COMP 251

Algorithms & Data Structures (Winter 2021)

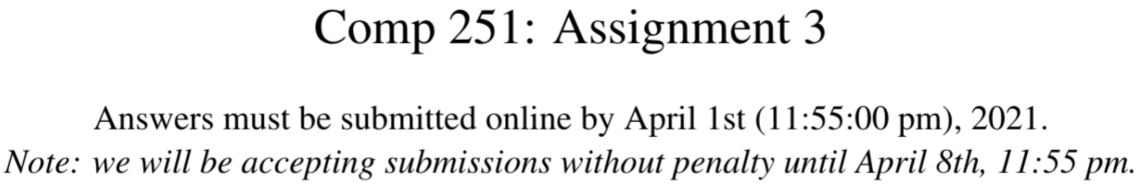
Graphs – Flow Network 1

School of Computer Science
McGill University

Slides of (Comp321 ,2021), Langer (2014), Kleinberg & Tardos, 2005 & Cormen et al., 2009, Jaehyun Park' slides CS 97SI, Top-coder tutorials, T-414-AFLV Course, Programming Challenges books, slides from D. Plaisted (UNC) and Comp251-Fall McGill.

Announcements

- Final exam.
  - April 22, 2021 at 2:00 PM - April 25, 2021 at 2:00 PM
- Assignment 3.
  - It will be released just after the lecture.



Comp 251: Assignment 3

Answers must be submitted online by April 1st (11:55:00 pm), 2021.

Note: we will be accepting submissions without penalty until April 8th, 11:55 pm.

  - 3 questions.
    - Topics needed to solve the assignment are already covered (after finishing todays lecture).

Outline

- Graphs.
  - Introduction.
  - Topological Sort. / Strong Connected Components
  - Network Flow 1.
    - Introduction
    - Ford-Fulkerson
  - Network Flow 2.
  - Shortest Path.
  - Minimum Spanning Trees.
  - Bipartite Graphs.

Flow Network

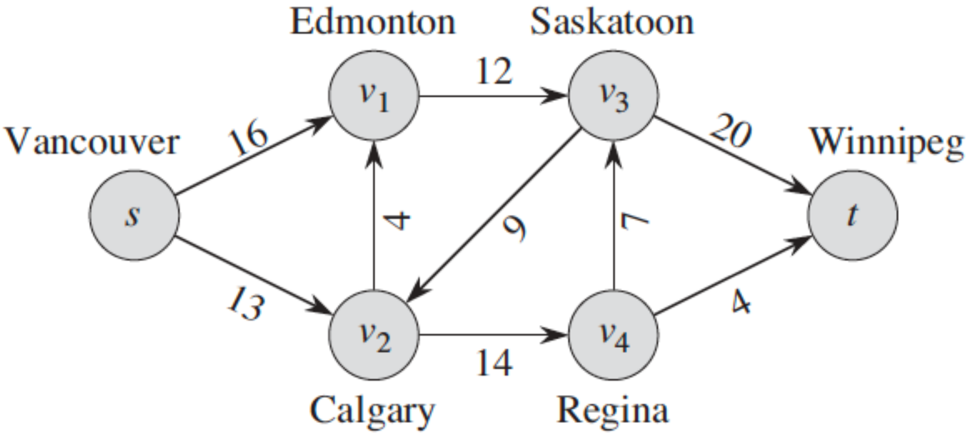
$G = (V, E)$ directed.

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

If $(u,\mathrm{v}) \notin E$, then $c(u,\mathrm{v}) = 0$.

Source vertex s, ***sink*** vertex t, assume $s \leadsto \mathrm{v} \leadsto t$ for all $\mathrm{v} \in V$.

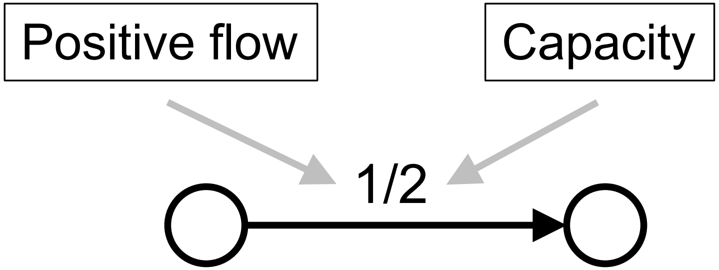
c(u,v) is a non-negative integer.
If $(u,\mathrm{v}) \in$ E, then $(u,\mathrm{v}) \notin$ E.
No incoming edges in source.
No outcoming edges in sink.



Flow Network - Definitions

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

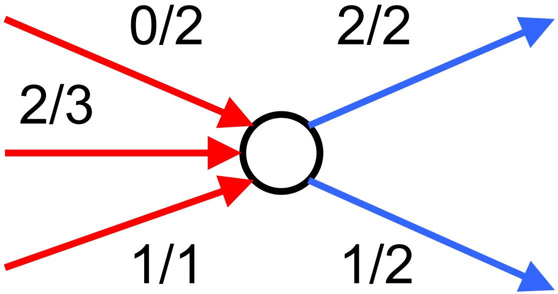
Capacity constraint: For all u, v ∈ V, 0 ≤ p(u, v) ≤ c(u, v)



Positive flow

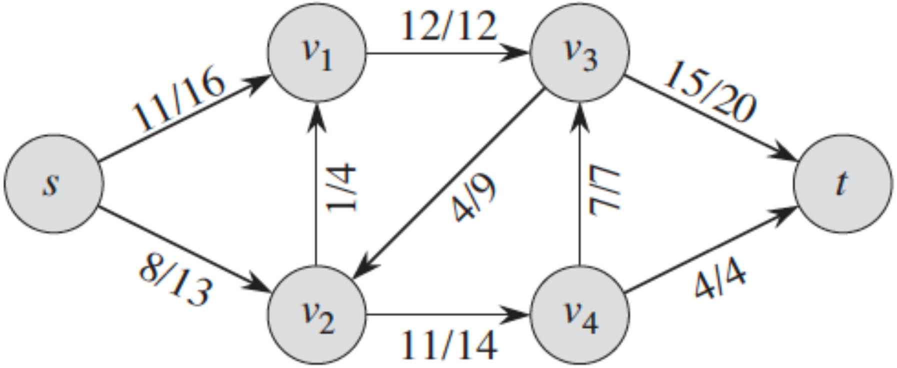
Capacity

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

Flow Network - Example



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

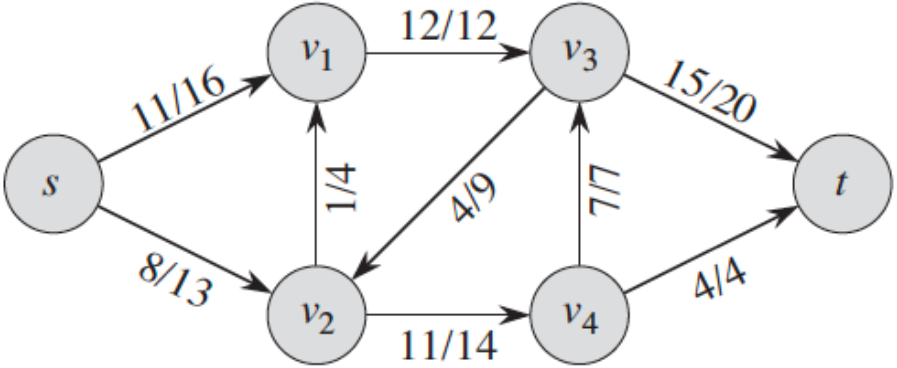
The value of the flow: the total flow out of the source minus the flow into the source

$$|f| = \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s)$$

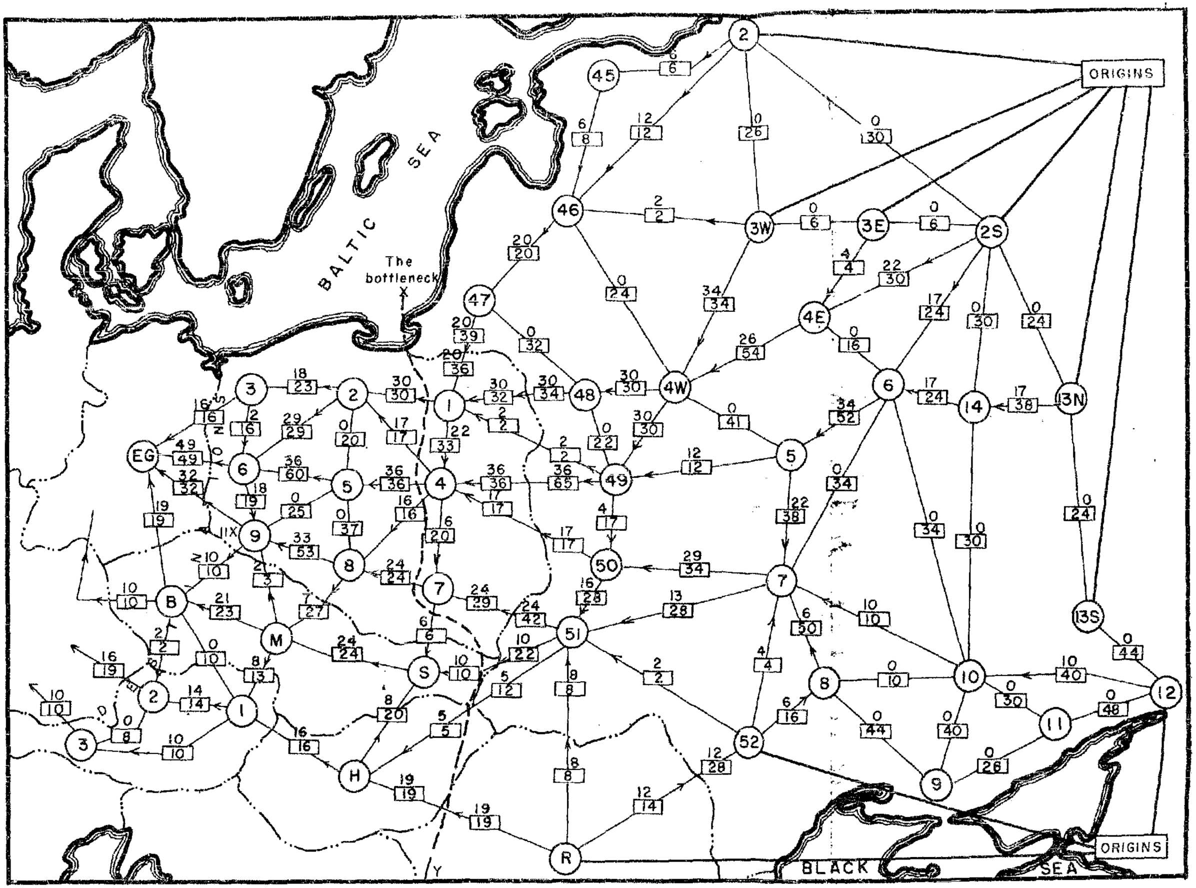
|f|is also equal to the total net flow into the target vertex *t*

Maximum-Flow Problem

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



Maximum-Flow Problem - Applications



SECRET RM-1573
10-24-55
-33-

Fig. 7 — Traffic pattern: entire network available

Legend:

— · · — International boundary

(B) Railway operating division

9 ←[12]— Capacity: 12 each way per day. Required flow of 9 per day toward destinations (in direction of arrow) with equivalent number of returning trains in opposite direction

All capacities in trains / √1000's of tons } each way per day

Origins: Divisions 2, 3W, 3E, 2S, 13N, 13S, 12, 52 (USSR), and Roumania

Destinations: Divisions 3, 6, 9 (Poland); B (Czechoslovavakia); and 2, 3 (Austria)

Alternative destinations: Germany or East Germany

Note IIX at Division 9, Poland

Maximum-Flow – Naïve algorithm

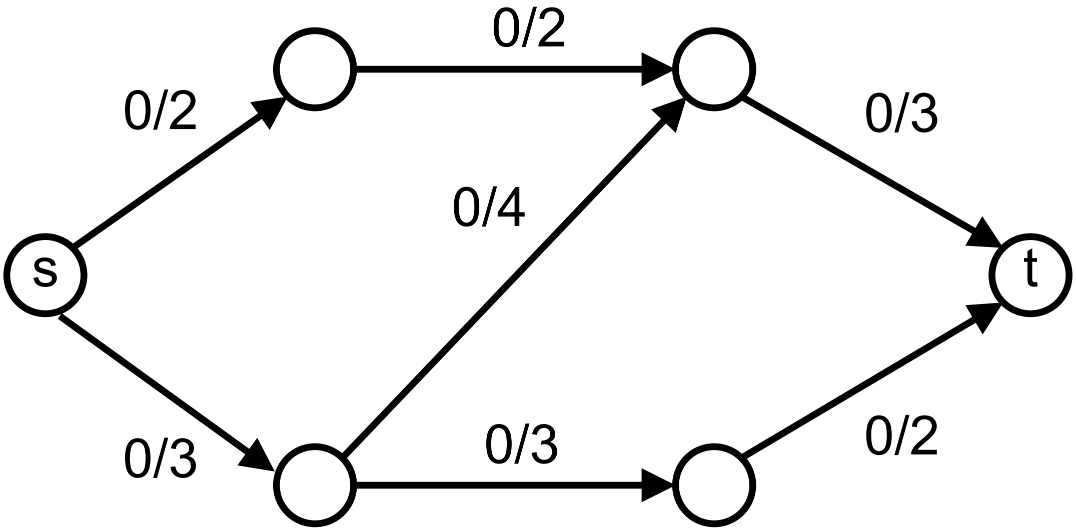
```
Initialize f = 0
While true {
    if (∃ path P from $s$ to $t$ such that all edges have a flow less than capacity)
    then
        increase flow on P up to max capacity
    else
        break
}
```

Maximum-Flow – 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 }
}
```

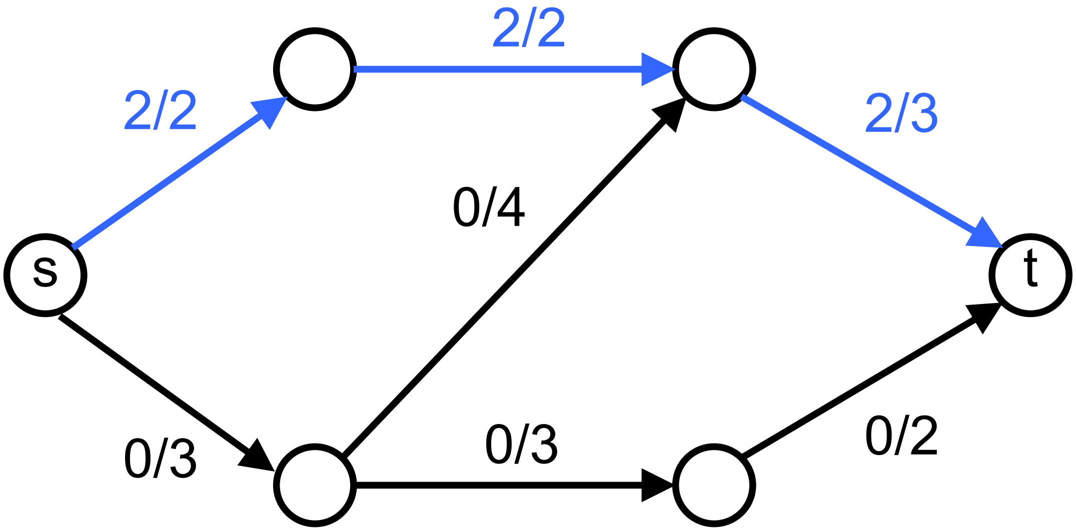
Maximum-Flow – Naïve algorithm

Example where algorithm works



Maximum-Flow – Naïve algorithm

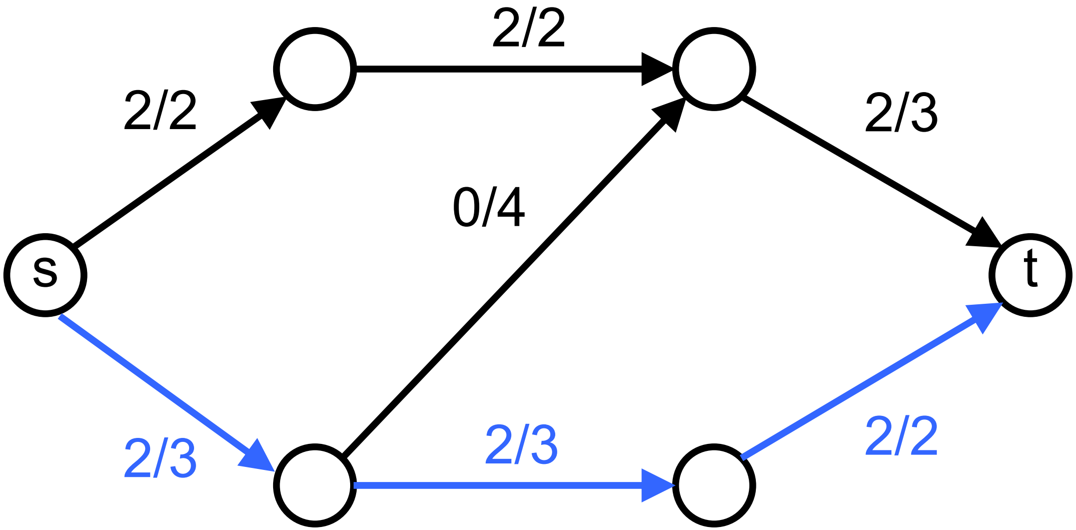
Example where algorithm works



|f|=2

Maximum-Flow – Naïve algorithm

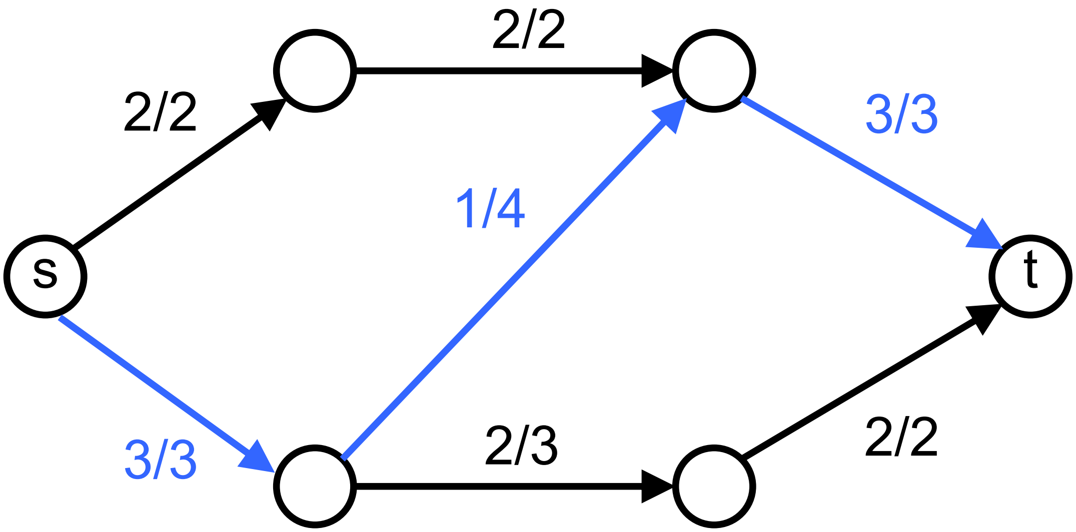
Example where algorithm works



|f|=4

Maximum-Flow – Naïve algorithm

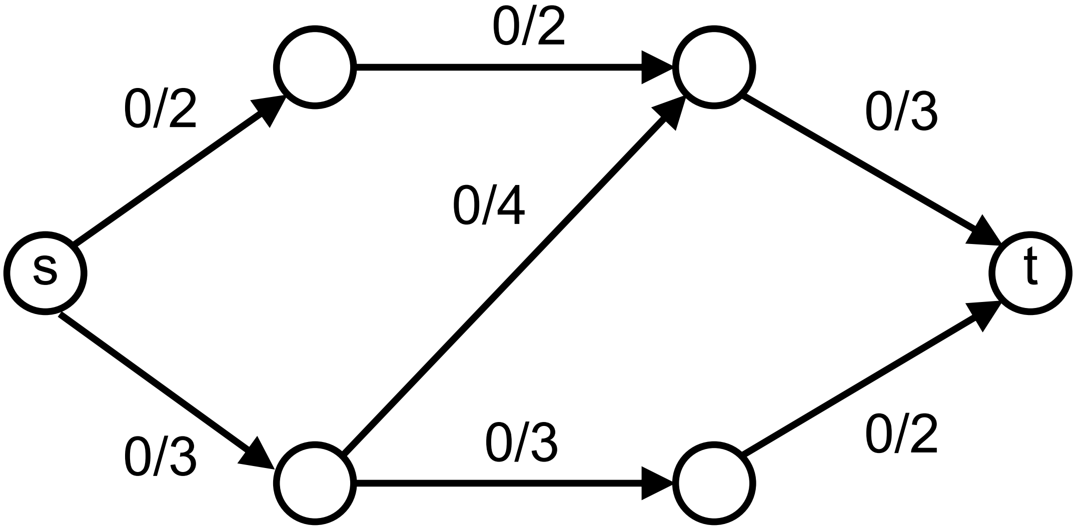
Example where algorithm works



|f|=5

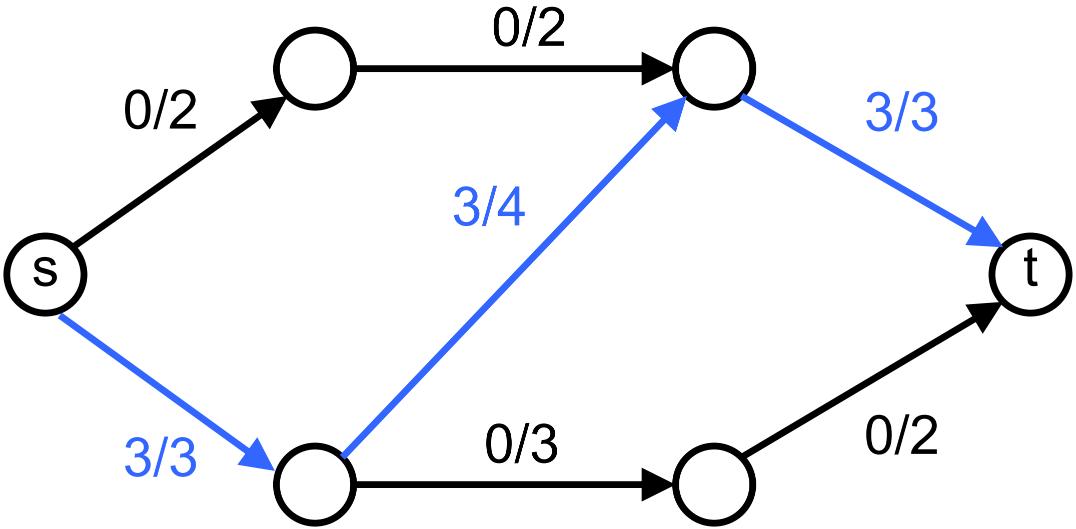
Maximum-Flow – Naïve algorithm

Example where algorithm fails!



Maximum-Flow – Naïve algorithm

Example where algorithm fails!



|f|=3 And terminates…

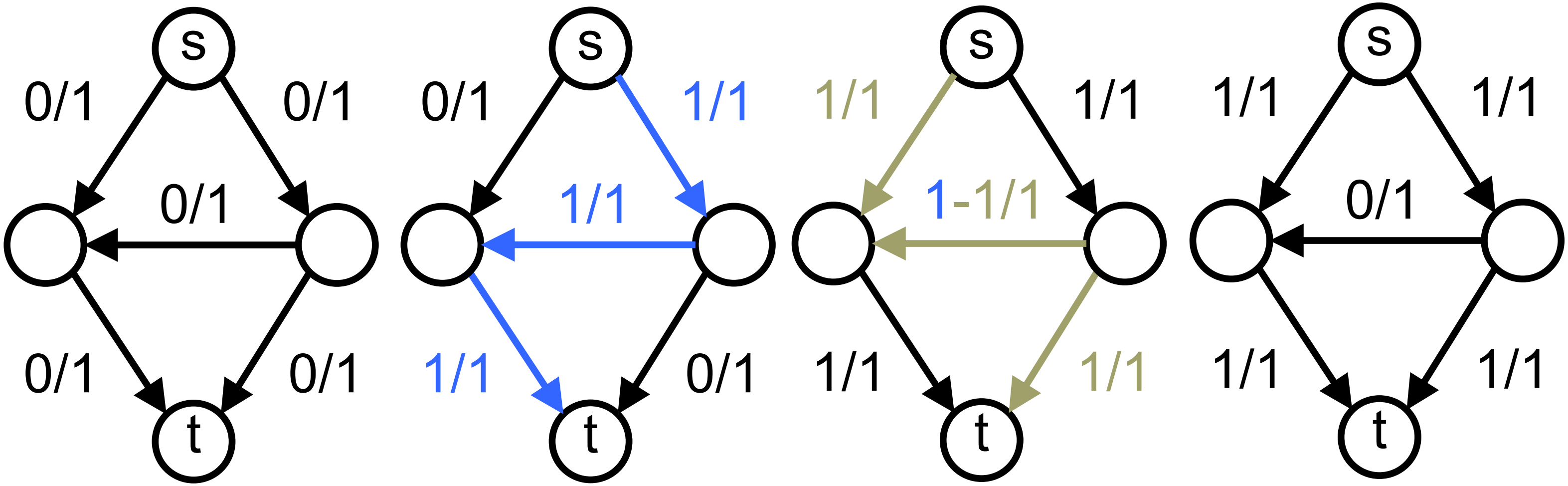
Maximum-Flow – Algorithm - Challenges

How to choose paths such that:

- We do not get stuck.
  - We need a way to ‘un-do’ actions.
- We guarantee to find the maximum flow
- The algorithm is efficient!

Maximum-Flow – A better Algorithm

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

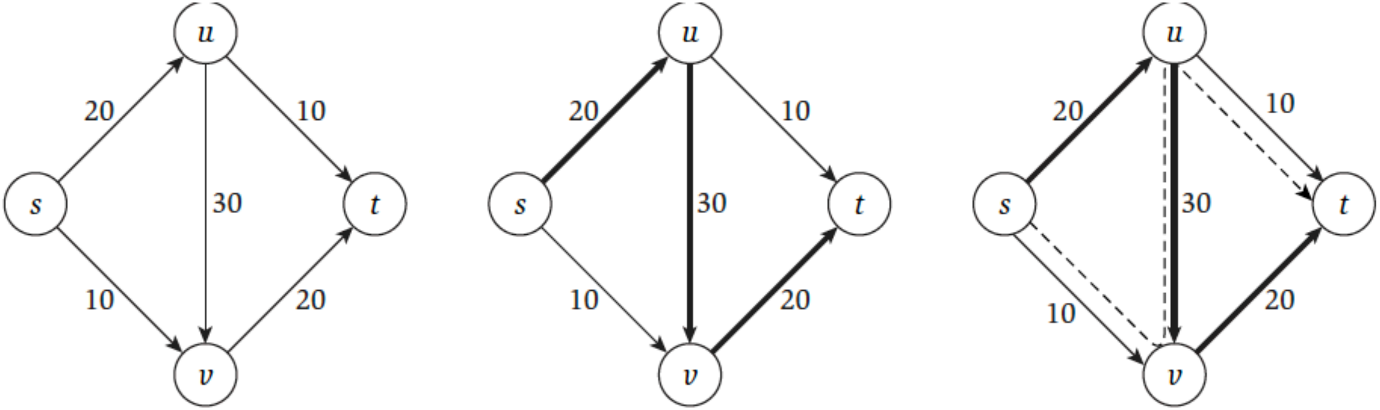


Algo 1 terminates here…

Negative value on edge that does not satisfy the definition

Maximum-Flow – A better Algorithm

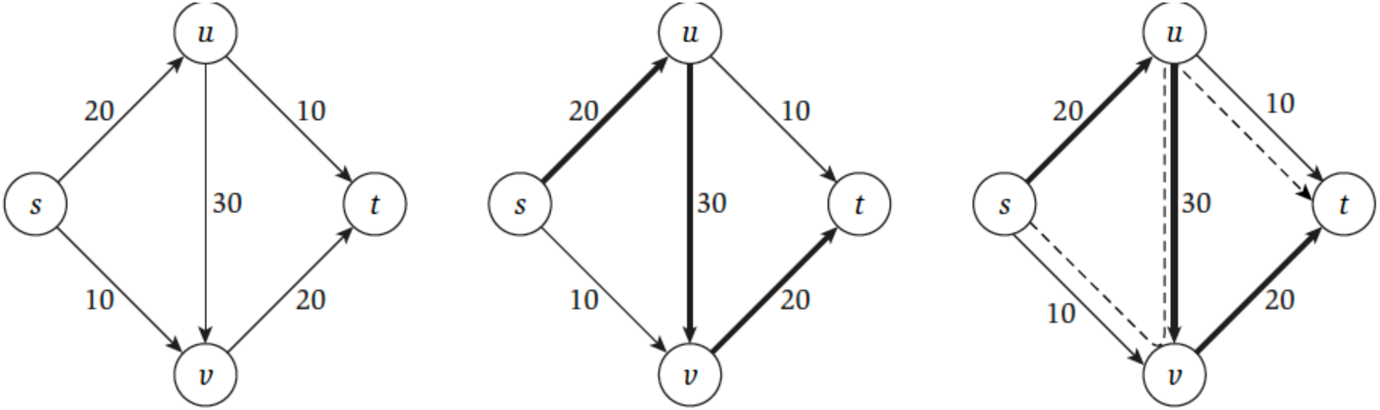
Motivation: If we could subtract flow, then we could find it.



- We push 10 units of flow along *(s, v).*
  - this now results in too much flow coming into *v*.
- So we “undo” 10 units of flow on *(u, v).*
  - this restores the conservation condition at v but results in too little flow leaving *u*.
- So, finally, we push 10 units of flow along *(u, t).*
  - restoring the conservation condition at *u*.
- We now have a valid flow, and its value is 30.

Maximum-Flow – A better Algorithm

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



- This is a more general way of pushing flow: We can push **forward** on edges with leftover capacity, and we can push **backward** on edges that are already carrying flow, to divert it in a different direction.
  - *Residual graphs* provides a systematic way to search for forward-backward operations such as this. A flow in a residual network provides a roadmap for adding flow to the original flow network.

Maximum-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:
  - Adding flow to an edge e ∈ E (forward edge)
    - For each edge e = (u, v) of G on which $f(e) < c_e$, there are $c_e - f(e)$ "leftover" units of capacity on which we could try pushing flow forward.
  - Subtracting flow from an edge ∈ E (backward edge)
    - For each edge e = (u, v) of G on which f (e) > 0, there are f (e) units of flow that we can "undo" if we want to, by pushing flow backward.
- The edges in E_f are either edges in E or their reversals.

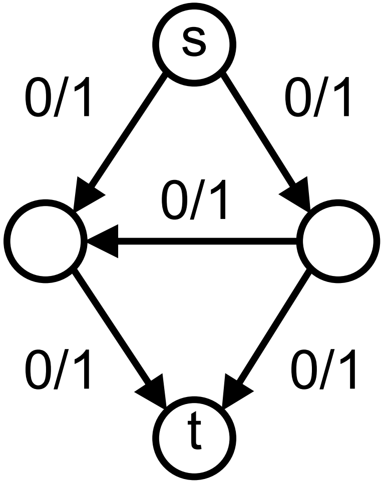
$$|E_f| \leq 2\ |E|$$

Maximum-Flow – Residual graphs

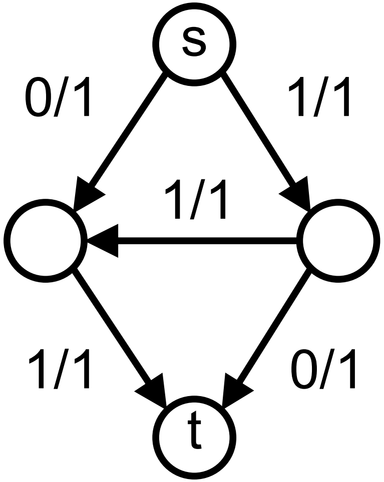
```
for each edge e = (u, v) ∈ E
      if f(e) < c(e)
      then {
            put a forward edge (u,v) in $E_f$
            with residual capacity $c_f(e)=c(e)-f(e)$
      }
      if f(e)>0
      then {
            put a backward edge (v,u) in $E_f$
            with residual capacity $c_f(e) = f(e)$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

Residual graphs – Example 1/3

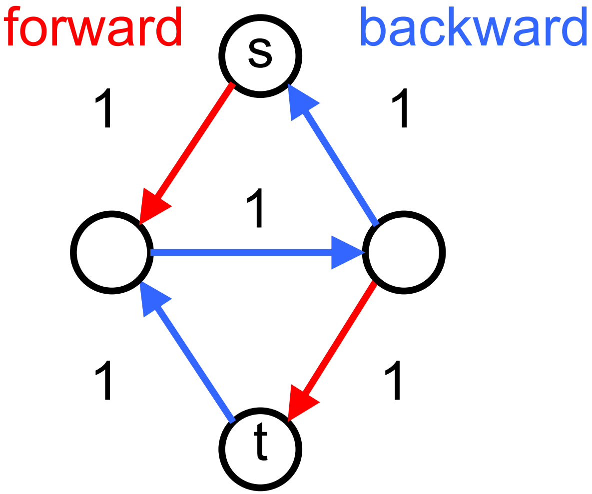
Flow network



Flow

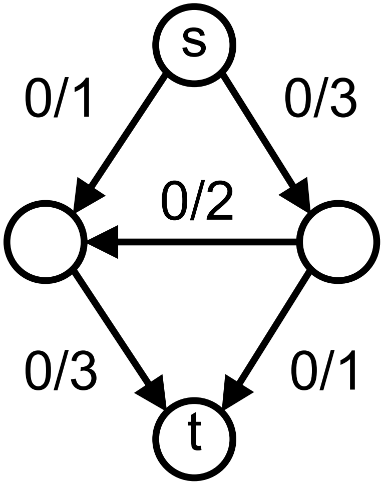


Residual graph

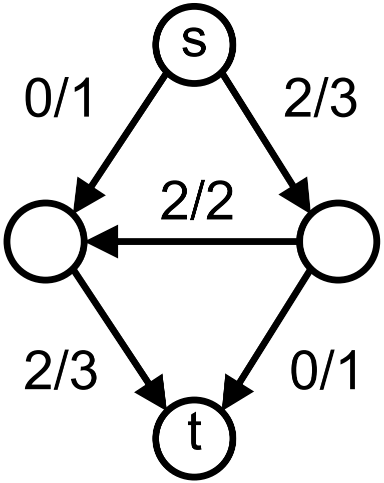


Residual graphs – Example 2/3

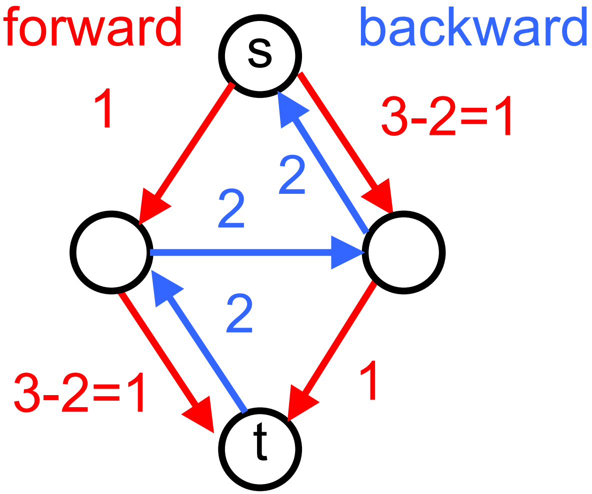
Flow network



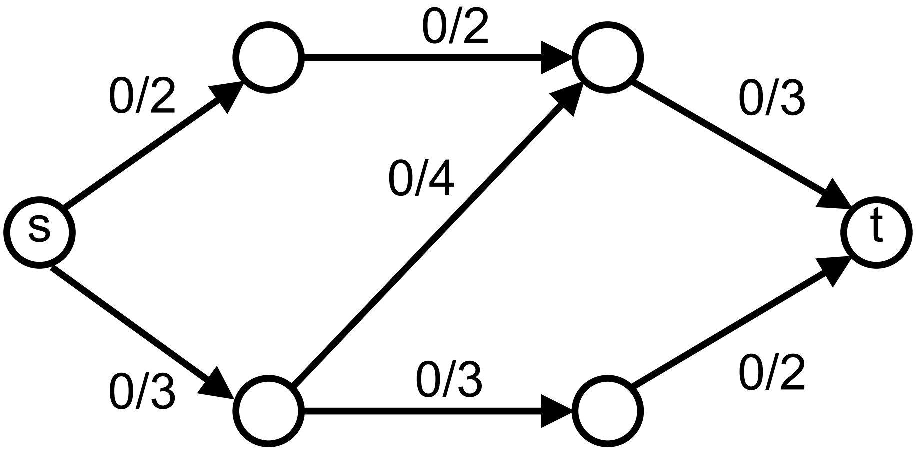
Flow

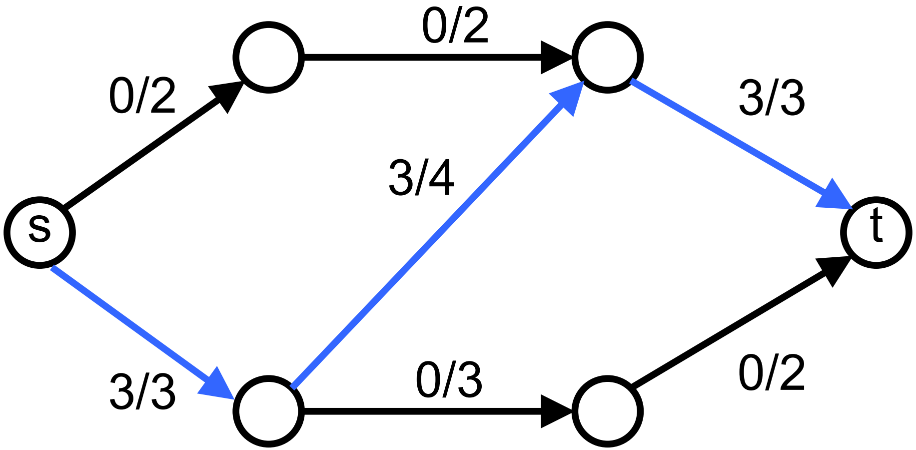


Residual graph

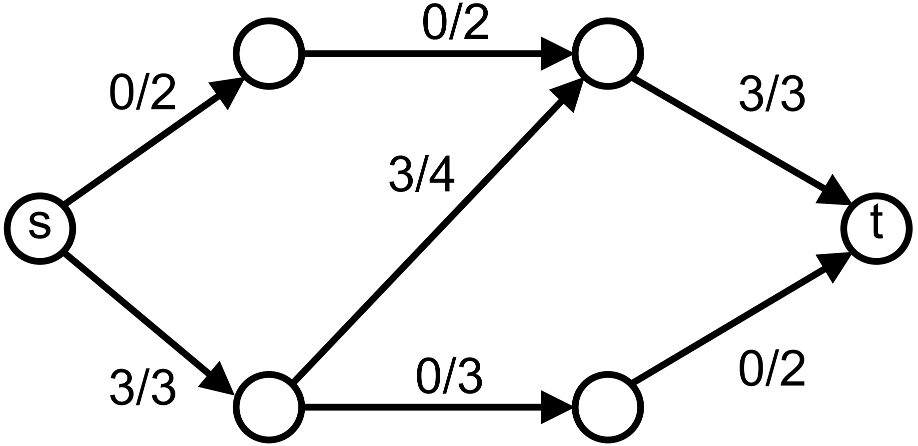


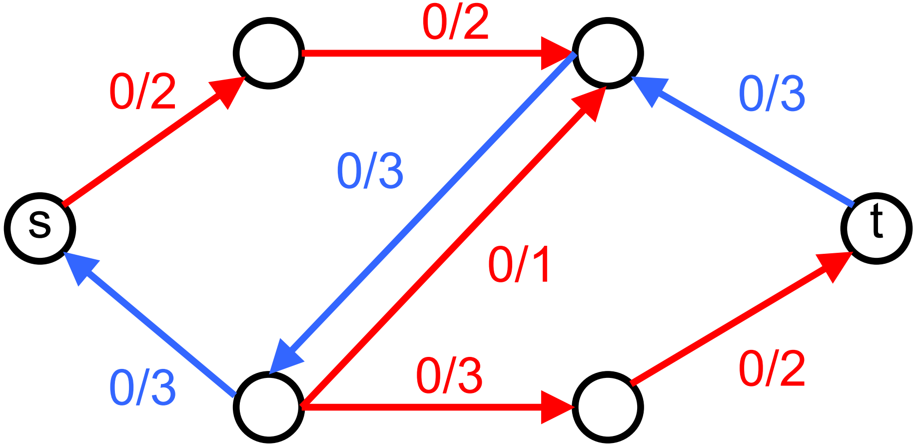
Residual graphs – Example 3/3





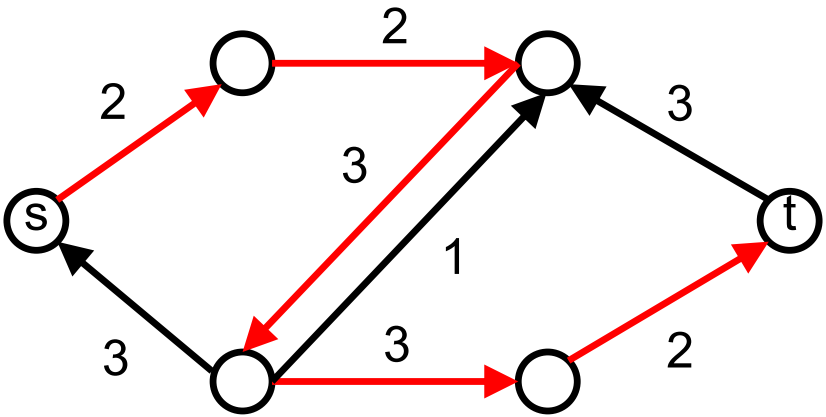
Residual graphs – Example 3/3





Augmenting path

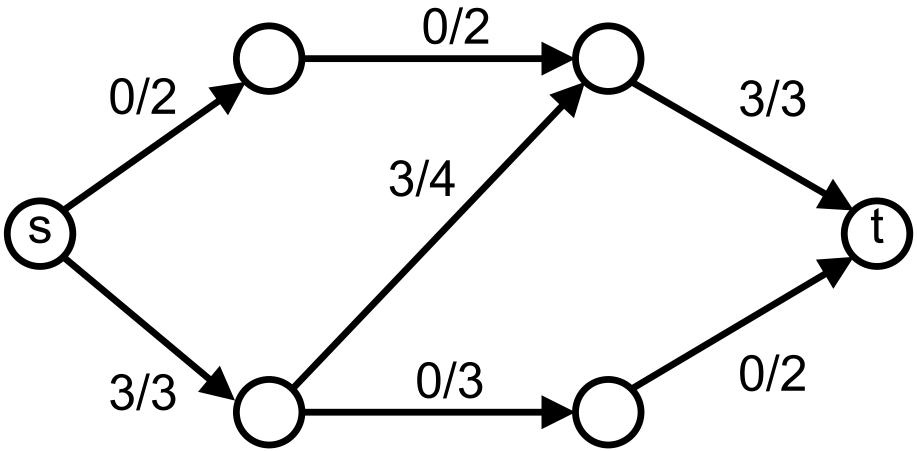
An augmenting path is a path from the source *s* to the sink *t* in the residual graph G_f that allows us to increase the flow.



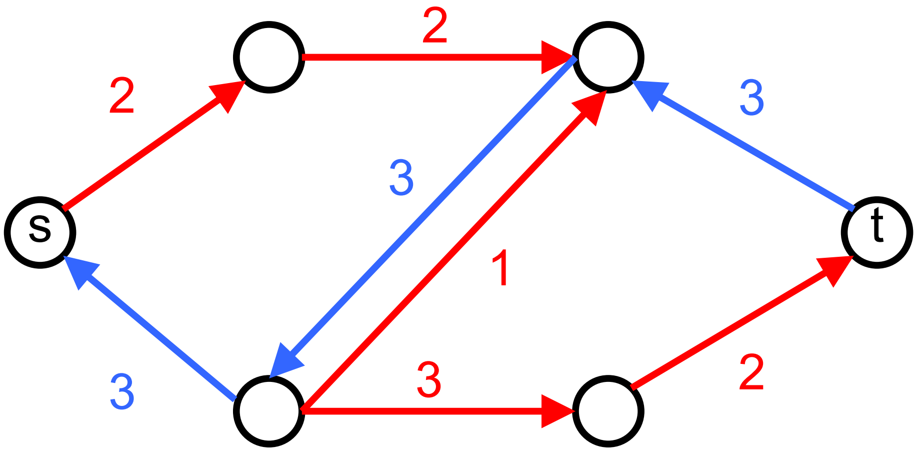
Q: By how much can we increase the flow using this path?

Augmenting path - Example

Flow in G

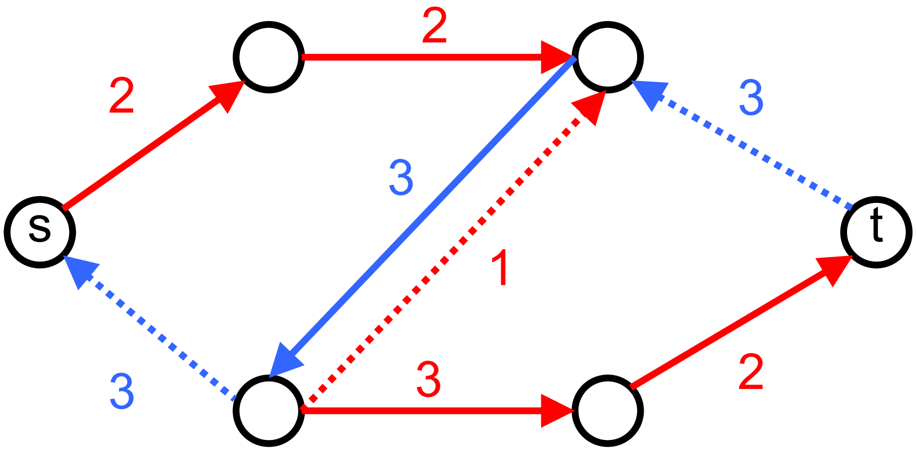


Residual graph G_f

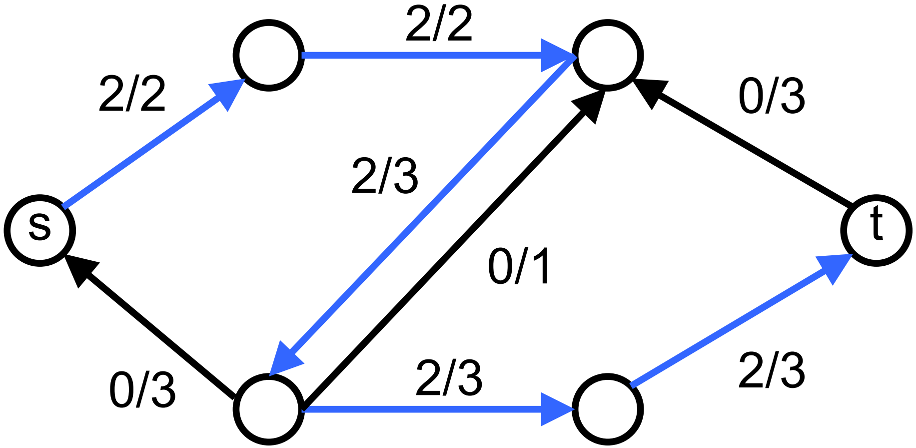


Augmenting path - Example

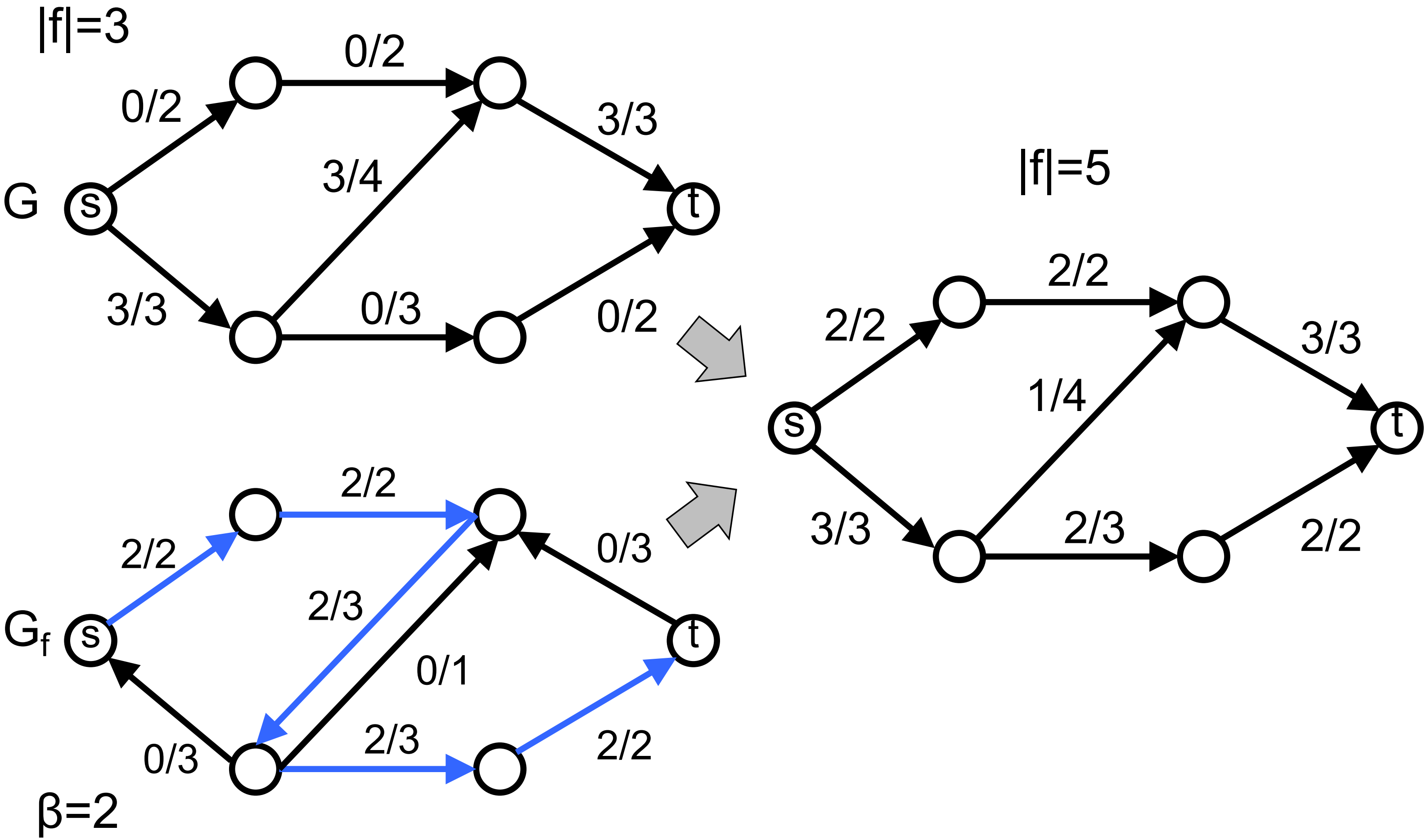
Residual graph G_f



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



Augmenting path - Example



Methodology

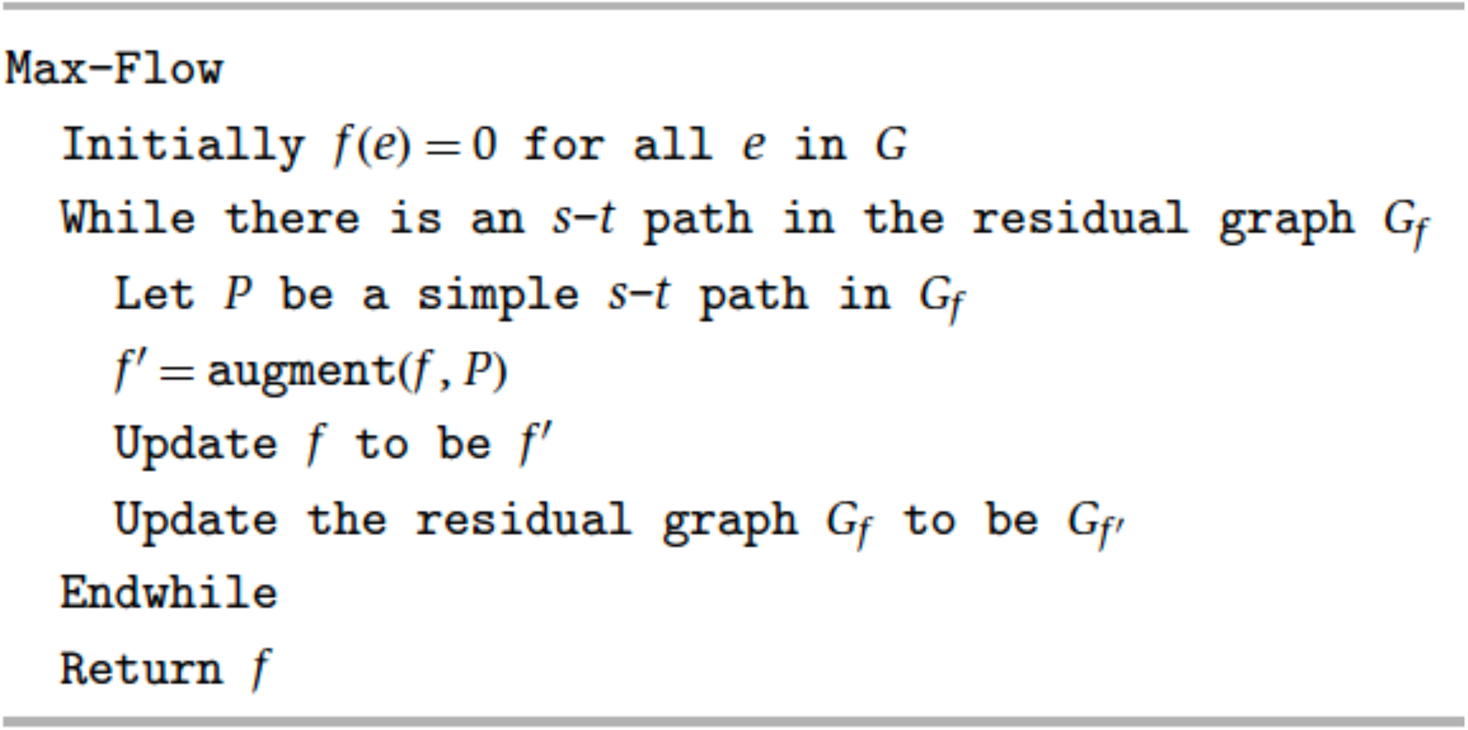
- Compute the residual graph G_f
- Find a path P
- Augment the flow f along the path P
  1. Let β be the bottleneck (smallest residual capacity $c_f(e)$ of edges on P)
  2. Add β to the flow f(e) on each edge of P.

Q: How do we add β into G?

Augmenting a path

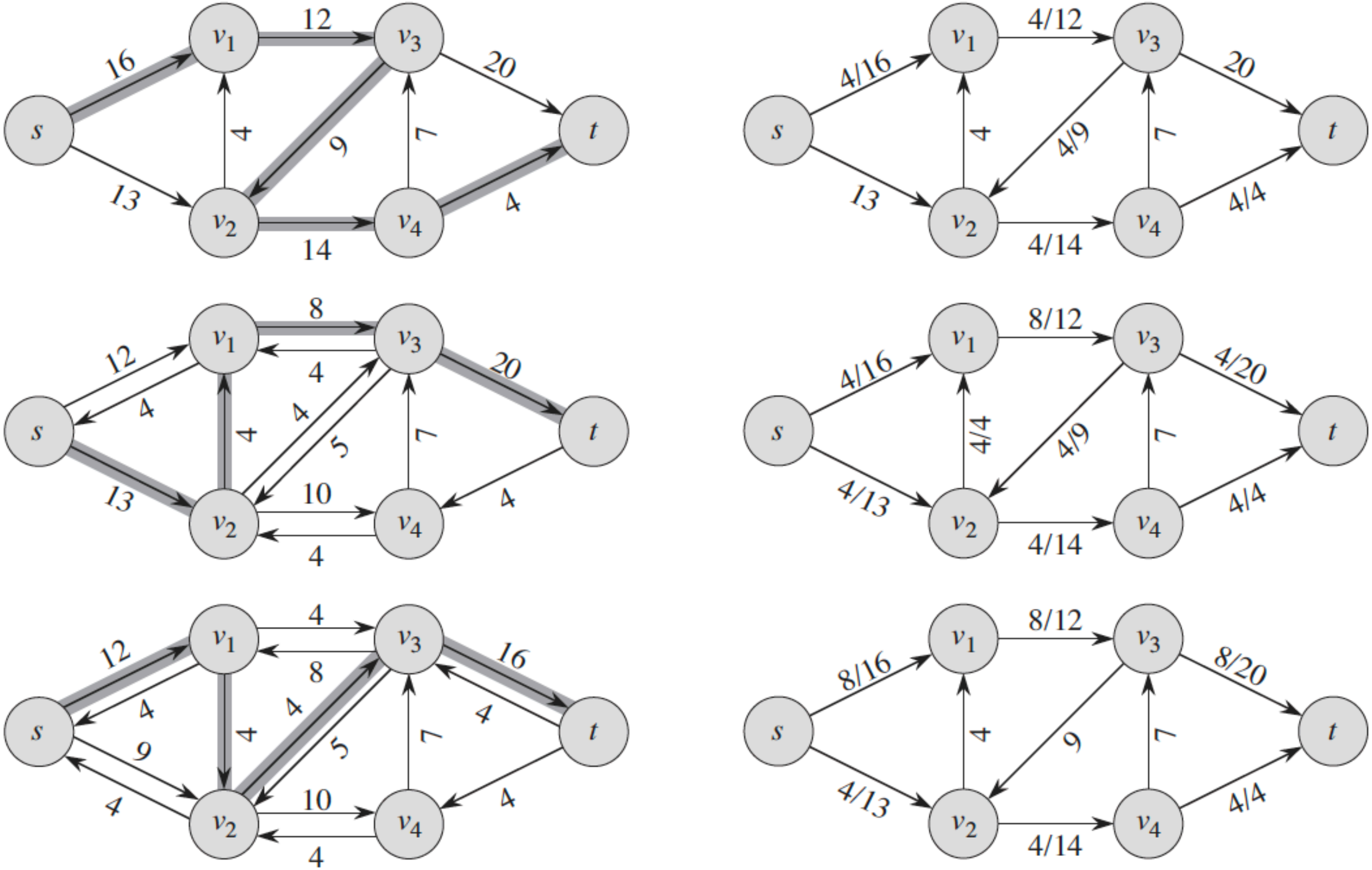
```
f.augment(P) {
        $\beta = \min \{ c_f(e) \mid e \in P \}$
        for each edge e = (u,v) ∈ P {
                if e is a forward edge {
                        f(e) += β
                } else { // e is a backward edge
                        f(e) -= β
                }
        }
}
```

Ford-Fulkerson algorithm

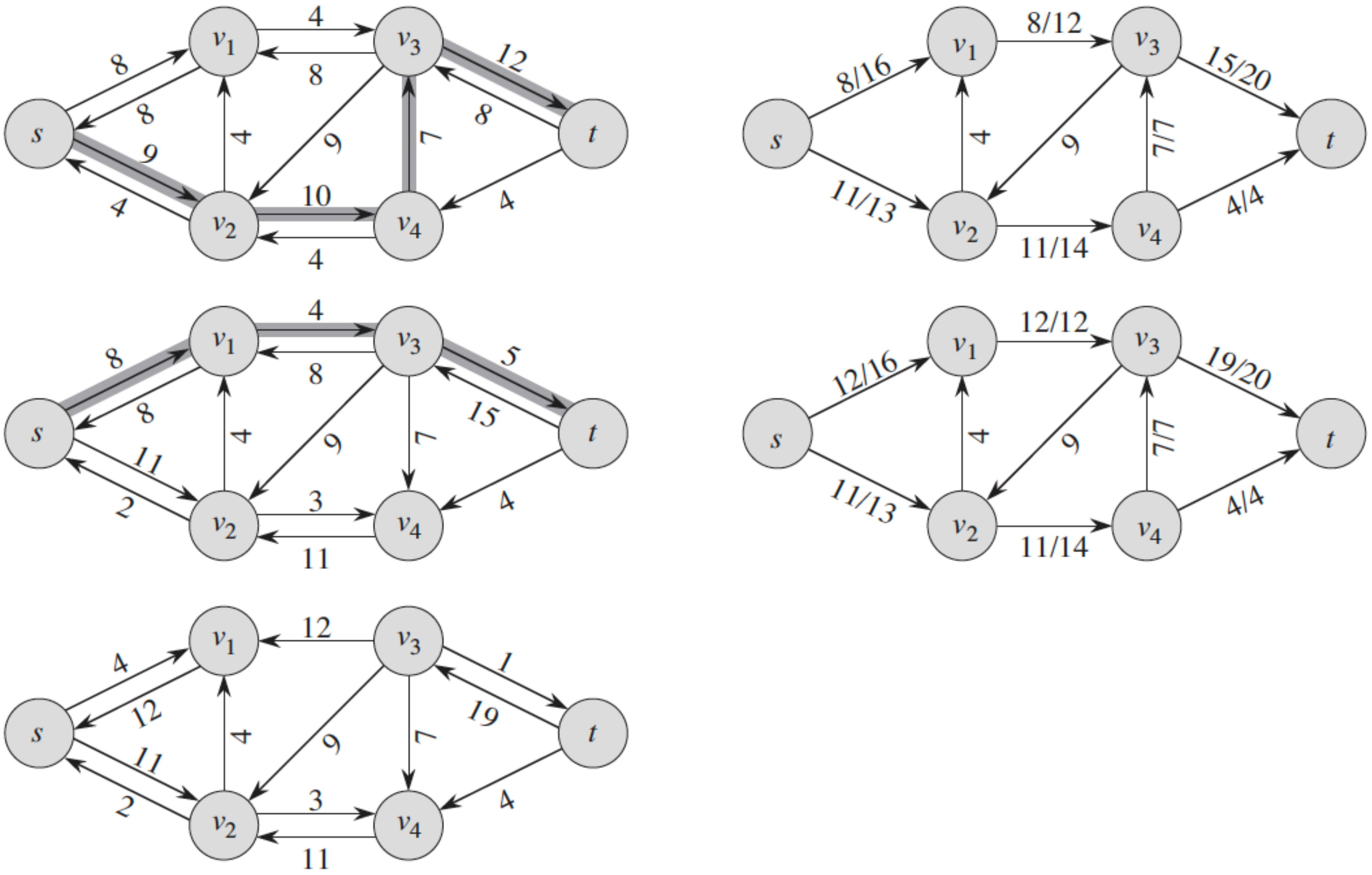


```
Max-Flow
  Initially $f(e)=0$ for all $e$ in $G$
  While there is an $s$-$t$ path in the residual graph $G_f$
    Let $P$ be a simple $s$-$t$ path in $G_f$
    $f' = \text{augment}(f, P)$
    Update $f$ to be $f'$
    Update the residual graph $G_f$ to be $G_{f'}$
  Endwhile
  Return $f$
```

Ford-Fulkerson algorithm



Ford-Fulkerson algorithm



Ford-Fulkerson algorithm - complexity

- 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* · |E|)

Outline

- Graphs.
  - Introduction.
  - Topological Sort. / Strong Connected Components
  - Network Flow 1.
    - Introduction
    - Ford-Fulkerson
  - Network Flow 2.
  - Shortest Path.
  - Minimum Spanning Trees.
  - Bipartite Graphs.


time for questions