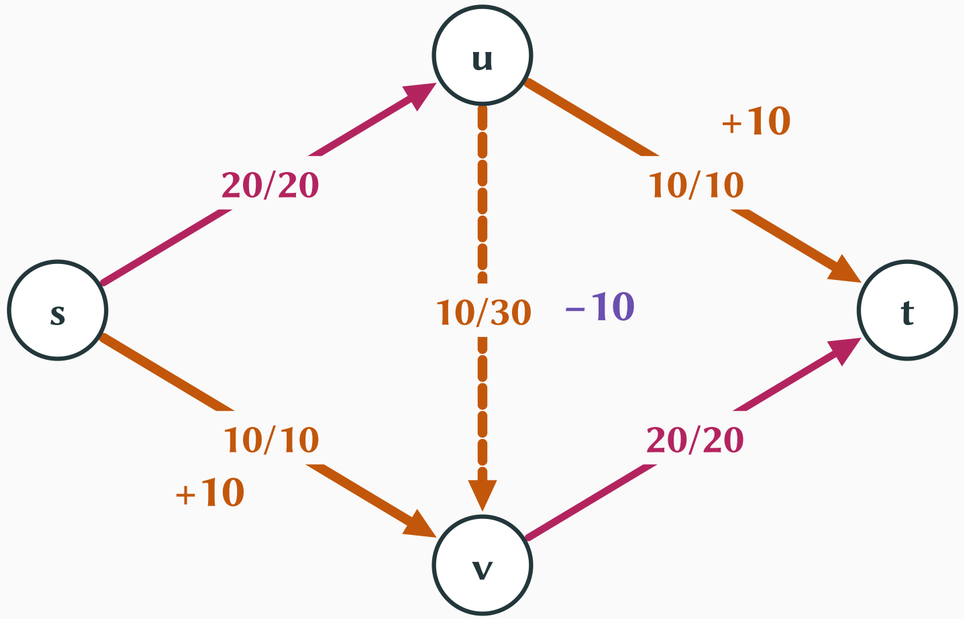
An **augmenting path** P is a path from s to t in G_f: here s, v, u, t.

Its **bottleneck** is the smallest capacity on P:

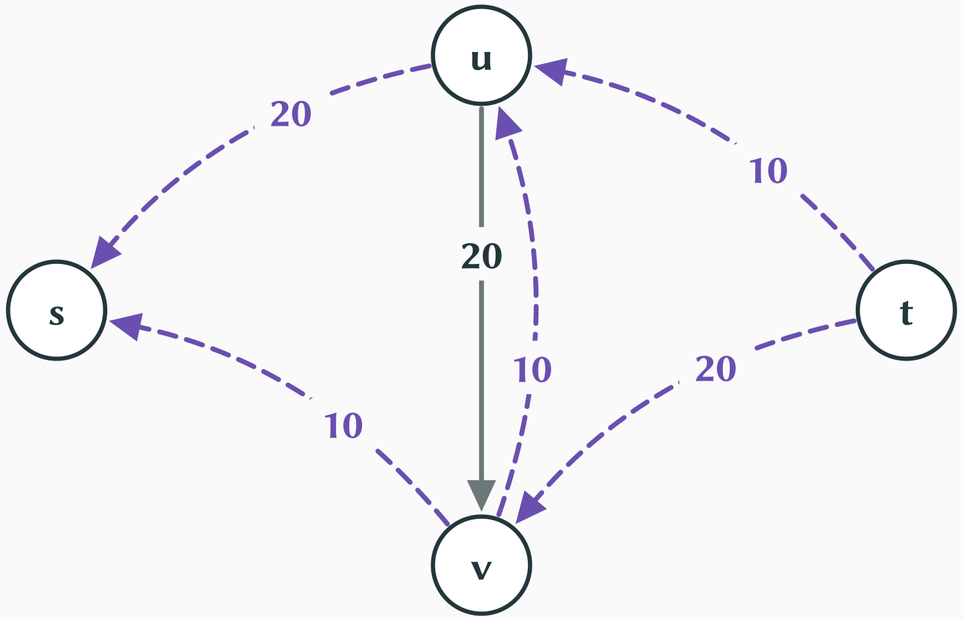
$$\text{bottleneck}(P, f) = \min\{10, 20, 10\} = 10.$$

Augmenting the flow

G with the new flow f'



the new residual graph $G_{f'}$



Augment along P by $b = 10$: **forward** edges get $+b$, **backward** edges get $-b$.

In $G_{f'}$ no edge leaves s: there is no augmenting path left.

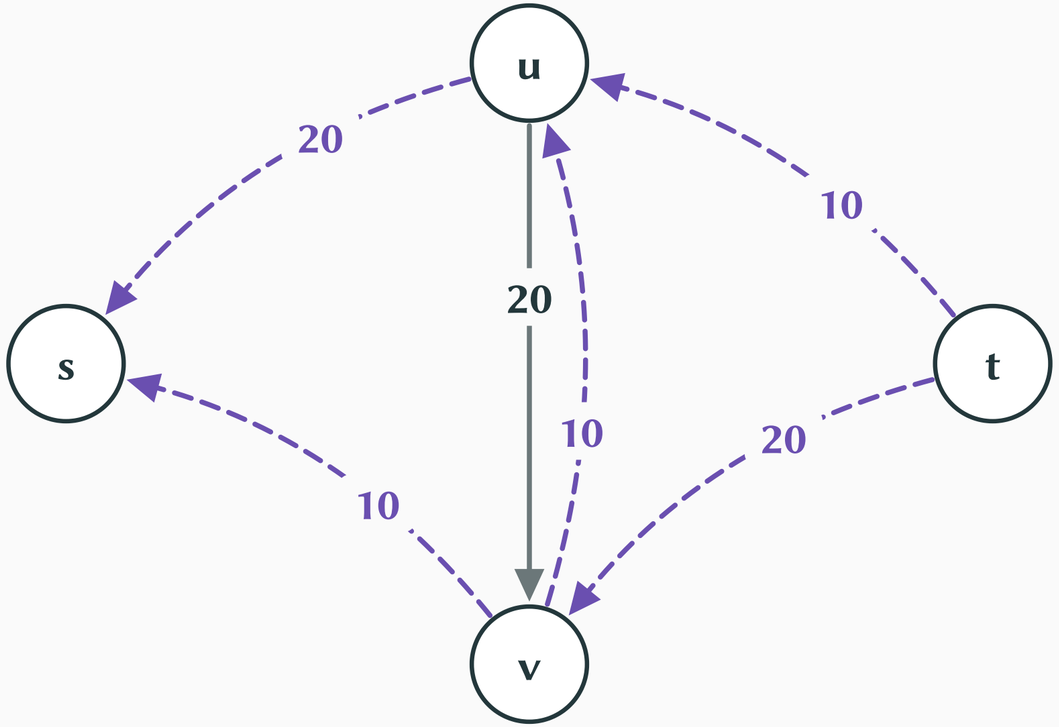
The Ford–Fulkerson algorithm

Ford–Fulkerson(G, s, t)

1 $f(e) \leftarrow 0$ for every edge e
2 **while** G_f has a path from s to t
3 choose such a path P
4 **Augment**(f, P)
5 **return** f

Augment(f, P)

6 $b \leftarrow$ bottleneck(P, f)
7 **for** each edge e of G used by P
8 **if** P uses e forward: $f(e) \leftarrow f(e) + b$
9 **else** backward: $f(e) \leftarrow f(e) - b$



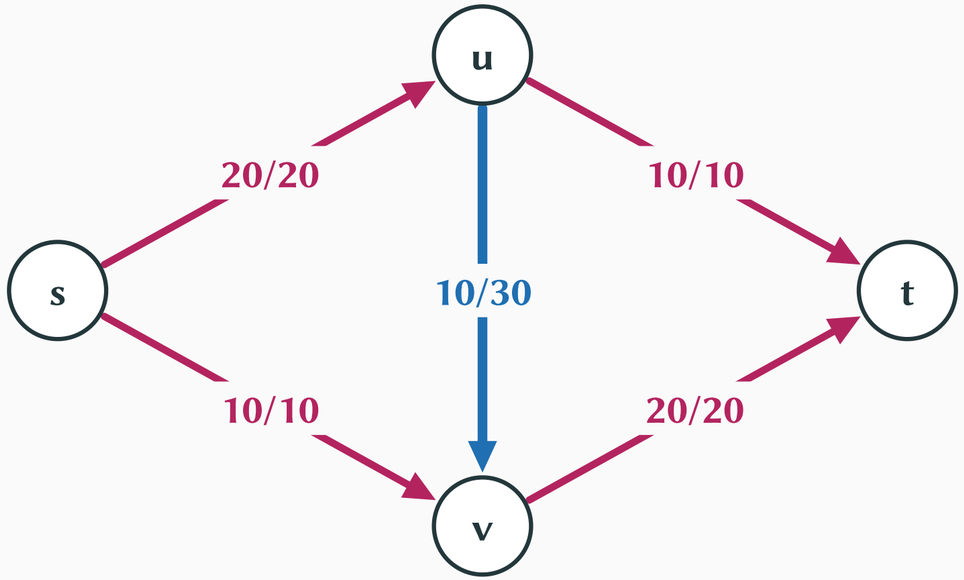
$\text{val}(f) = 30$

No path from s to t: return f, of value 30.

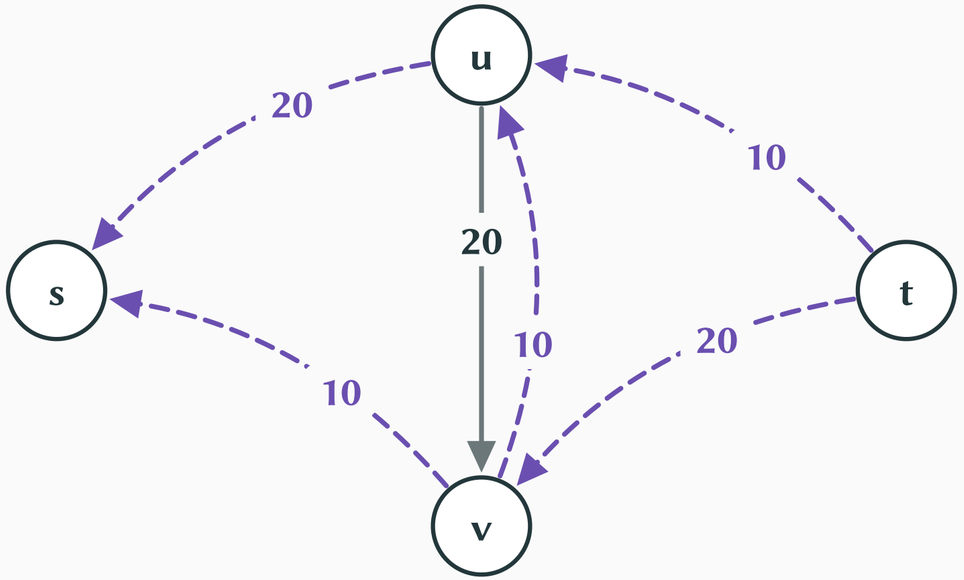
Paths: s, u, v, t with $b = 20$; s, v, u, t with $b = 10$.

Run it yourself

G with the flow f



the residual graph G_f



$\mathrm{val}(f) = 30$

no augmenting path left

1. s, u, t: +10 2. s, v, t: +10 3. s, u, v, t: +10

stop: value 30 after 3 augmentations

What do we need to know?

1. **Feasibility.** Is the result of every iteration a flow? — TODAY
2. **Termination.** Does the algorithm stop? — TODAY
3. **Running time.** How many iterations, how long is each? — TODAY, LECTURE 10
4. **Optimality.** Is the flow it returns a maximum flow? — LECTURE 11

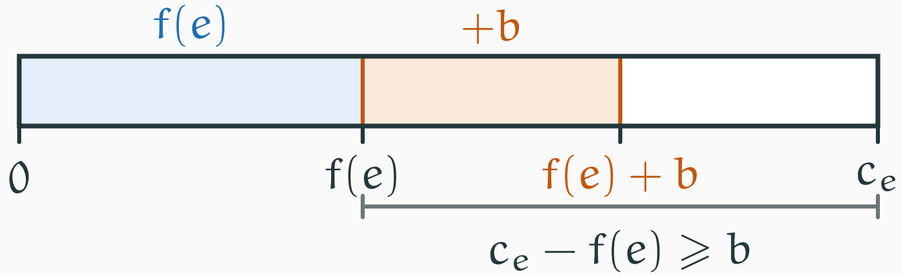
Lemma. If f is a flow and P an augmenting path,
then $f' = \text{augment}(f, P)$ is a flow,
and $\text{val}(f') = \text{val}(f) + \text{bottleneck}(P, f)$.

Why f′ is a flow: capacity

Only edges on P change. Let e be one of them, and $b = \text{bottleneck}(P, f)$.

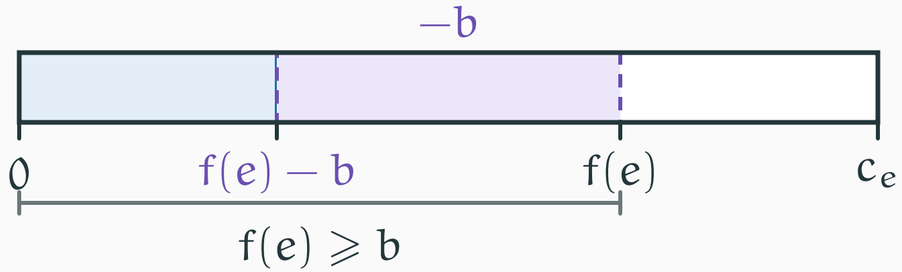
Forward edge: $b \leqslant c_e - f(e)$, so

$$0 \leqslant f(e) \leqslant f(e) + b \leqslant f(e) + (c_e - f(e)) = c_e.$$



Backward edge: $b \leqslant f(e)$, so

$$c_e \geqslant f(e) \geqslant f(e) - b \geqslant f(e) - f(e) = 0.$$

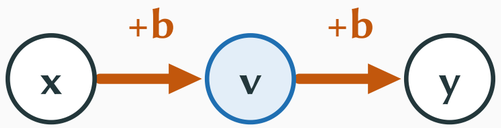


So $0 \leqslant f'(e) \leqslant c_e$ for every edge: **capacity holds**.
Edges not on P are unchanged.

Why f′ is a flow: conservation

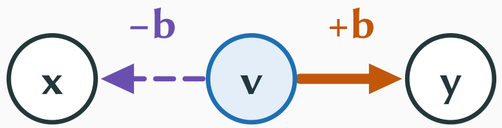
P is simple: it visits an internal node v at most once,
coming in by one edge of G_f and going out by another.
Each is forward or backward: four cases.

Forward, forward



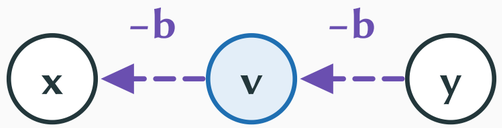
$f^{in}(v)$: $+b$
$f^{out}(v)$: $+b$

Backward, forward



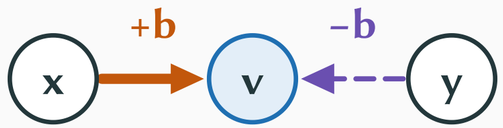
$f^{in}(v)$: 0
$f^{out}(v)$: $-b + b = 0$

Backward, backward



$f^{in}(v)$: $-b$
$f^{out}(v)$: $-b$

Forward, backward



$f^{in}(v)$: $+b - b = 0$
$f^{out}(v)$: 0

In every case $f^{in}(v)$ and $f^{out}(v)$ change by the **same** amount,
so $f^{in}(v) = f^{out}(v)$ still holds.

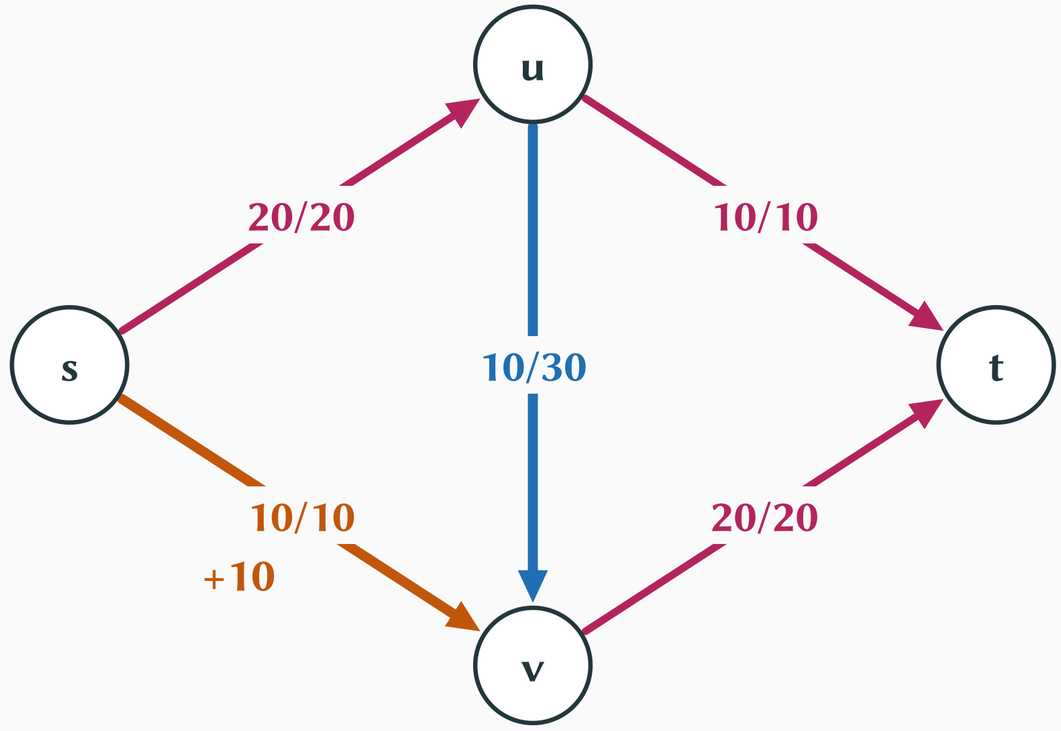
Why the value grows

$\mathrm{val}(f) = f^{\mathrm{out}}(s)$: the flow on the edges leaving s.

P starts at s with a **forward** edge (s, w):
no edge of G enters s, so G_f has
no backward edge leaving s.

P never returns to s:
no other edge of P touches s.

> Only $f(s, w)$ changes, by $+b$:
> $\mathrm{val}(f') = \mathrm{val}(f) + b$, and $b > 0$.



$\mathrm{val}(f') = 30$

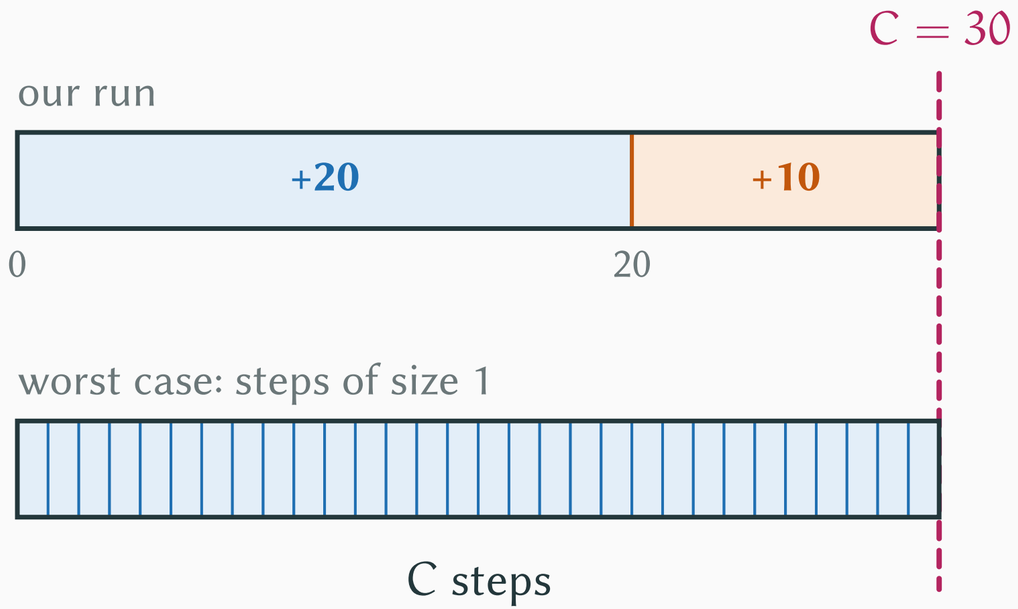
Termination

Invariant: every $f(e)$ and every residual capacity is an **integer**.

True for $f = 0$. Integers in, integers out:
$c_e - f(e)$ and $f(e)$, so b, so f'.

Each iteration raises $\mathrm{val}(f)$ by $b \geqslant 1$.

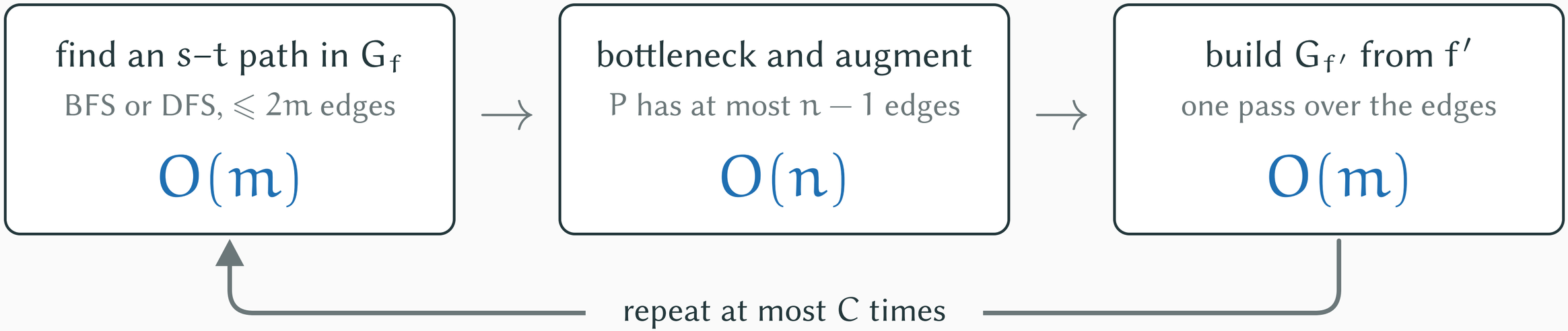
Always $\mathrm{val}(f) \leqslant C$, the capacity out of s.



At most C iterations: it terminates.

Irrational capacities: b can be tiny, and there is no such bound.
The algorithm may even run forever.

Running time



Theorem. Ford–Fulkerson runs in time $C \cdot O(m) = O(mC)$.

More precisely $O(mF)$, where $F \leqslant C$ is the value of a maximum flow.

What is an efficient algorithm?

Definition. An algorithm is **efficient** if its worst-case running time is **polynomial in the size N of the input**: $O(N^\alpha)$ for some constant α.

$O(\log N)$	$O(N)$	$O(N \log N)$	$O(N^2)$	$O(N^\alpha)$	$O(c^N)$
logarithmic	**linear**	**near-linear**	**quadratic**	**polynomial**	**exponential**
does not read all of the input	reads the input a constant number of times	splits in two halves, solves both, merges	looks at all pairs	many nested loops	looks at all subsets
efficient					not efficient

? *Is $O(mC)$ polynomial in the size of the input?*

Pseudo-polynomial running time

The size of a number is its number of **digits**.

$5 = 101_2$ needs 3 bits, but 11111 in unary needs 5.

Input: m capacities of k bits, so $N \approx m\,k$.
But $C \approx 2^k$ is possible.

bits per capacity k	**10**	**20**	**30**	**40**
input size $N \approx m\,k = 5k$	50	100	150	200
bound $C \approx 2^k$	10^3	10^6	10^9	10^{12}

$O(mC)$ is **pseudo-polynomial**: polynomial in the *values*, exponential in the number of bits.

? *Only a weakness of our analysis?*

Lecture 10: no. With 5 edges and $C = 200$, unlucky paths need 200 iterations.

Summary

1. **Flows.** Capacity and conservation; we want a flow of maximum value.
2. **Residual graph.** Forward edges $c_e - f(e)$, backward edges $f(e)$: flow can be undone.
3. **Ford–Fulkerson.** Augment along s–t paths of G_f by the bottleneck. Every step gives a flow, with value up by at least 1.
4. **Analysis.** At most C iterations of $O(m)$ time: $O(mC)$, pseudo-polynomial.

Can it be faster? Lecture 10.
Is the output a maximum flow? Lecture 11.

Reference: KT, Chapter 7.1; CLRS, Chapters 24.1, 24.2