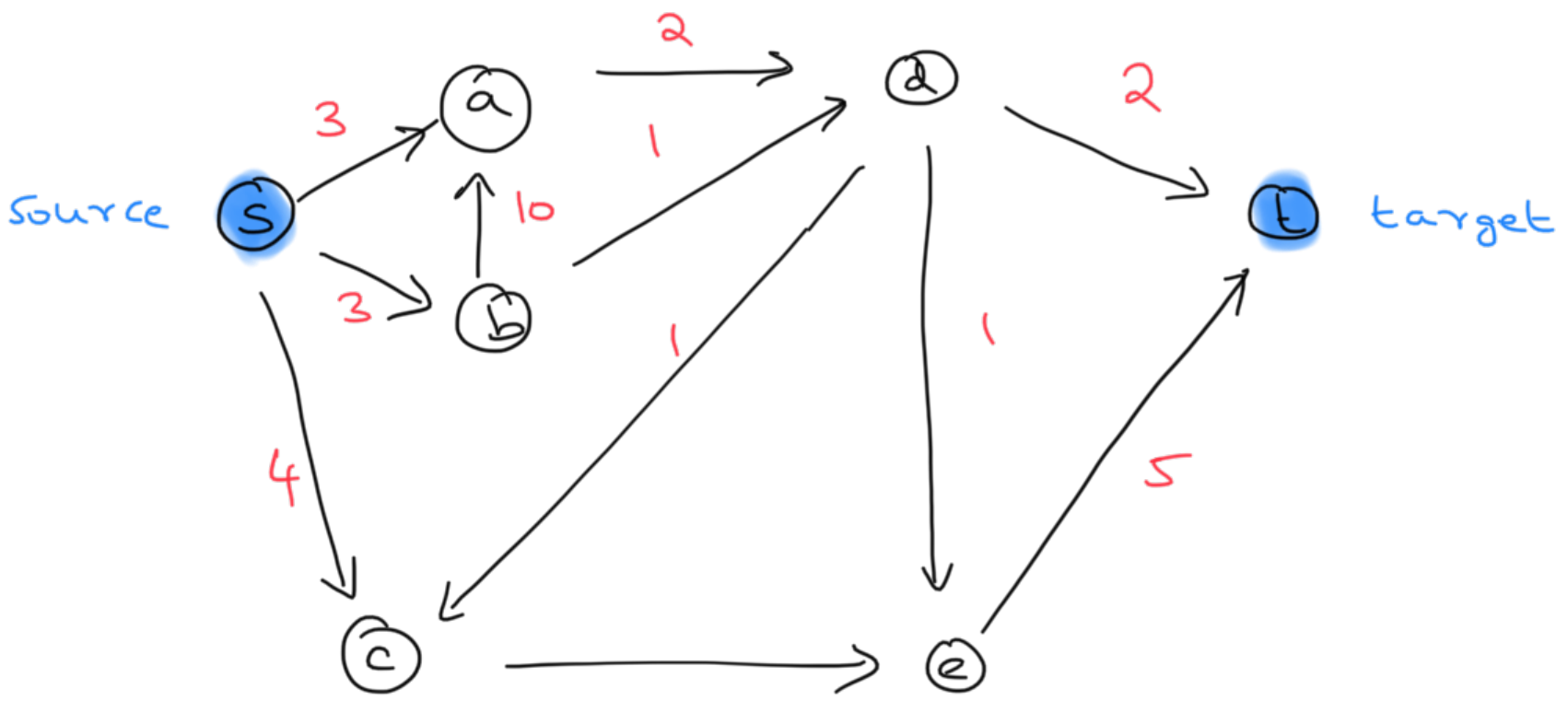
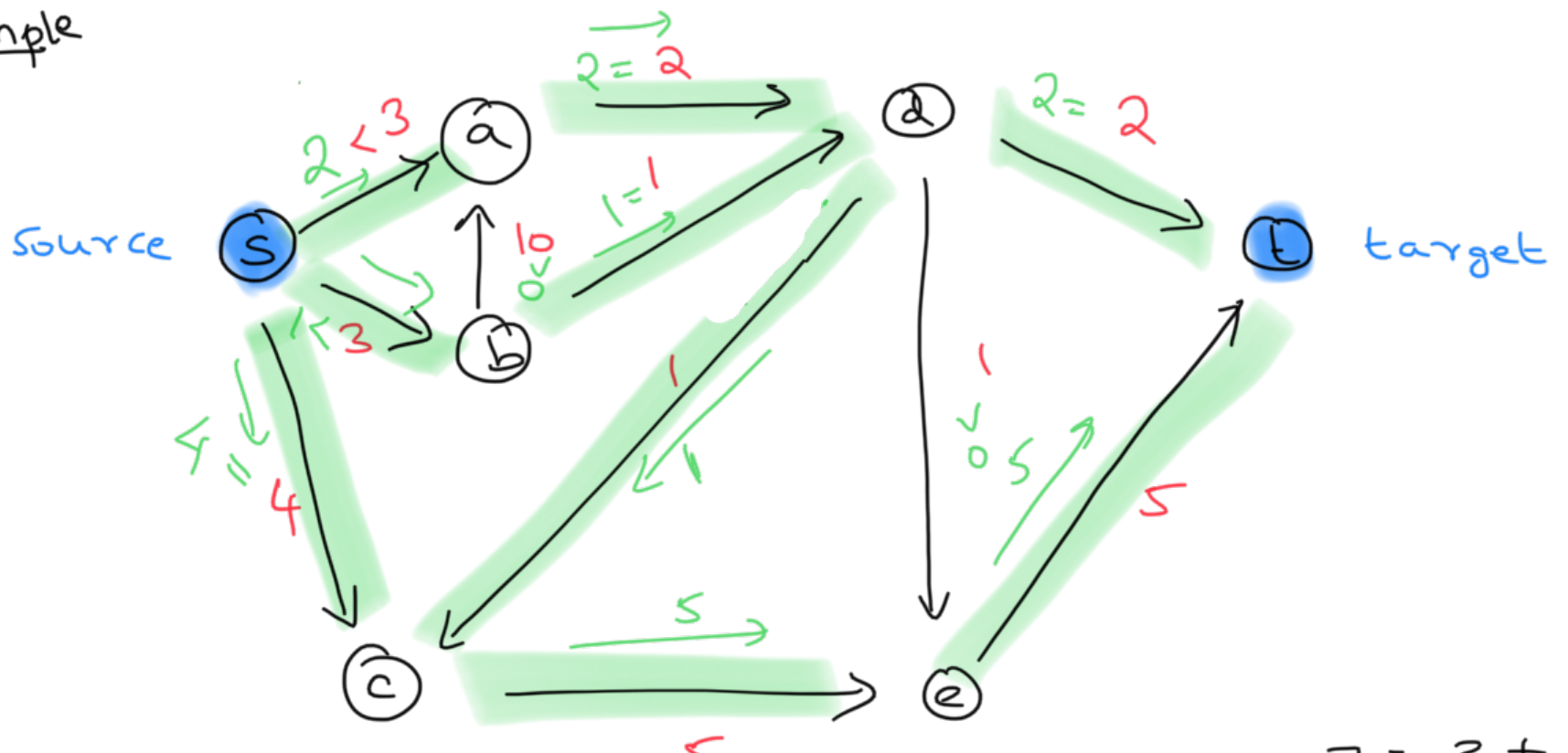
Linear Programming
Network flows



→ ship as much oil as possible from s to t

→ no storage on the way

example



can flow 7 units of oil

7 = 2 + 1 + 4 (source

= 5 + 2 (target

maximum flow ?

Formally

- Directed graph
- source s (no incoming edges)
- target t (no outgoing edges)
- each edge e has capacity c_e

Find flow f_e for each edge e so that

* $f_e \leq c_e$

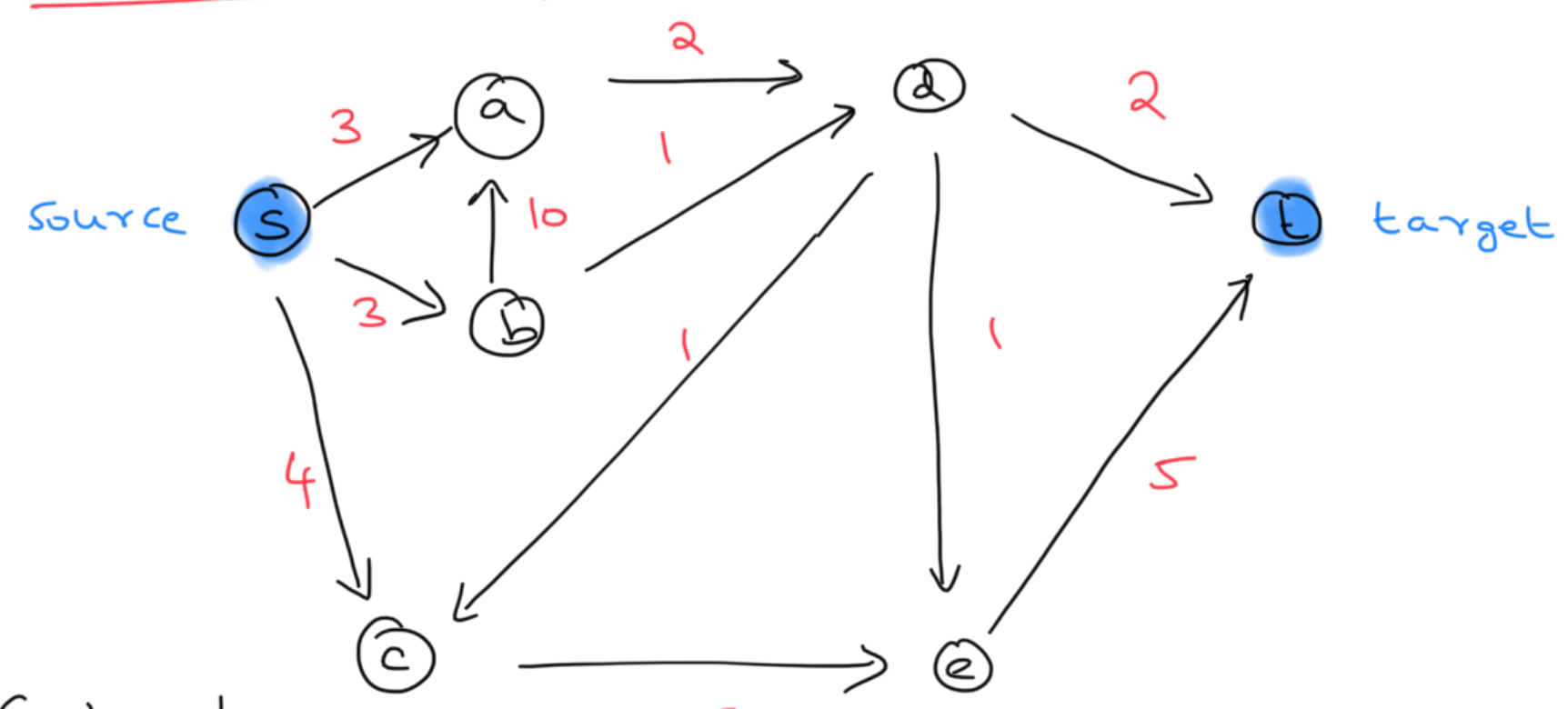
Conservation of flow:

* at each node $v \neq s, t$,

$\sum$ incoming flows to v = $\sum$ outgoing flows from v

Maximize Val of flow (:= $\sum$ outgoing flows from s = $\sum$ incoming flows to t)

LP for example



Constraints

① $f_{sa} \leq 3$, $f_{sb} \leq 3$, $f_{ba} \leq 10$, $f_{ad} \leq 2$, $f_{bd} \leq 1$

$f_{sc} \leq 4$, $f_{dc} \leq 1$, $f_{de} \leq 1$, $f_{ce} \leq 5$, $f_{dt} \leq 2$, $f_{et} \leq 5$

② $f_{sa} + f_{ba} = f_{ad}$ (conservation of flow at a)

⋮

maximize : $f_{sa} + f_{sb} + f_{sc}$

simplex method takes vertex in feasible region

→ moves to adjacent vertex if flow ↑

↳ can convert to direct algorithm for max flow

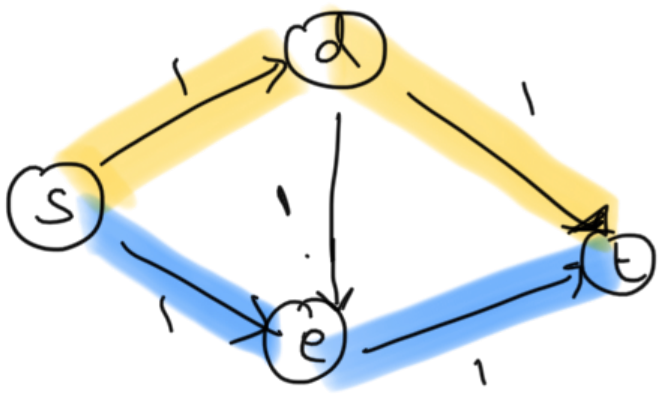
Ford - Fulkerson algorithm

→ Start with 0 flow

→ choose a path from s to t
where there is spare capacity.

→ Augment flow to saturate path

(eg)

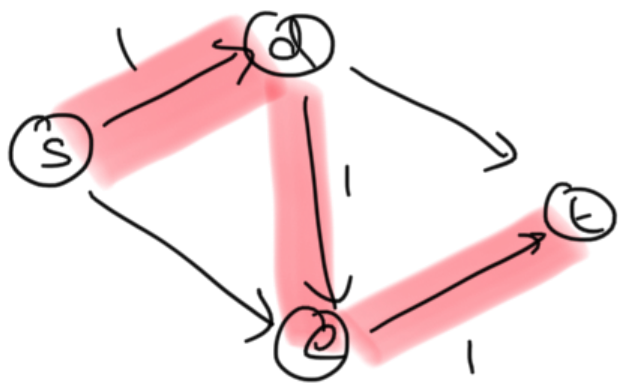


yellow path → 1

blue path → 1

max flow = 2

what if you start with a bad path?

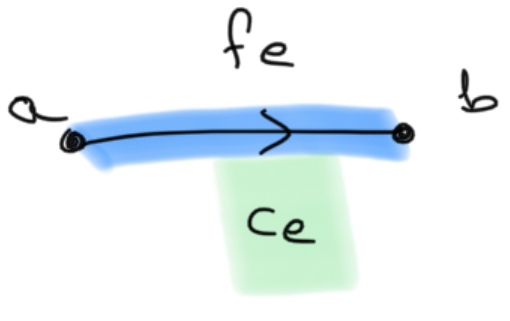


Red path → 1

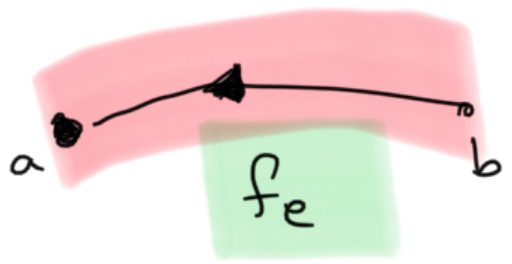
can't augment flow!

" need a process to reverse the flow in bad path"

Residual graph: For each edge e, current flow f_e capacity c_e

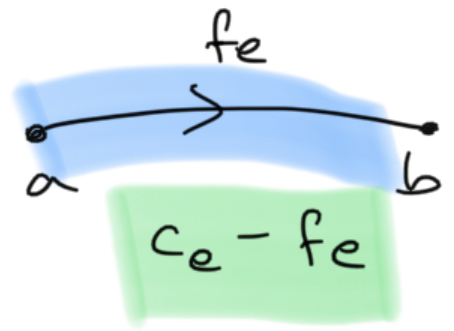


Add reverse edge



with capacity f_e

Reduce capacity of edge to $c_e - f_e$

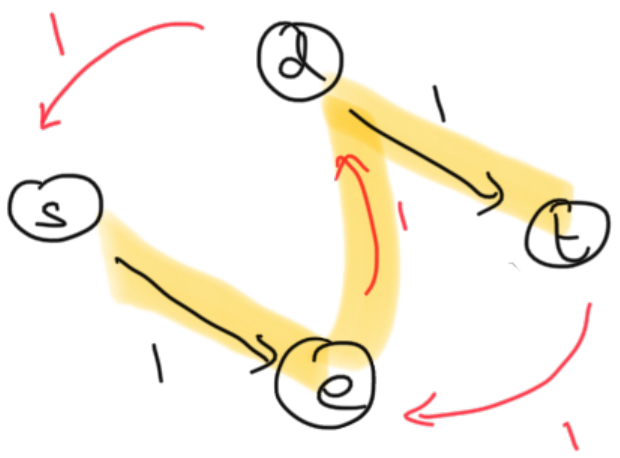


Dry Run

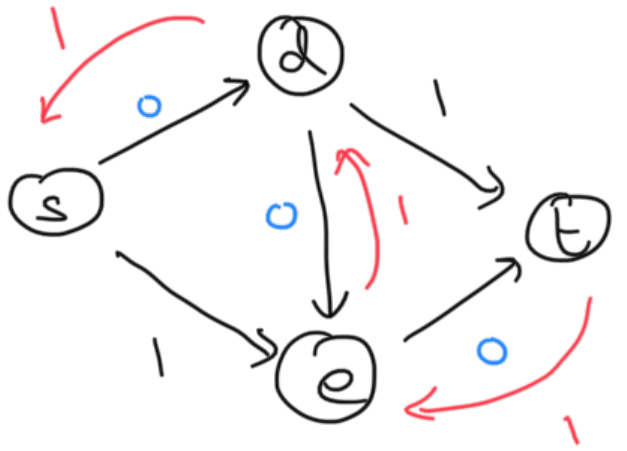
* flow on path

$s - d - e - t$: 1

maxflow = 1



=

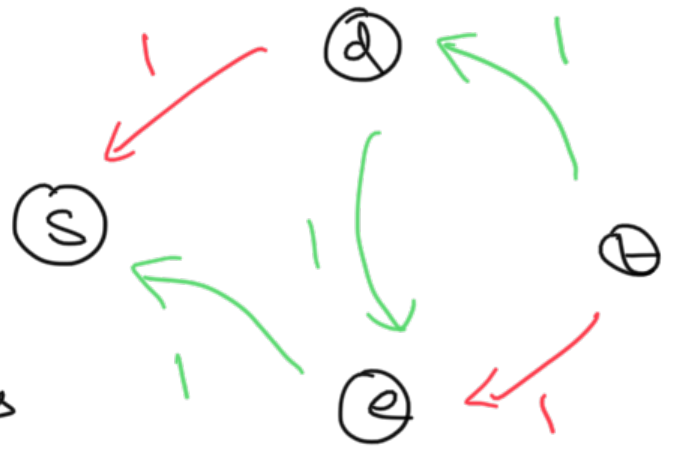


Residual graph for

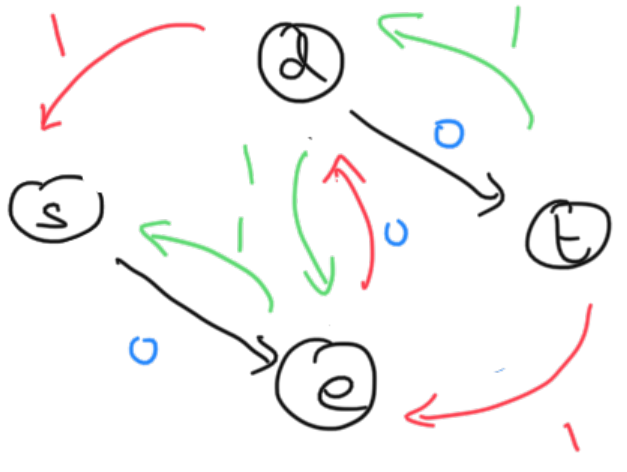
* flow on path

$s - e - d - t$: 1

& maxflow = 2



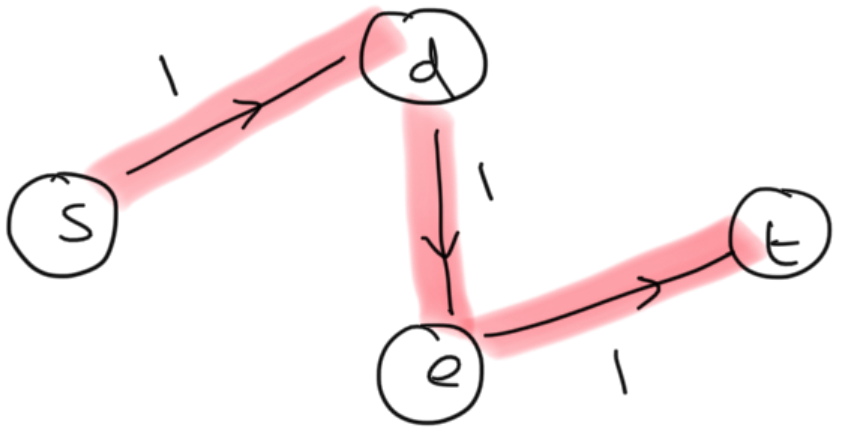
=



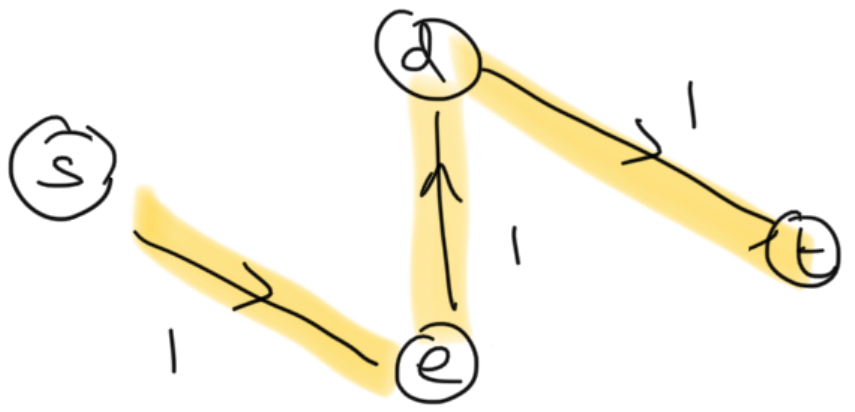
no more paths from s!
Terminate.

Residual graph for

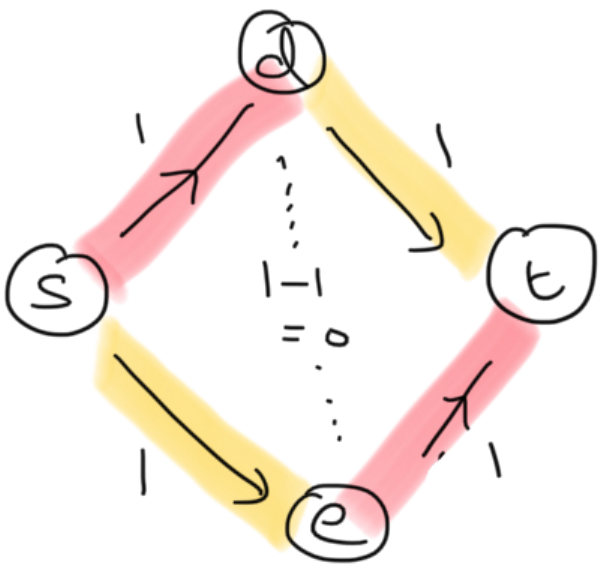
Max flow = 2 .



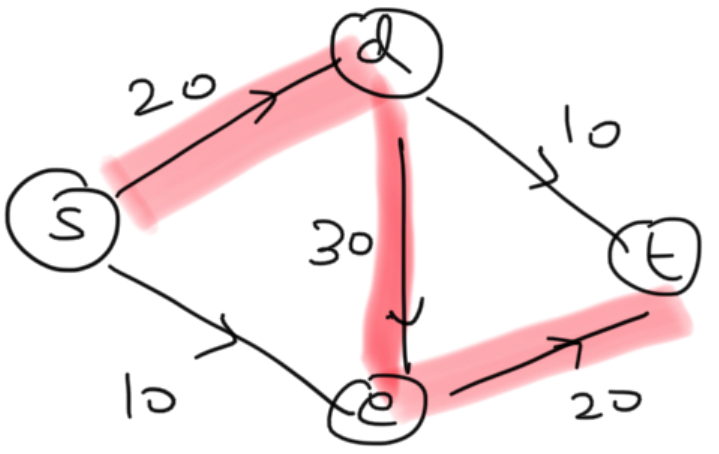
+



=

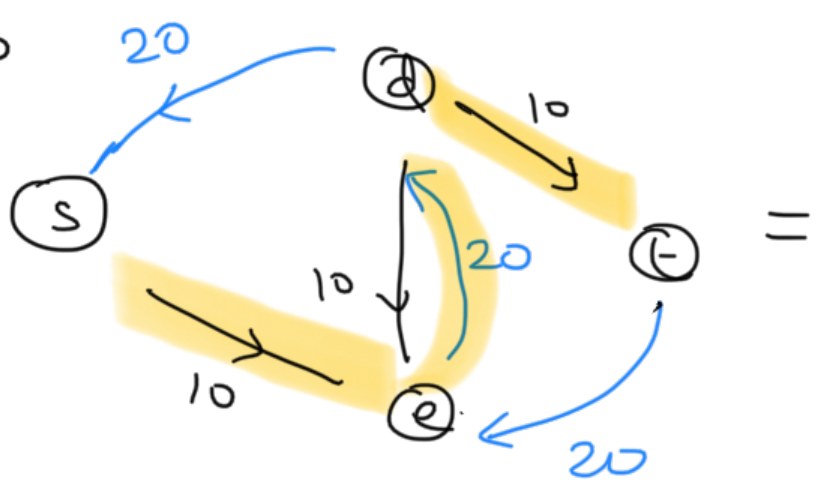


Another example

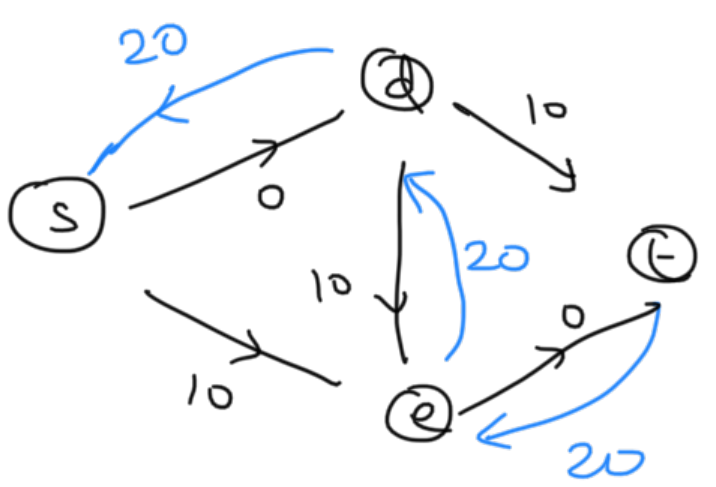


* flow on path s-d-e-t = 20
  max flow = 20

* flow on path s-e-d-t = 10
  max flow = 30



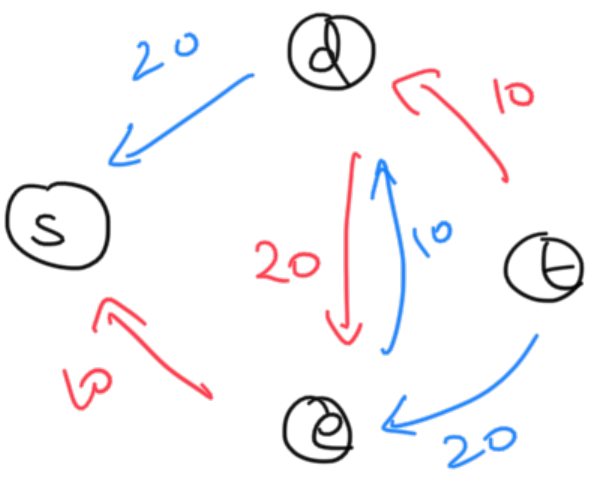
=



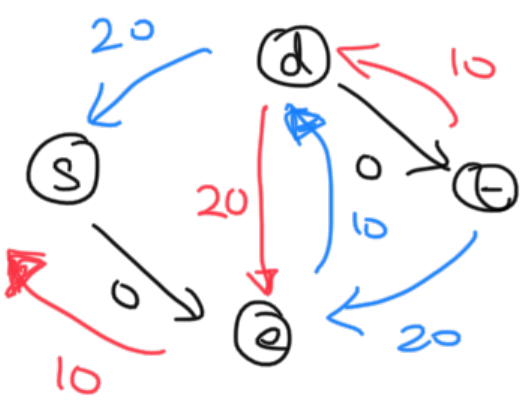
Residual graph for [redacted]

*

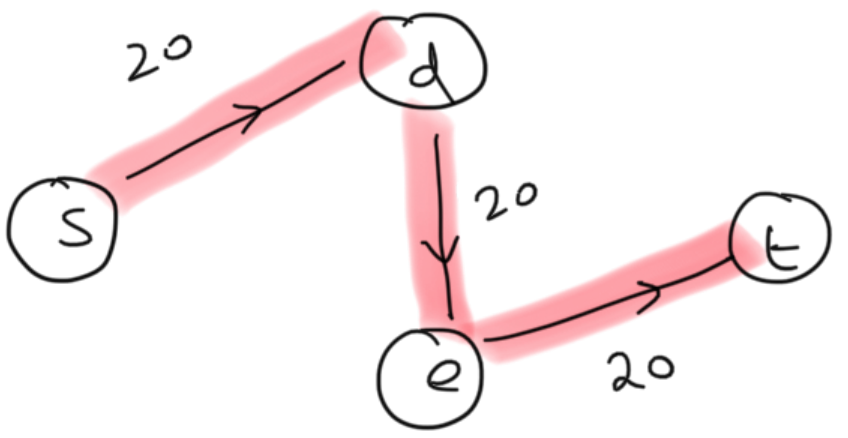
no more paths from s!
Terminate.



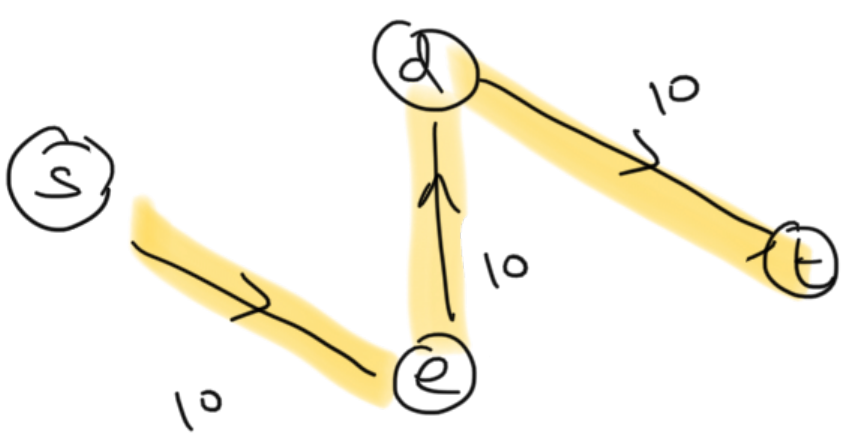
=



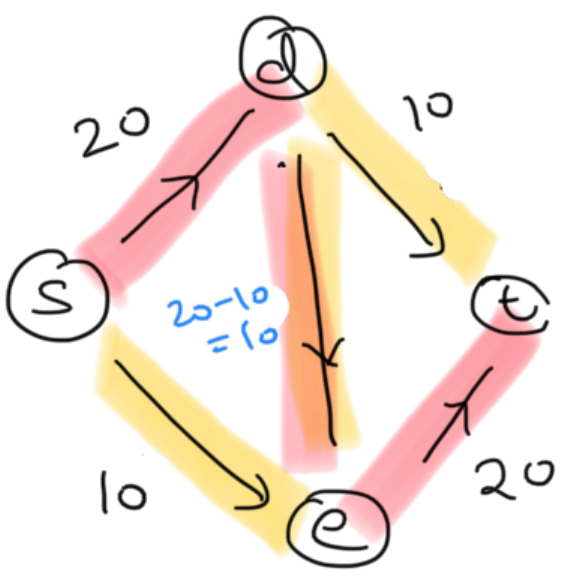
Max flow = 30



+



=



---

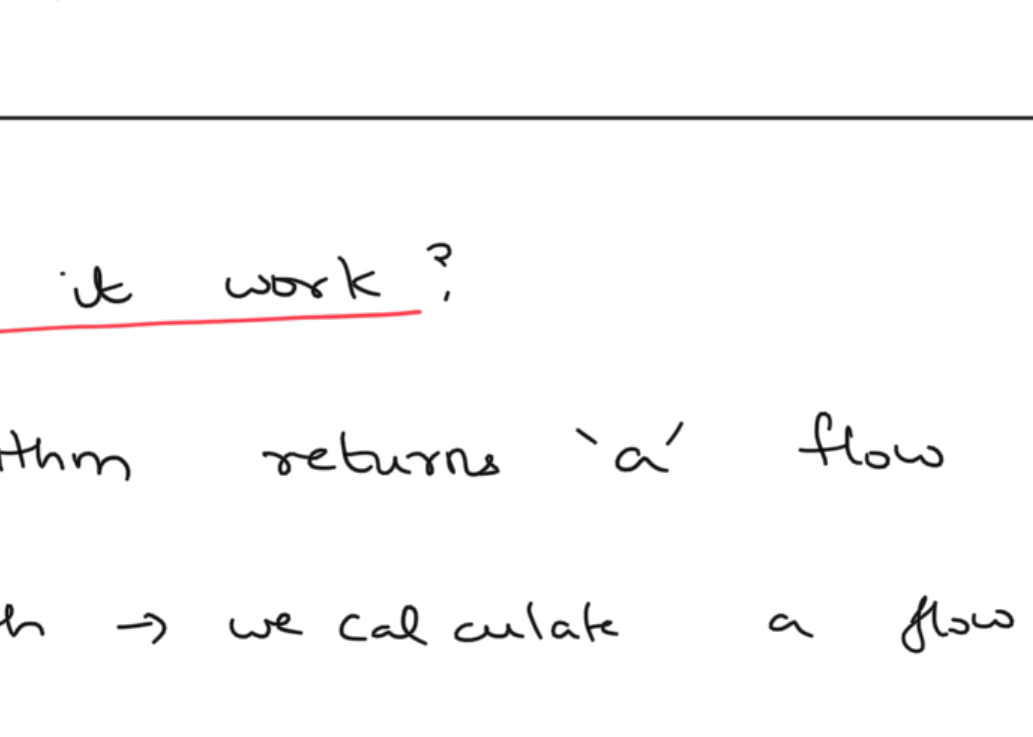
why does it work ?

→ The algorithm returns `a' flow because for each path → we calculate a flow

→ Summing it doesn't destroy conservation ∩ flow.

→ why optimal ?

A Cut :

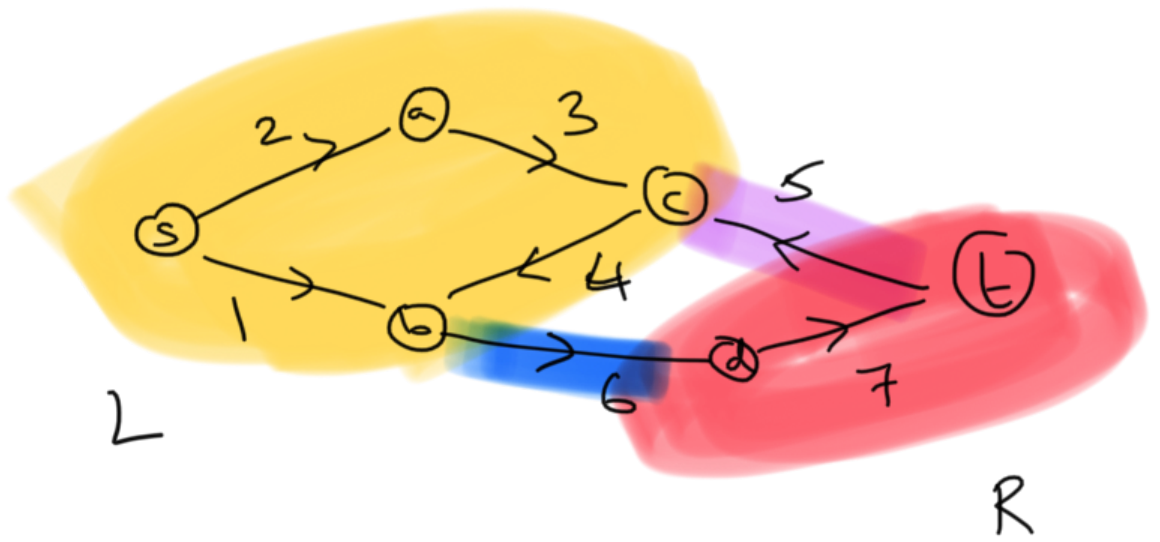


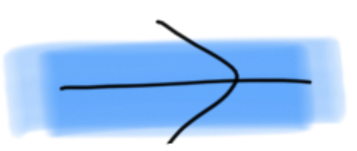
Partition ∩ vertices

$L \amalg R$ = vertices ∩ graph, $s \in L$, $t \in R$

weight (cut) = sum ∩ edges leaving L.

(eg)



weight of cut = 6 

Note flow in a network $\leq$ wt of any cut

So max flow in a network $\leq$ wt of any cut

So max flow in a network $\leq$ min cut weight

Lemma: Let f be a flow in a network and $(A, \bar{B})$ be a cut. So $B = V \setminus A$.

Define $\text{netflow}(A, \bar{B}) := \sum_{\substack{e \text{ edge} \\ \text{out of } A}} f_e - \sum_{\substack{e' \text{ edge} \\ \text{into } A}} f_{e'}$

Then $|f| = \text{flow value} = \text{netflow}(A, \bar{B})$

Pf

$$|f| = \sum_{\substack{e'' \text{ out} \\ \text{of } s}} f_{e''}$$

$$= \sum_{\substack{e'' \text{ out} \\ \text{of } s}} f_{e''} - \sum_{\substack{e''' \text{ into} \\ s}} f_{e'''}$$

0 as no edge into s!

$$= \sum_{v \in A} \left(\sum_{\substack{e \text{ out} \\ \text{of } v}} f_e - \sum_{\substack{e' \text{ into} \\ v}} f_{e'} \right)$$

as $s \in A$ and flow conservation

$$= \sum_{\substack{e \text{ out} \\ \text{of } A}} f_e - \sum_{\substack{e' \text{ into} \\ A}} f_{e'} = \text{netflow}(A, \bar{B})$$

Max flow - min cut theorem

max-flow = min cut !

Pf Claim: TFAE

① f is max flow

② No augmenting path relative to f (in residual graph)

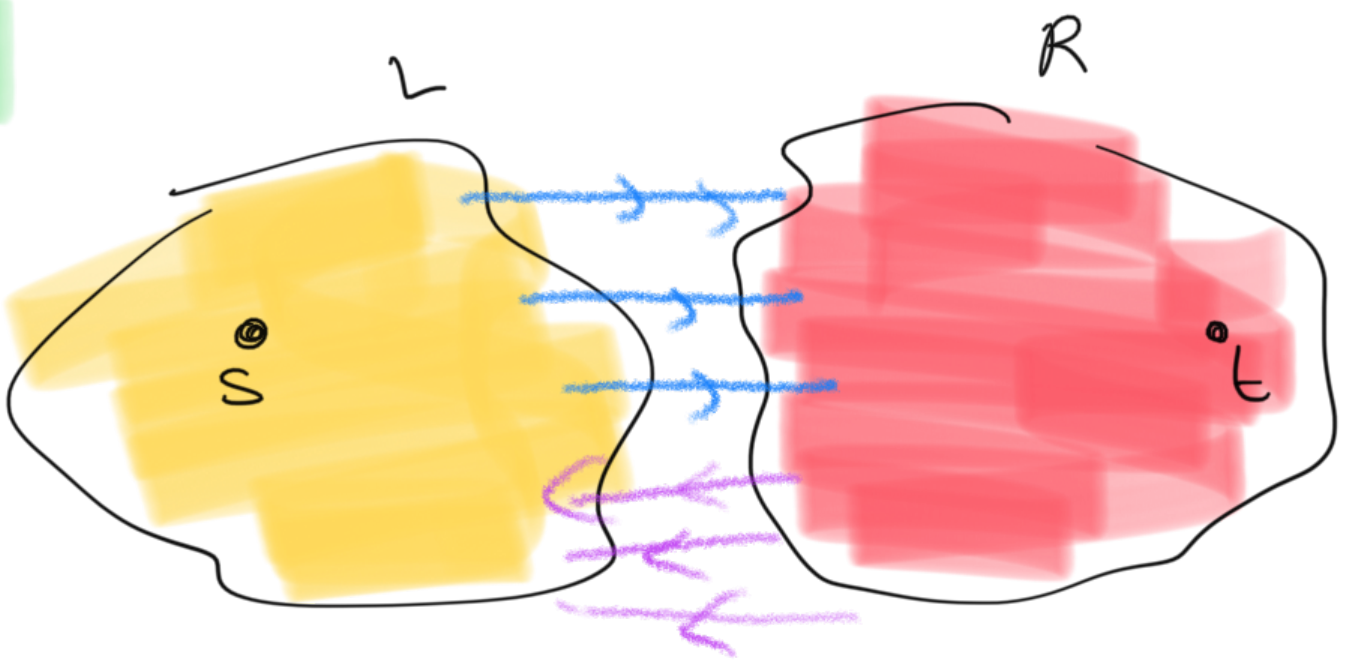
③ ∃ cut C with weight = flow(f)

Note that claim's ① ⇒ ③ + max flow ≤ min cut shows max flow = min cut

---

Pf of ② ⇒ ③

Residual graph : G_f



L = set of vertices reachable from s in residual graph
So $s \in L$

R = $V \setminus L$

since no augmenting path from s to t, $t \notin L$ and so $t \in R$. By def $L \cap R = \emptyset$ and $L \cup R = V$

So (L, R) = cut

Let G be original graph

Claim: ⓐ Any edge in original graph G, leaving L has full current flow

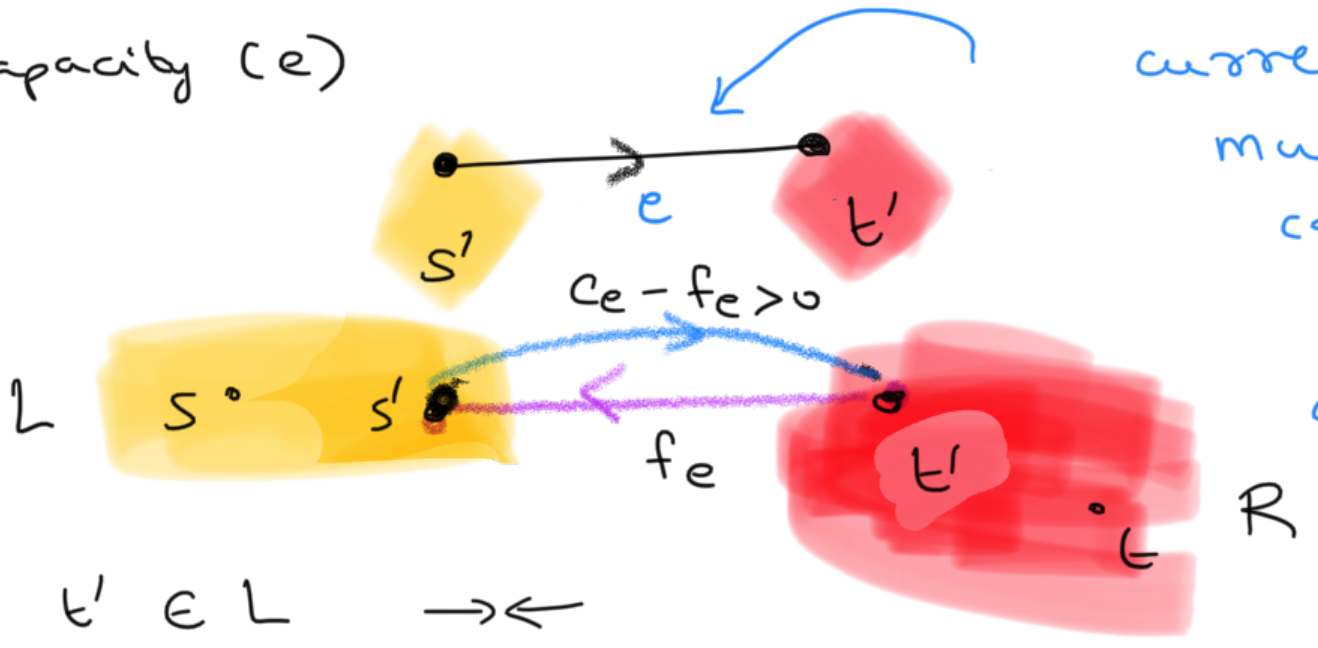
ⓑ any edge in original graph entering L has 0 current flow

Pf

ⓐ Look at edge leaving L in ORIGINAL GRAPH

f_e = current flow (e)

c_e = capacity (e)



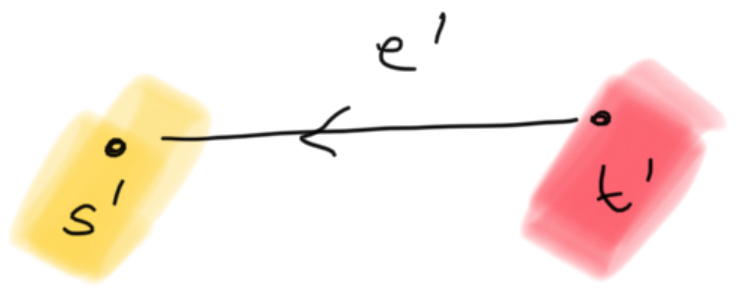
currently must be at full capacity, because if not, $f_e < c_e$ and in residual graph

So $t' \in L$ →←

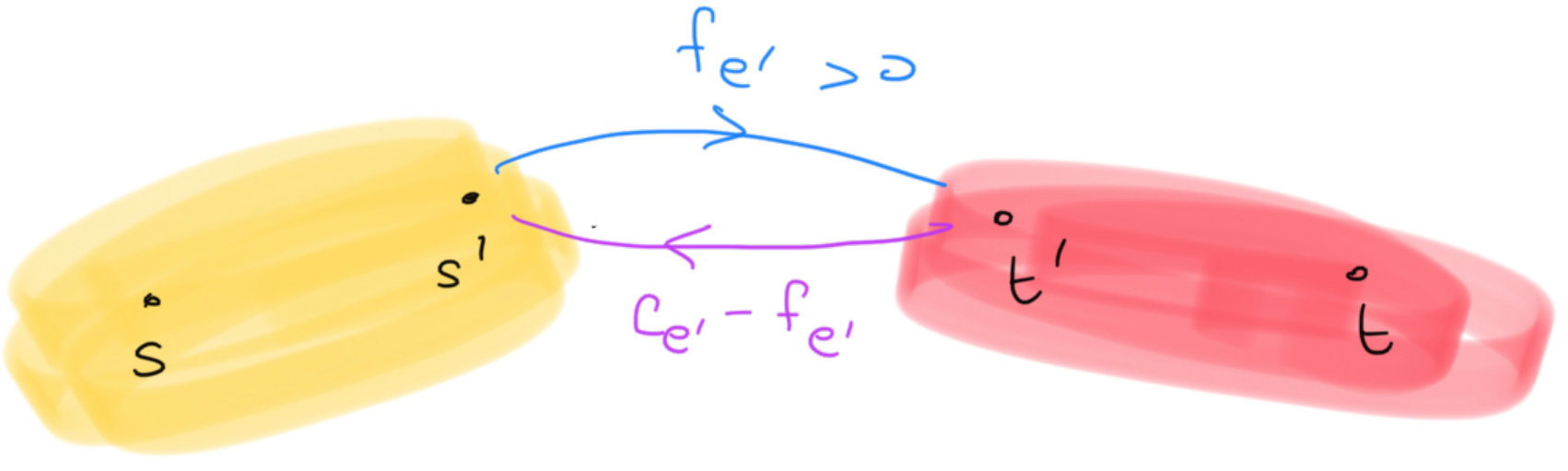
ⓑ Similarly look at edge e' entering L in ORIGINAL GRAPH G

$f_{e'}$ = current flow (e')

$c_{e'}$ = capacity (e')

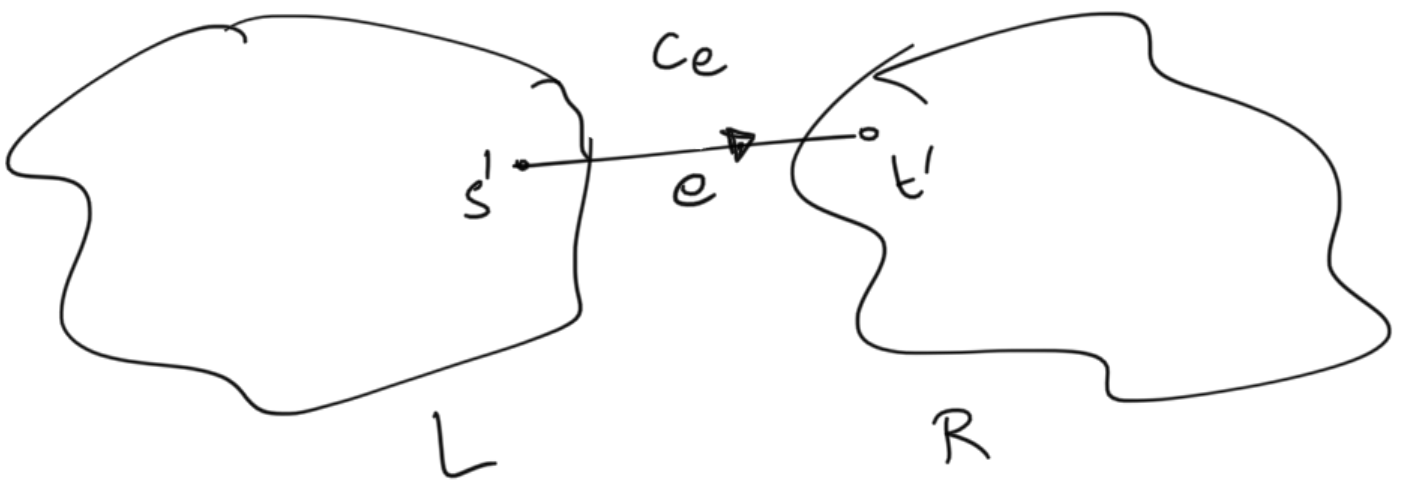


must have 0 current flow, because if not $f_{e'} > 0$ and in residual graph



So $t' \in L$ →←

This cut (L, R) has weight

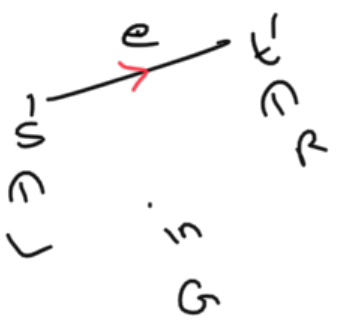


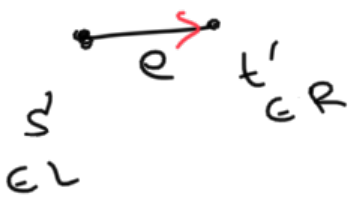
$$= \sum_{e \text{ edge out of } S} f_e - \sum_{e' \text{ edge into } S} f_{e'}$$

$$= \text{Net flow}(S, V \setminus S) = |f|$$

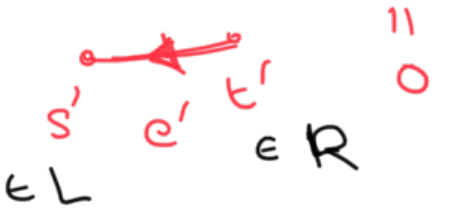
By Lemma!

$\sum c_e \geq \sum f_e$





$+ \sum f_{e'}$, e' in G



Pf of (3) ⇒ (1) ∃ cut C with wt = flow(f)

But flow(f) ≤ wt (any cut!) (see for yourself why)

So if flow(f) = wt (cut C), then flow(f) can't be increased

Pf of (1) ⇒ (2) if ∃ augmenting path p',

max flow = flow(f) + flow(p') > flow(f)

So flow(f) can't be max.
