COMP251: Network flows (1)

Jérôme Waldispühl & Giulia Alberini
School of Computer Science
McGill University

Based on slides from M. Langer (McGill) & (Cormen *et al.*, 2009)

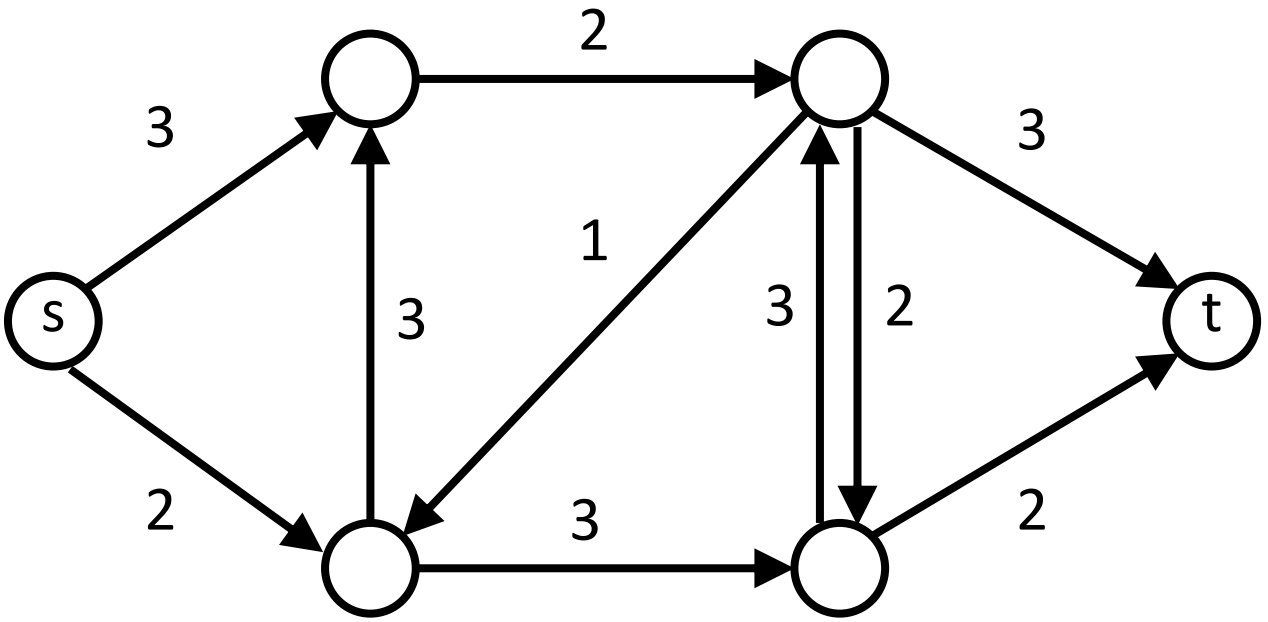
Flow Network

$G = (V, E)$ directed.

Each edge (u, v) has a ***capacity*** $c(u, v) \geq 0$.

If $(u,v) \notin E$, then $c(u,v) = 0$. If there is no edge, there is no capacity

Source vertex s, ***sink*** vertex t, assume $s \rightsquigarrow v \rightsquigarrow t$ for all $v \in V$.



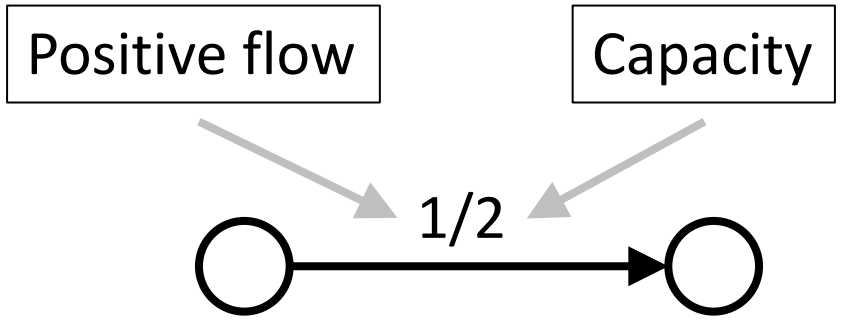
The flow is flowing from the source to the sink

Definitions

Each pair of vertices gets a value in R

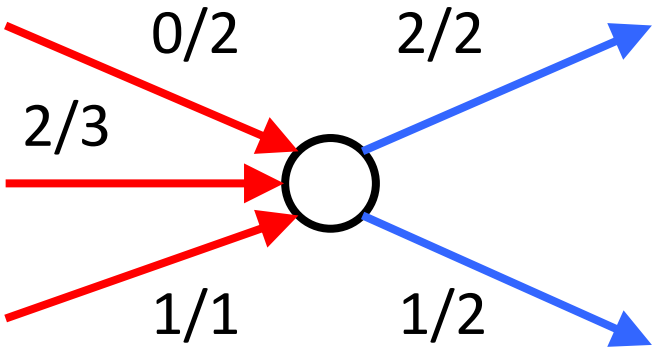
Positive flow: A function p : V × V → **R** satisfying.

Capacity constraint: For all u, v ∈ V, 0 ≤ p(u, v) ≤ c(u, v)



Flow conservation: For all u ∈ V – {s, t},

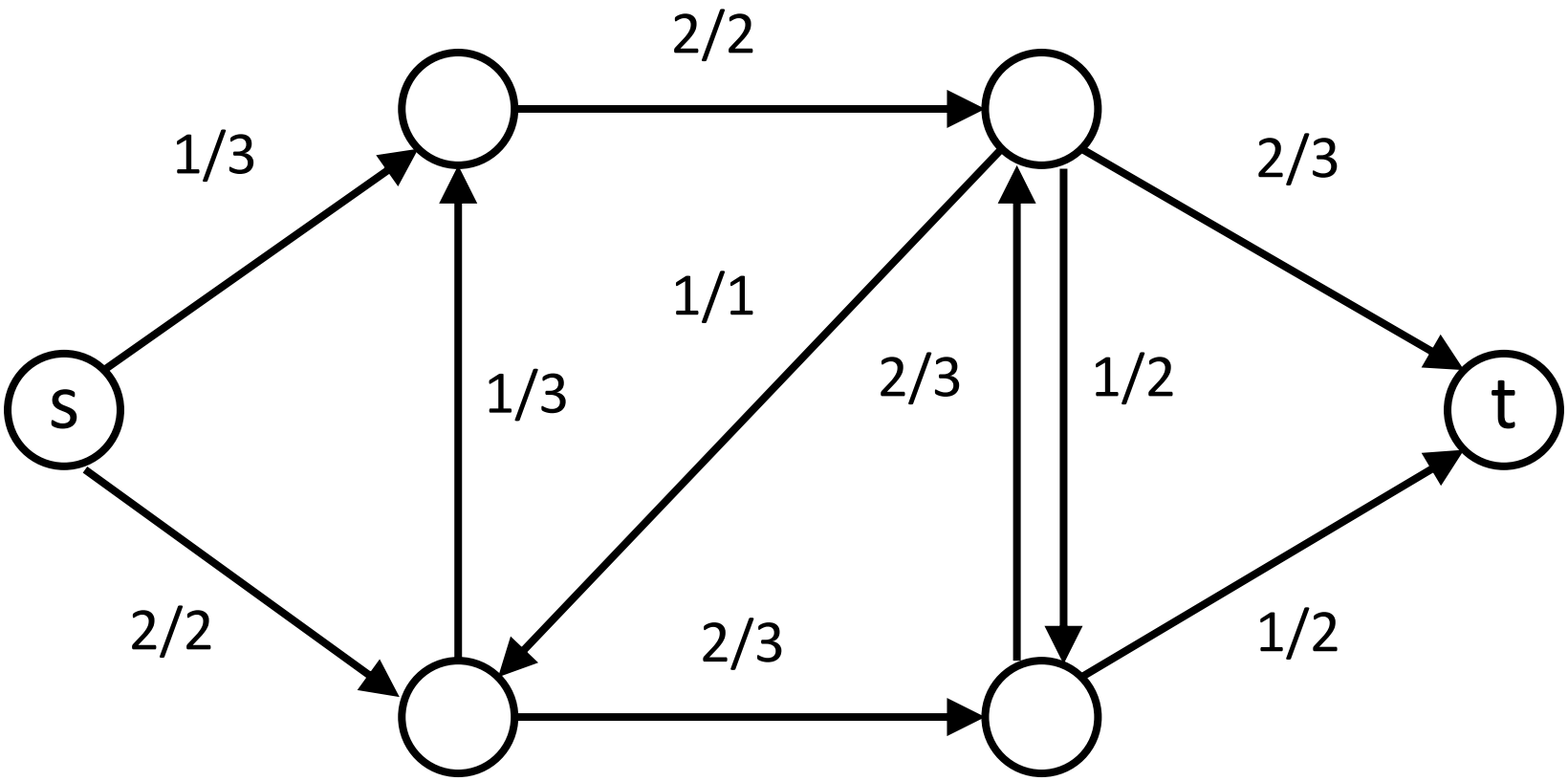
$$\underbrace{\sum_{v \in V} p(v,u)}_{\text{Flow into u}} = \underbrace{\sum_{v \in V} p(u,v)}_{\text{Flow out of u}}$$



Flow in: 0 + 2 + 1 = 3

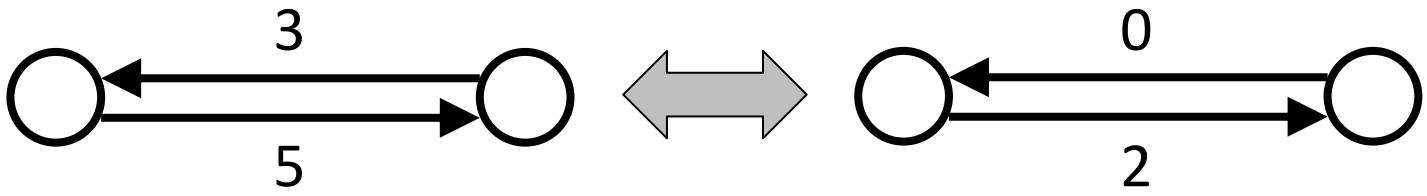
Flow out: 2 + 1 = 3

Example



Cancellation with positive flows

- Without loss of generality, can say positive flow goes either from *u* to v or from v to *u*, but not both.
- In the example below, we can "cancel" 3 units of flow in each direction between the vertices.



- Capacity constraint is still satisfied.
- Flow conservation is still satisfied.

Logic: think about water in a river. Some of the water flowing in, in the opposite direction to the current would not lead to water going in both directions. The current would keep in direction and the water would slow down.

Net flow

A function $f : V \times V \rightarrow \mathbf{R}$ satisfying:

- ***Capacity constraint:*** For all $u, v \in V$, $f(u, v) \leq c(u, v)$

  The flow respects the capacity of each edge
- ***Skew symmetry:*** For all $u, v \in V$, $f(u, v) = -f(v, u)$
- ***Flow conservation:*** For all $u \in V - \{s, t\}$, $\sum_{v \in V} f(u, v) = 0$

$$\underbrace{\sum_{v \in V; f(v,u) > 0} f(v,u)}_{\text{Total positive flow entering u}} = \underbrace{\sum_{v \in V; f(u,v) > 0} f(u,v)}_{\text{Total positive flow leaving u}}$$

Except the source and the sink, the flow entering a vertex is equal to the flow leaving a vertex

Positive vs. Net flows

Define net flow in terms of positive flow:

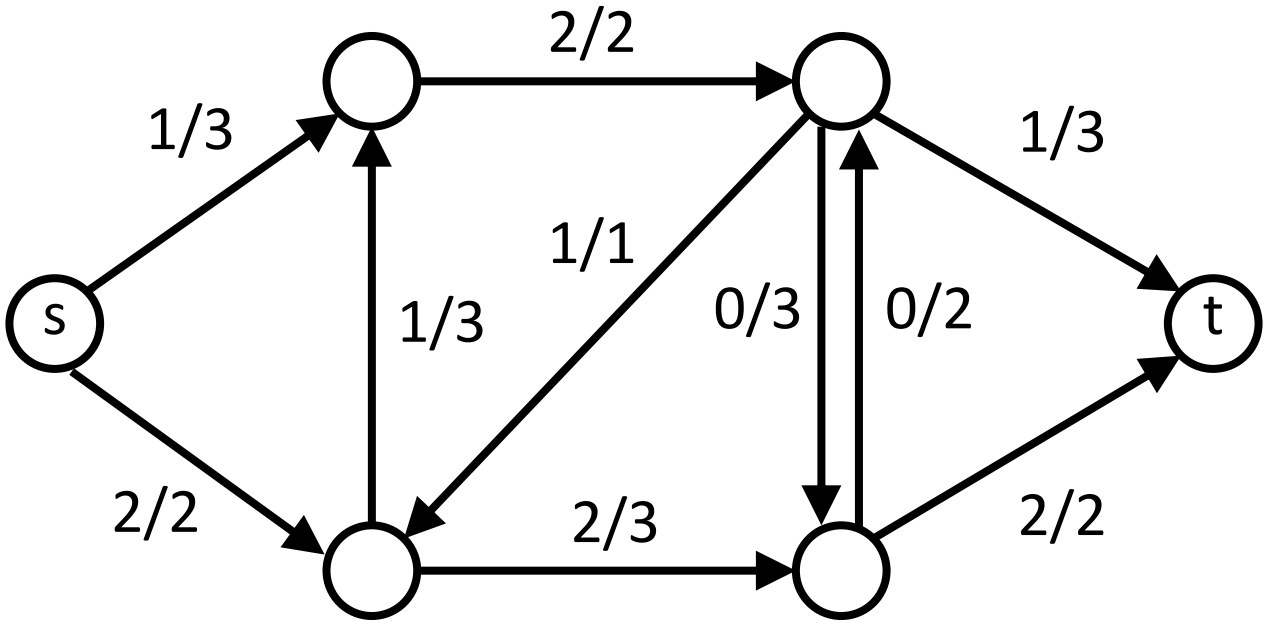
$$f(u,v) = p(u,v) - p(v,u).$$

The differences between positive flow p and net flow f :

- $p(u,v) \geq 0$,
- f satisfies skew symmetry.

Values of flows

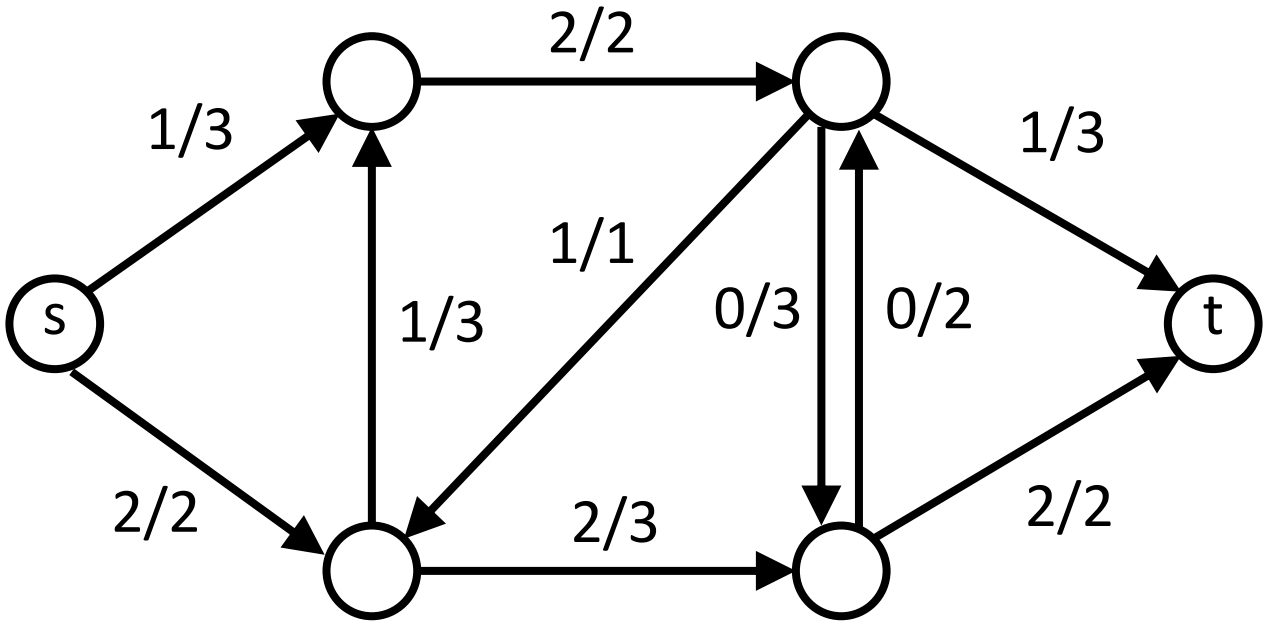
Definition: $f = |f| = \sum_{v \in V} f(s, v)$ = total flow out of source.



Value of flow $f = |f| = 3$.

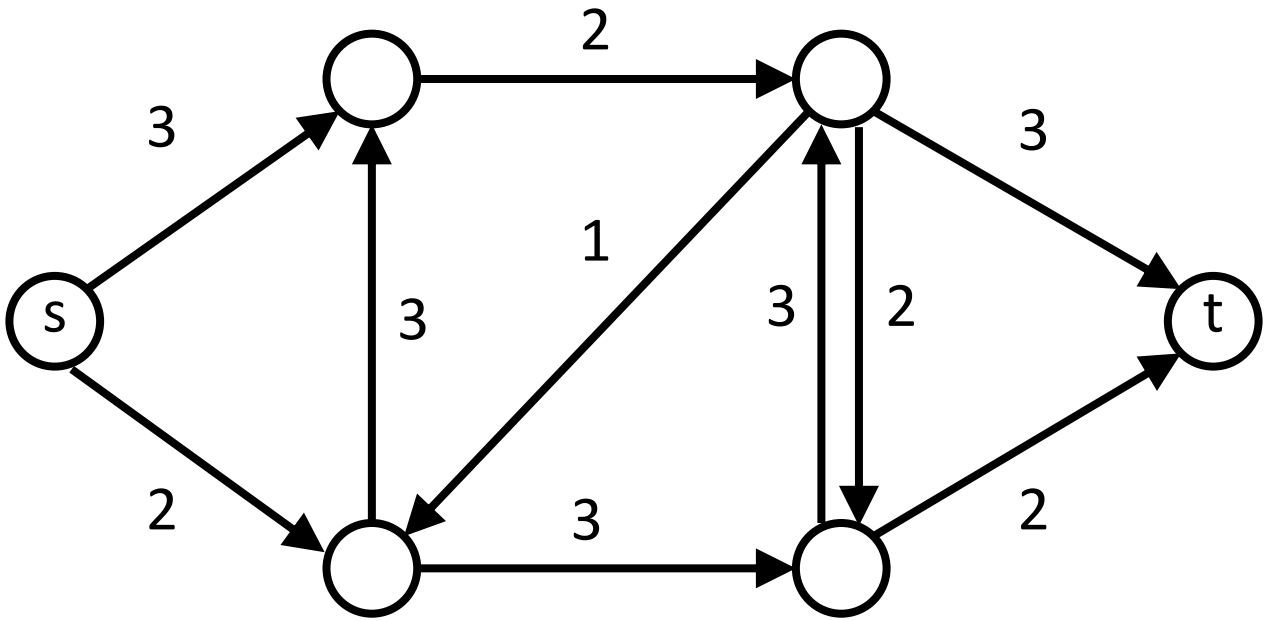
Flow properties

- Flow in == Flow out
- Source *s* has outgoing flow
- Sink *t* has ingoing flow
- Flow out of source *s* == Flow in the sink *t*

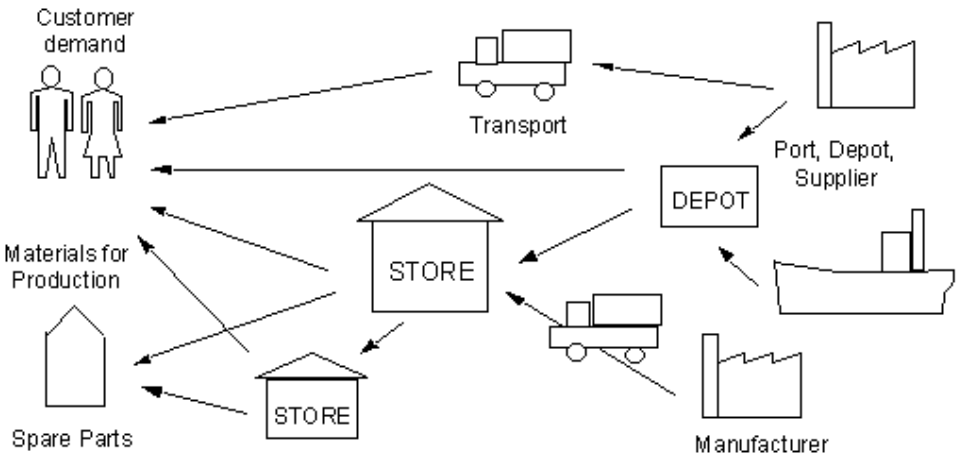


Maximum-flow problem

Given *G*, *s*, *t*, and *c*, find a flow whose value is maximum.



Applications



(https://ais.web.cern.ch/ais/)

(http://driverlayer.com)

Naïve algorithm

```
Initialize f = 0
While true {
   if (∃ path P from s to t such that all
edges have a flow less than capacity)
   then
      increase flow on P up to max capacity
   else
      break
}
```

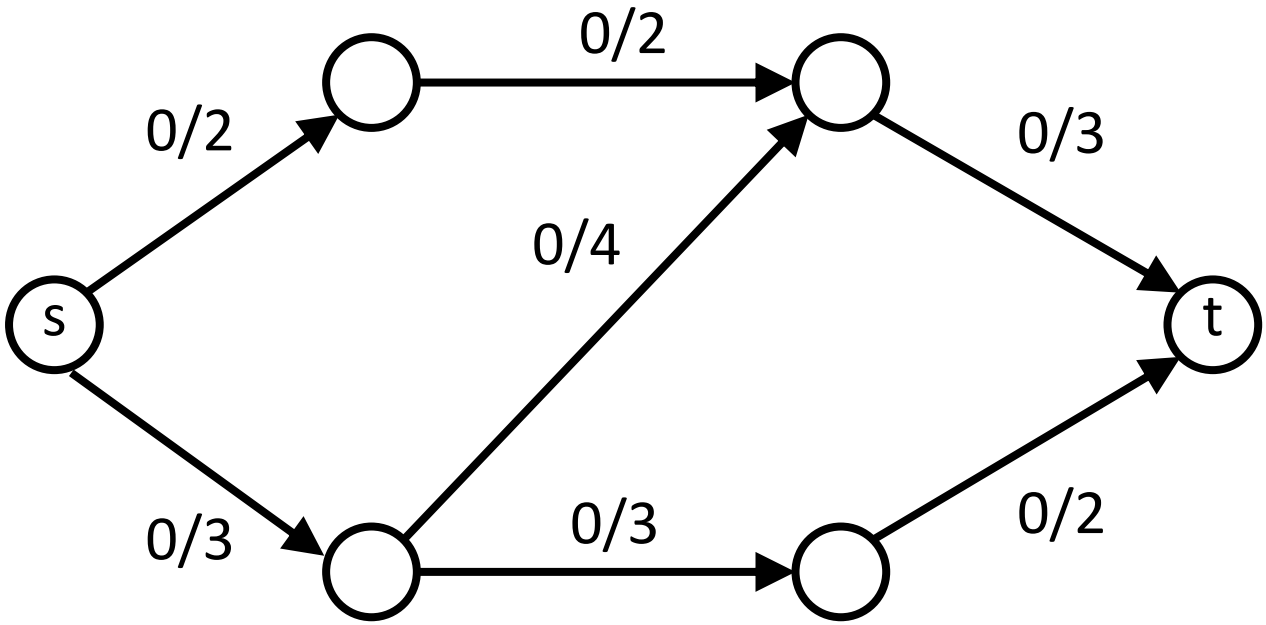
Naïve algorithm

```
Initialize f = 0
While true {
   if (∃ a path P from s to t  s.t. all
edges e ∈ P f(e) < c(e) )
   then {
      β = min{ c(e)-f(e) | e ∈ P}
      for all e ∈ P { f(e) += β }
   } else { break }
}
```

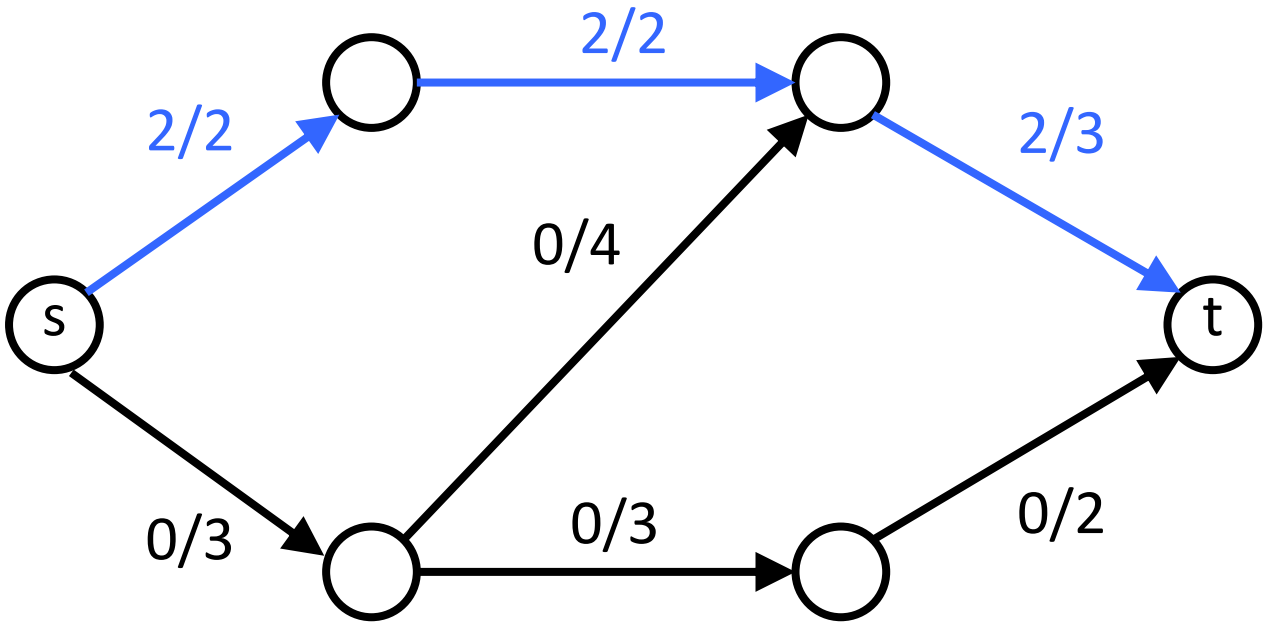
Find the path

Identify the minimum difference between capacity and flow and increment every edge's flow by that much.

Example where algorithm works

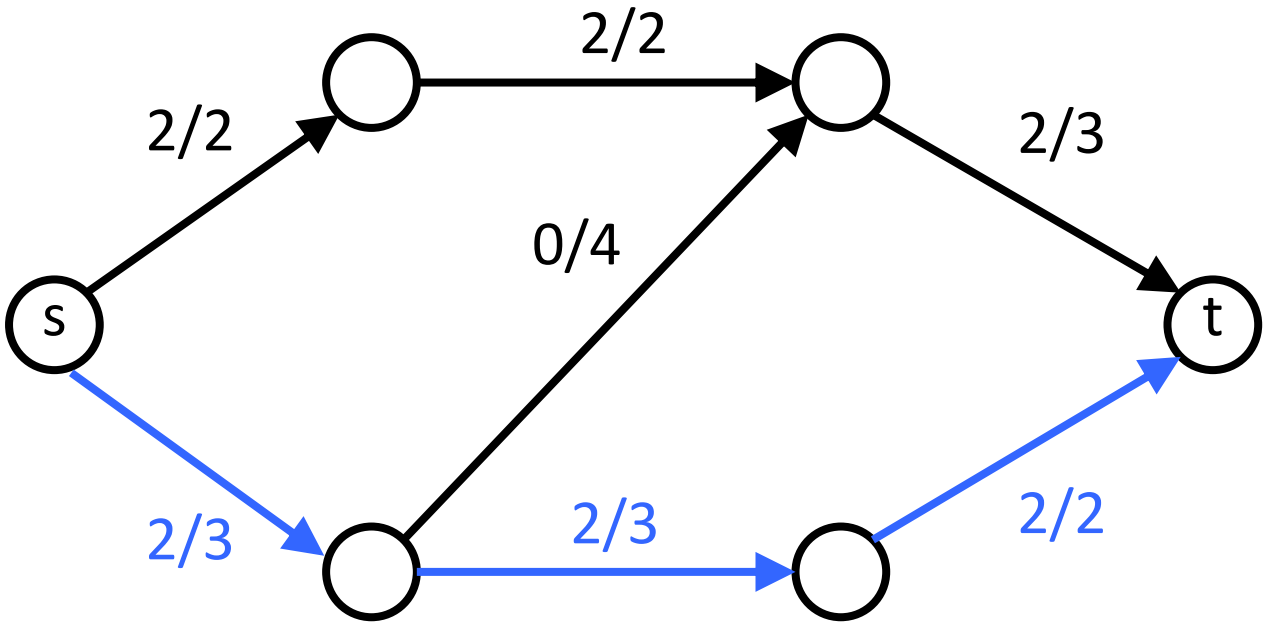


Example where algorithm works



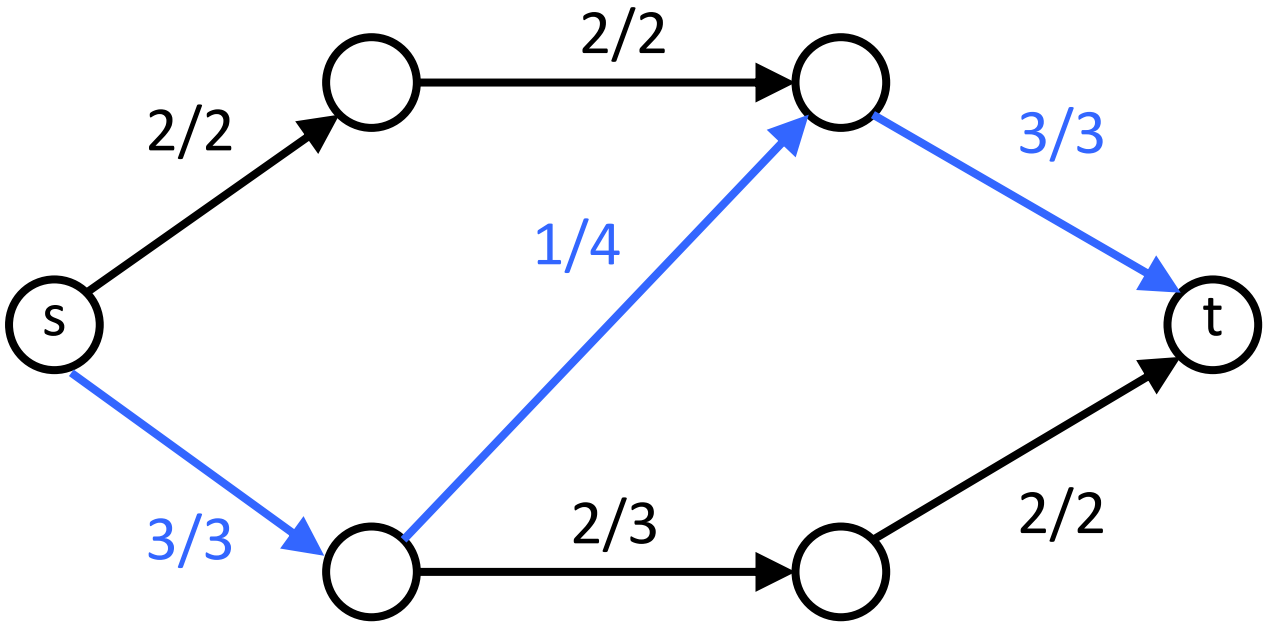
$|f|=2$

Example where algorithm works



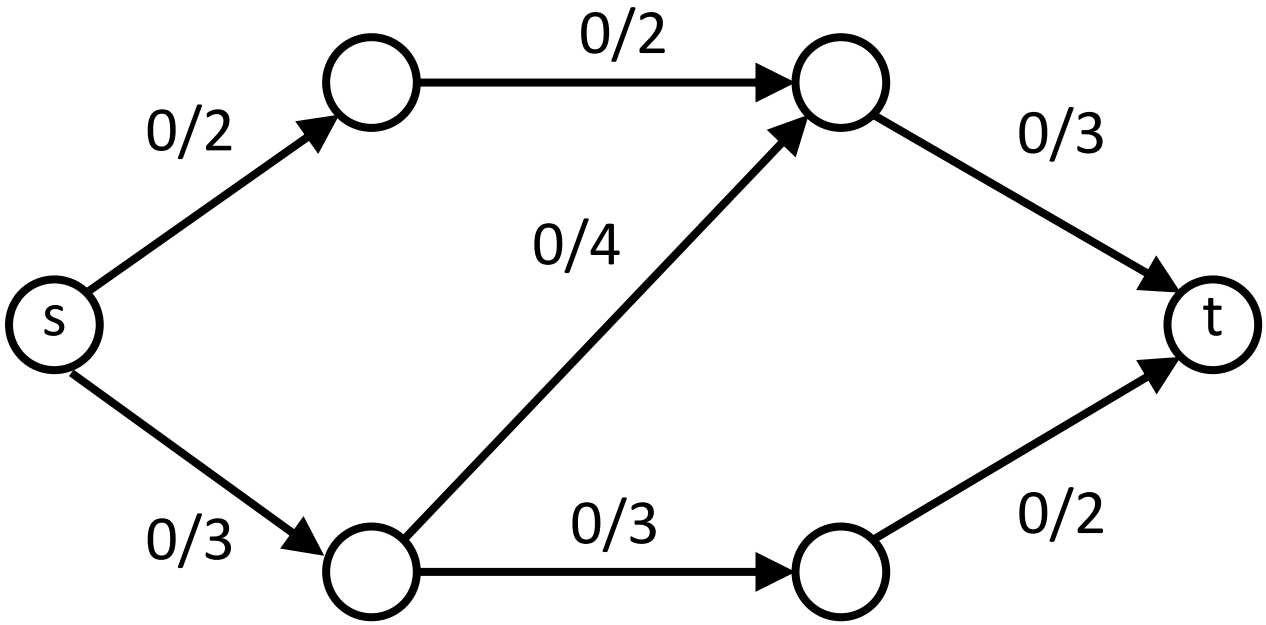
$|f|=4$

Example where algorithm works

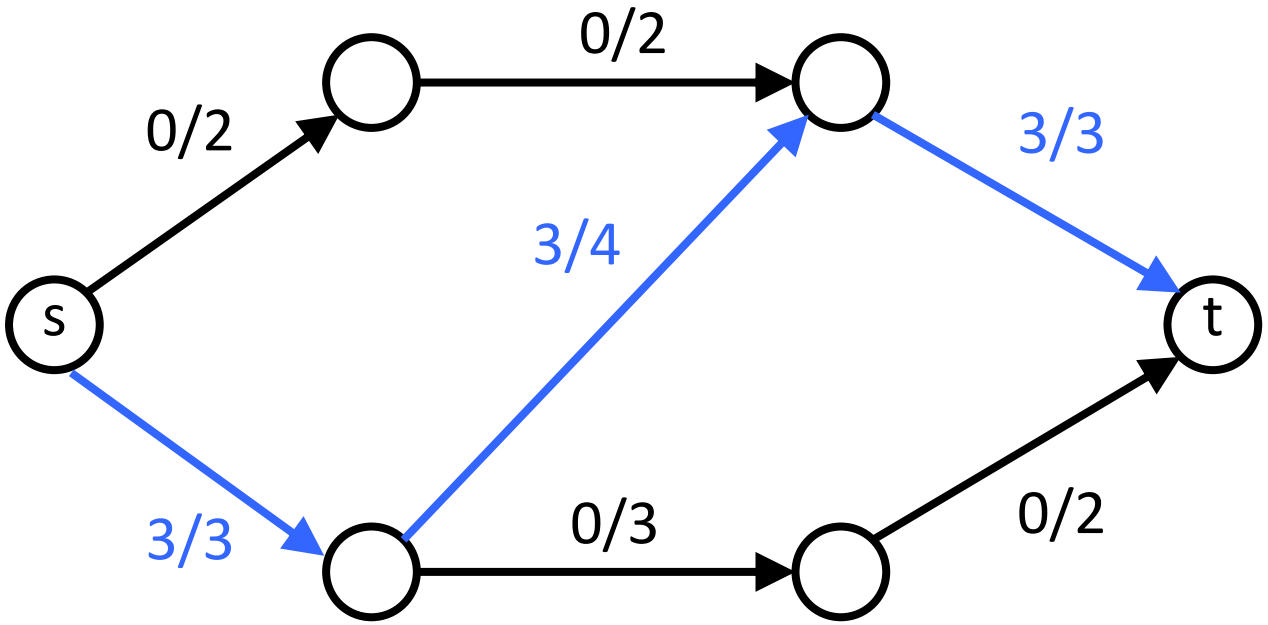


$|f|=5$

Example where algorithm fail!



Example where algorithm fail!



|f|=3 And terminates…

Challenges

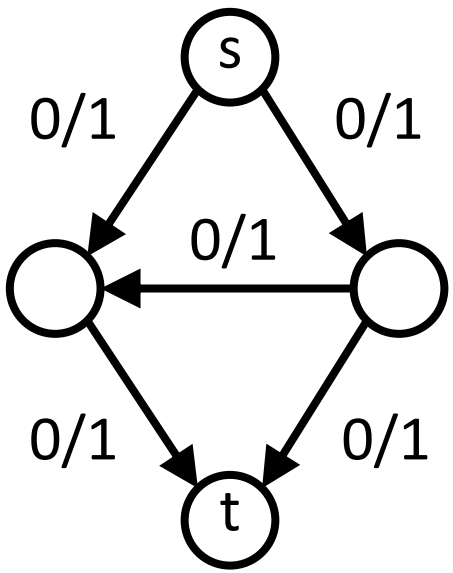
How to choose paths such that:

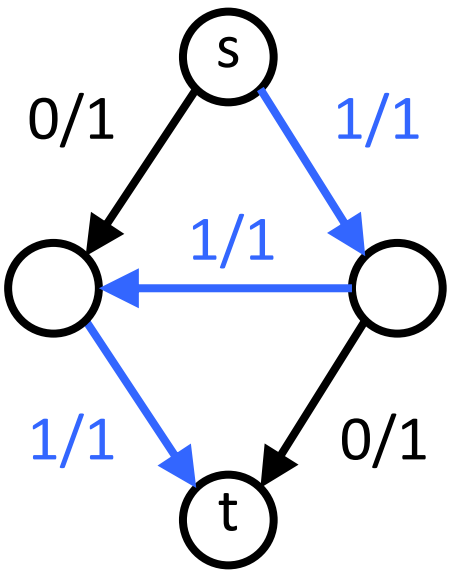
- We do not get stuck
- We are guarantee to find the maximum flow
- The algorithm is efficient!

Just taking the first path we see is not sufficient!

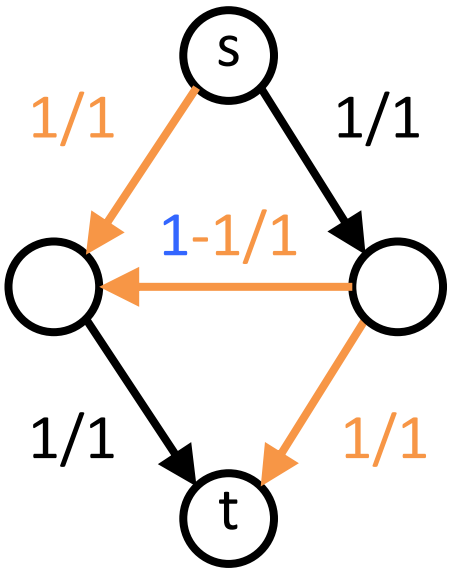
A better algorithm

Motivation: If we could subtract flow, then we could find it.

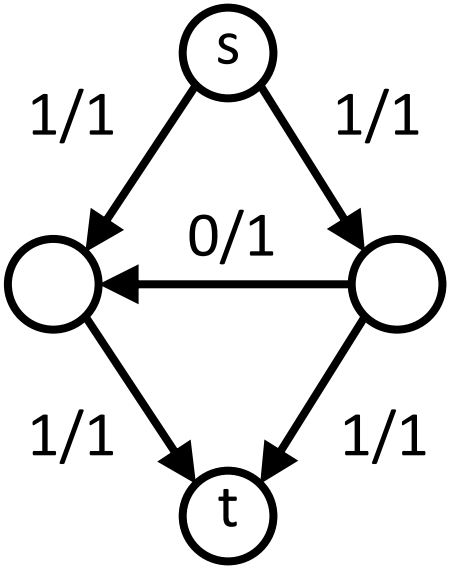




Algo 1 terminates here…



Negative value on edge that does not satisfy the definition



Using an edge is the opposite direction is equivalent to subtracting the flow

Residual graphs

Given a flow network G=(V,E) with edge capacities c and a given flow f, define the *residual graph* G_f as:

- G_f has the same vertices as G
- The edges E_f have capacities c_f (called *residual capacities*) that allow us to change the flow f, either by:
  1. Adding flow to an edge $e \in E$
  2. Subtracting flow from an edge $\in E$

Residual graphs

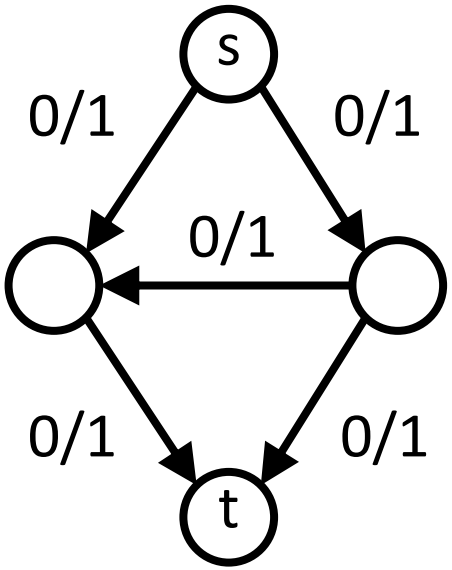
```
for each edge e = (u, v) ∈ E
   if f(e) < c(e)
   then {
      put a forward edge (u,v) in E_f
      with residual capacity c_f(e)=c(e)–f(e)
   }
   if f(e)>0
   then {
      put a backward edge (v,u) in E_f
      with residual capacity c_f(e) = f(e)
   }
}
```

We make a forward edge with “unused capacity”

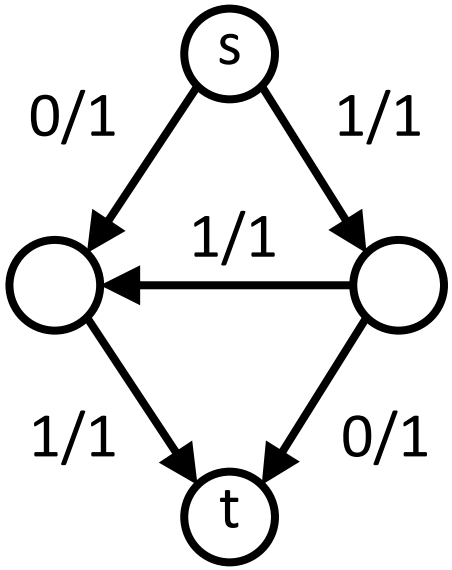
We make a backward edge with the opposite of the positive flow

Example 1/3

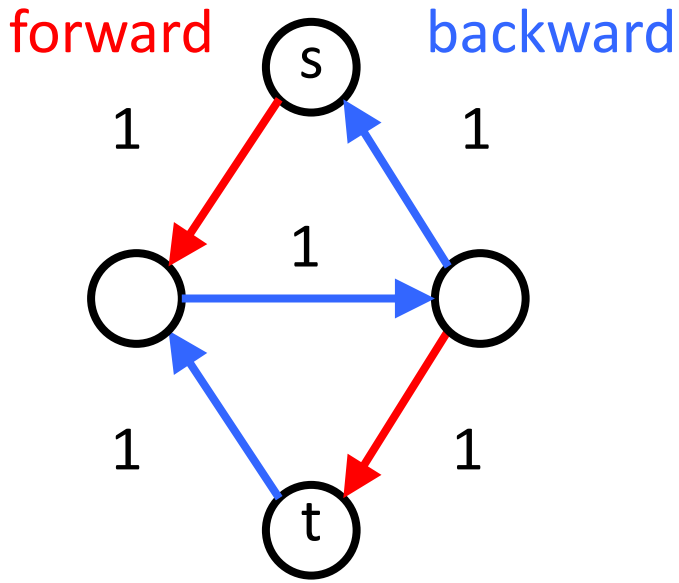
Flow network



Flow

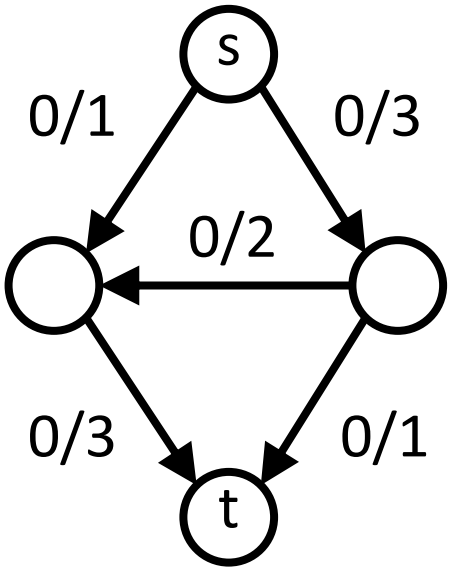


Residual graph

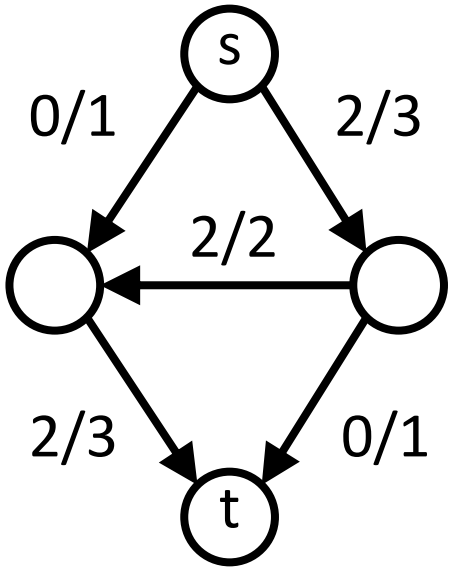


Example 2/3

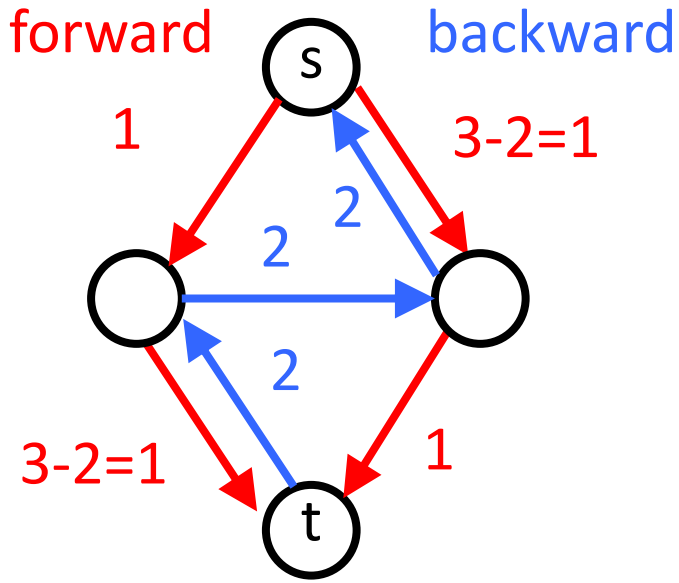
Flow network



Flow



Residual graph



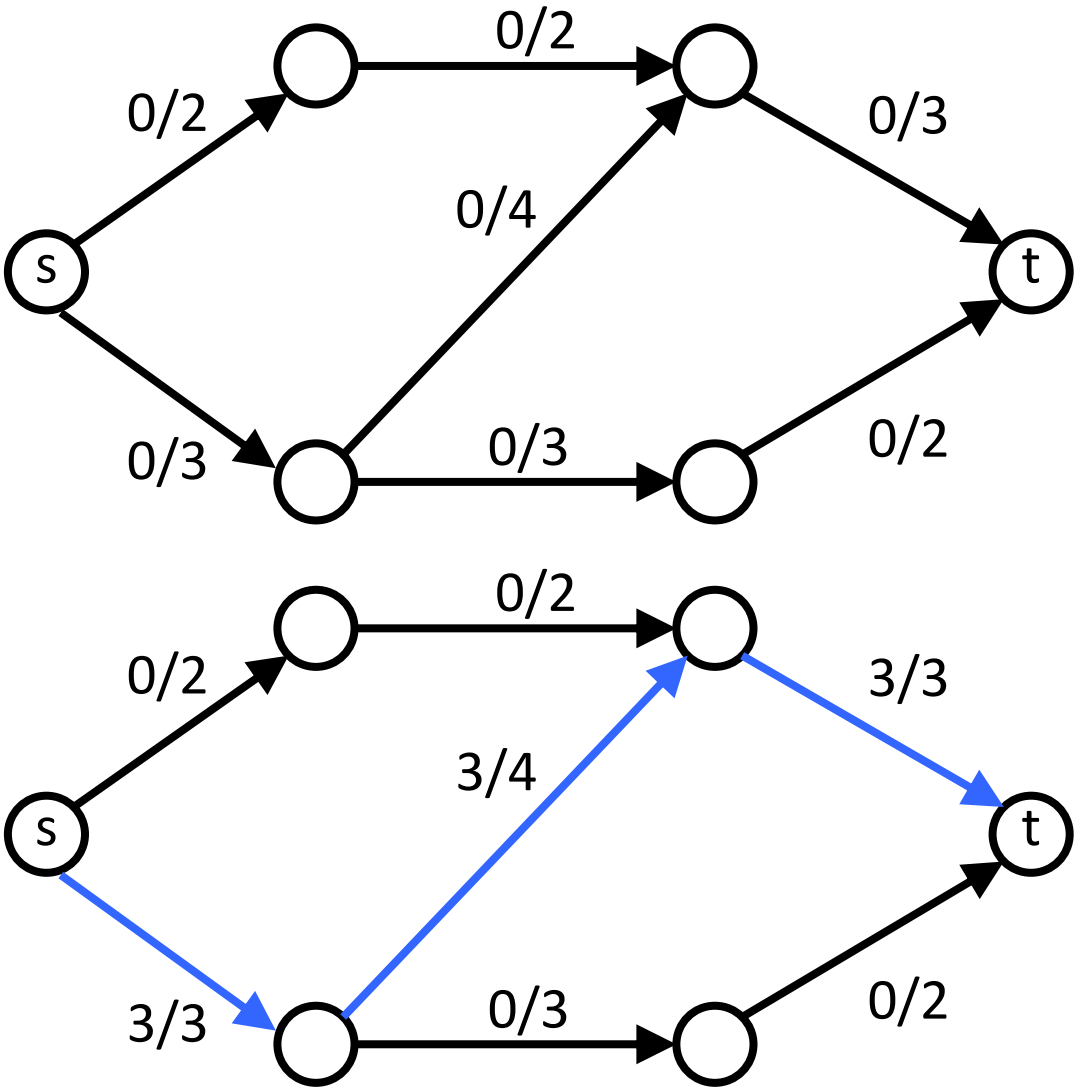
Remember!

Forward edge: unused capacity

Backwards edge: opposite of the positive flow (i.e., the flow that could be cancelled out)

If you think about it, the opportunity to cancel out that flow is also unused capacity

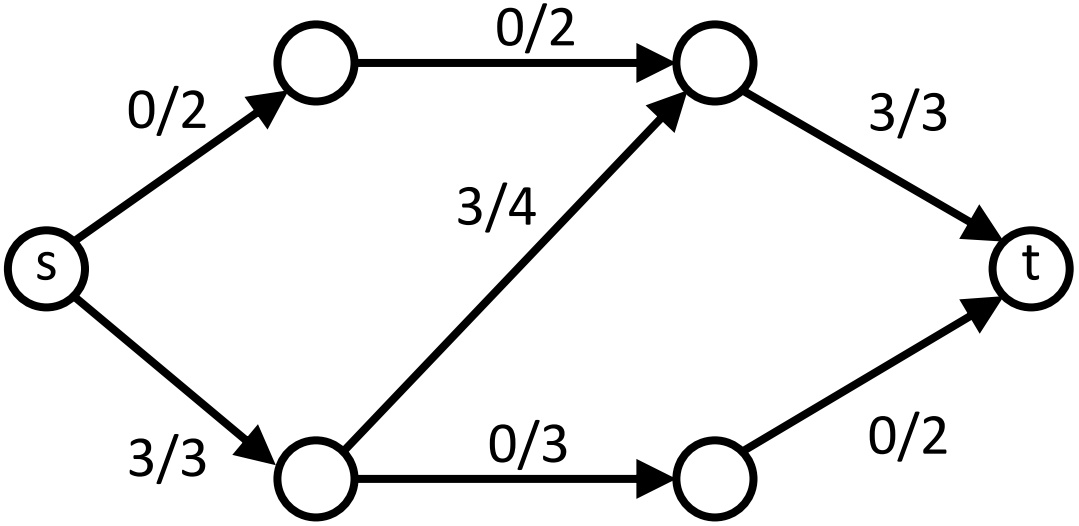
Example 3/3



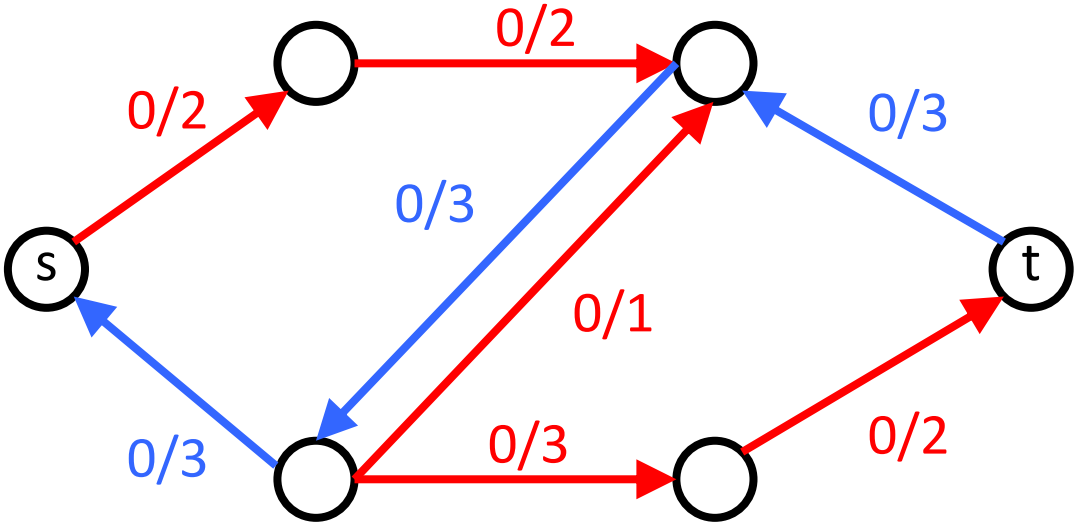
Example 3/3

Notation: flow/capacity

Flow

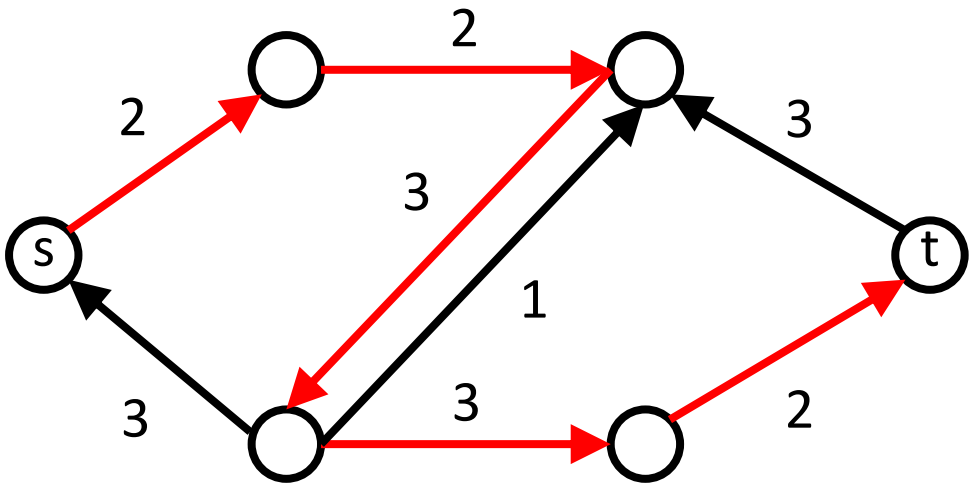


Residual graph



Augmenting path

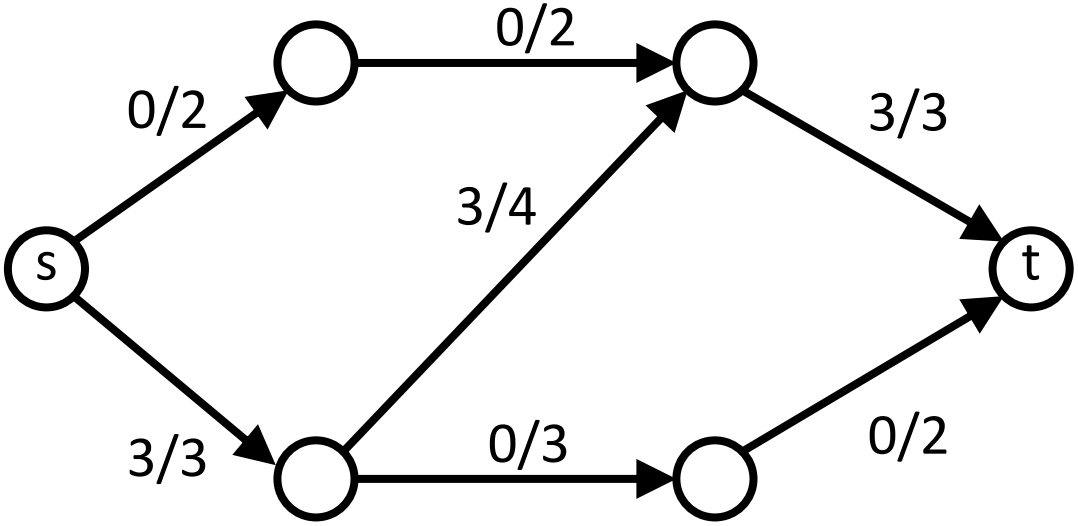
An augmenting path is a path from the source *s* to the sink *t* in the residual graph G_f that allows us to increase the flow.



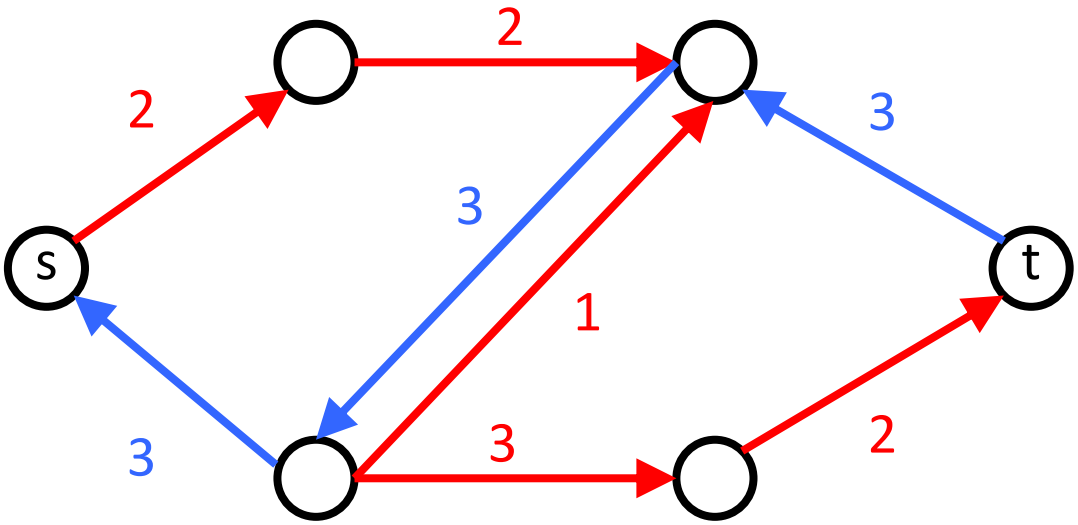
Q: By how much can we increase the flow using this path?

Example

Flow in G

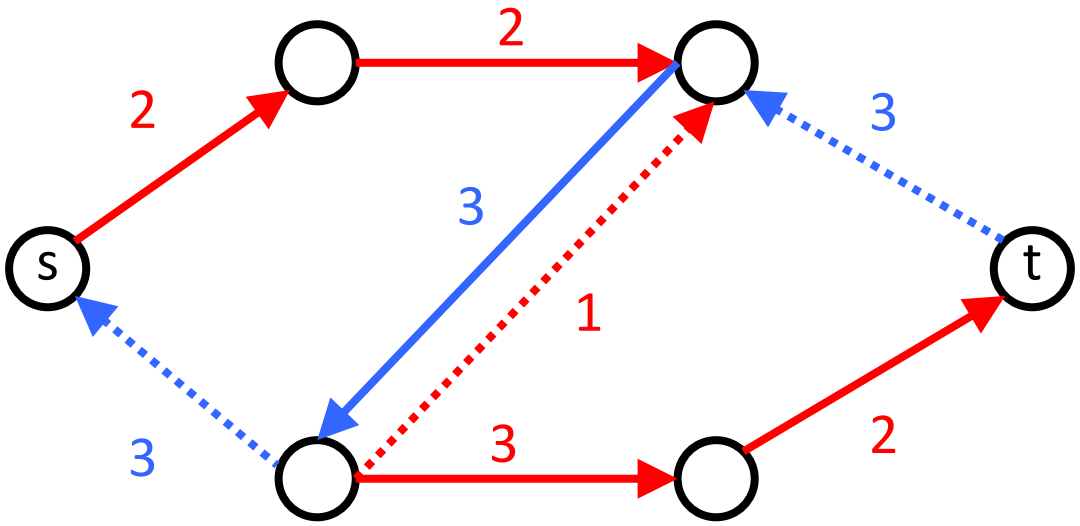


Residual graph G_f

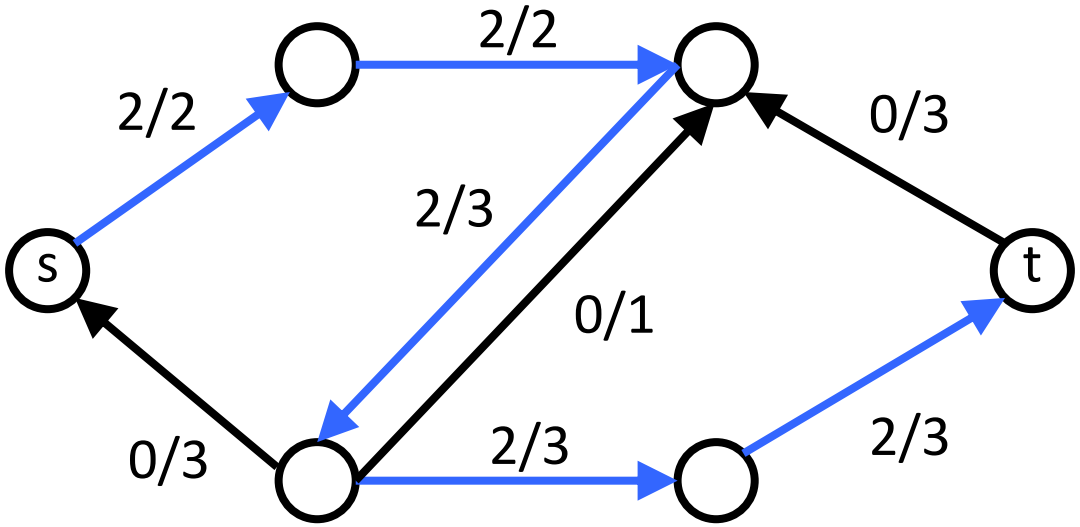


Example

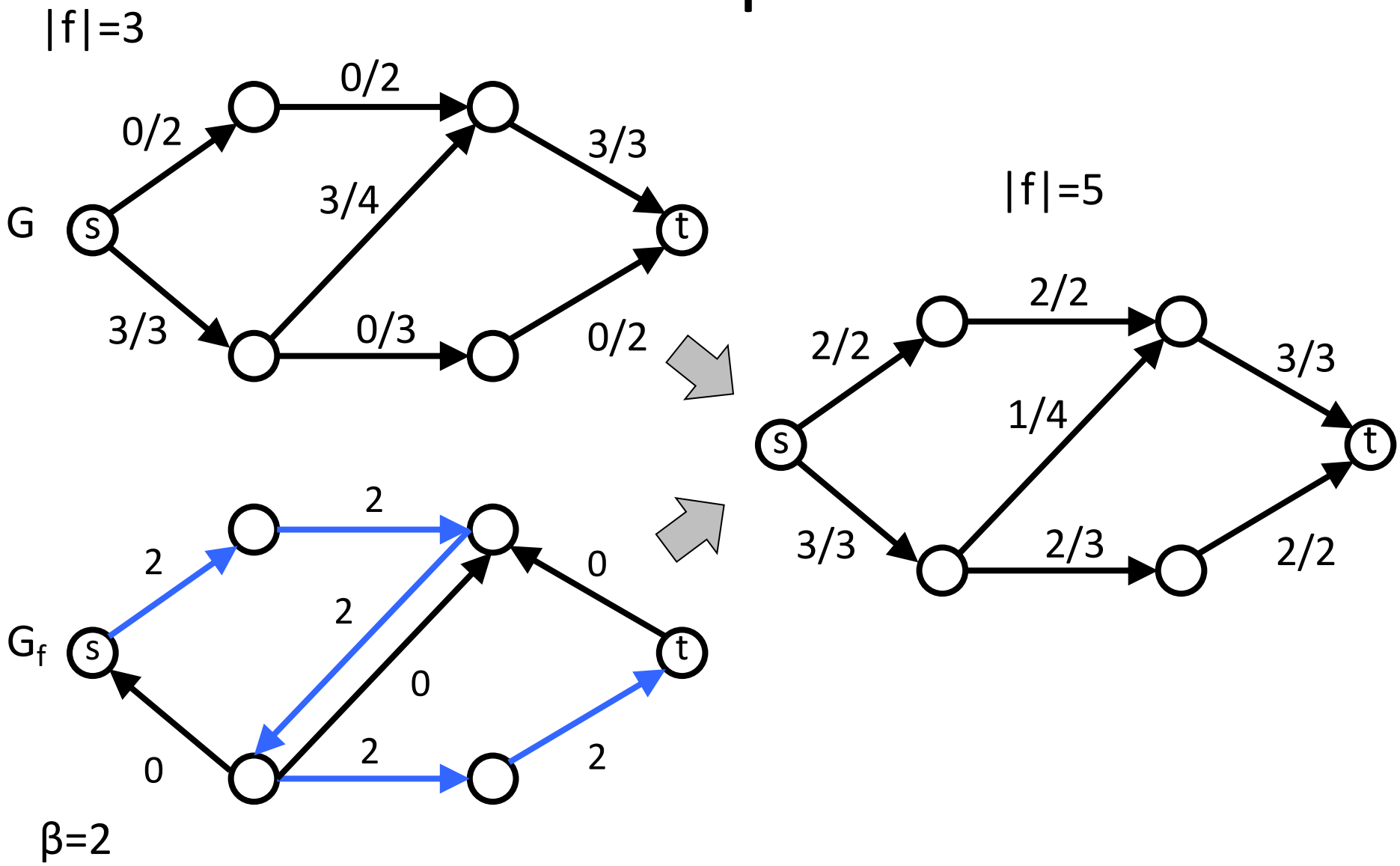
Residual graph G_f



Augmented path in G_f (value of the flow is the bottleneck value)



Example



Methodology

- Compute the residual graph G_f
- Find a path P
- Augment the flow f along the path P
    1. Let β be the bottleneck (smallest residual capacity $c_f(e)$ of edges on P)
    2. Add β to the flow f(e) on each edge of P.

Q: How do we add β into G?

Augmenting a path

```
f.augment(P) {
    β = min { c_f(e) | e ∈ P }
    for each edge e = (u,v) ∈ P {
        if e is a forward edge {
            f(e) += β
        } else { // e is a backward edge
            f(e) -= β
        }
    }
}
```

Forward edges model unused flow, so they can be used directly
Backward edges model the ability to reduce opposite flow, so we subtract

Ford-Fulkerson algorithm

f $\leftarrow$0

$G_f \leftarrow G$

`while` (there is a s-t path in G_f) {

    `f.augment(P)`

    update G_f based on new f

}

Correctness (termination)

Claim: The Ford-Fulkerson algorithm terminates.

Proof:

- The capacities and flows are strictly positive integers.
- The sum of capacities leaving s is finite.
- Bottleneck values β are strictly positive integers.
- The flow increase by β after each iteration of the loop.
- The flow is an increasing sequence of integers that is bounded.

Complexity (Running time)

- Let $C = \sum_{\substack{e \in E \\ outgoing \\ from\ s}} c(e)$
- Finding an augmenting path from *s* to *t* takes O(|E|) (e.g., BFS or DFS).
- The flow increases by at least 1 at each iteration of the main while loop.
- The algorithm runs in O($C \cdot |E|$)