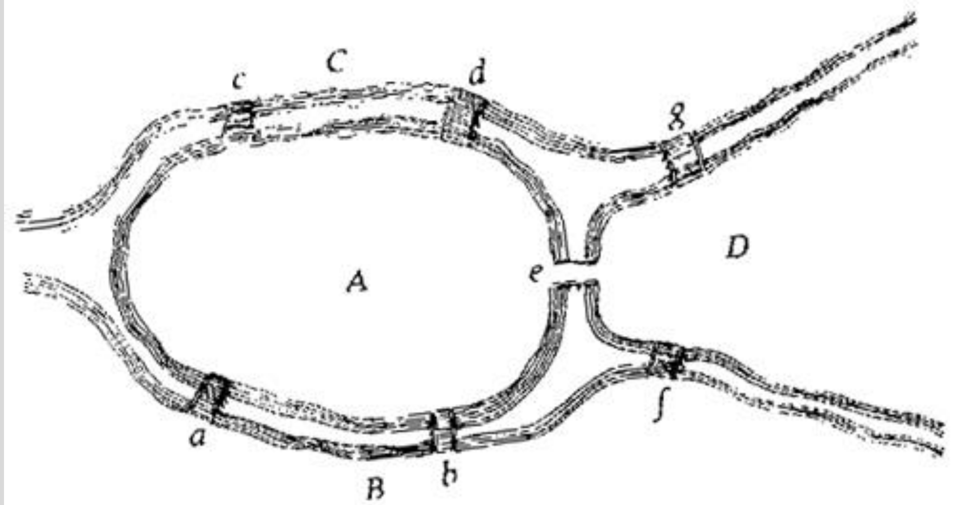
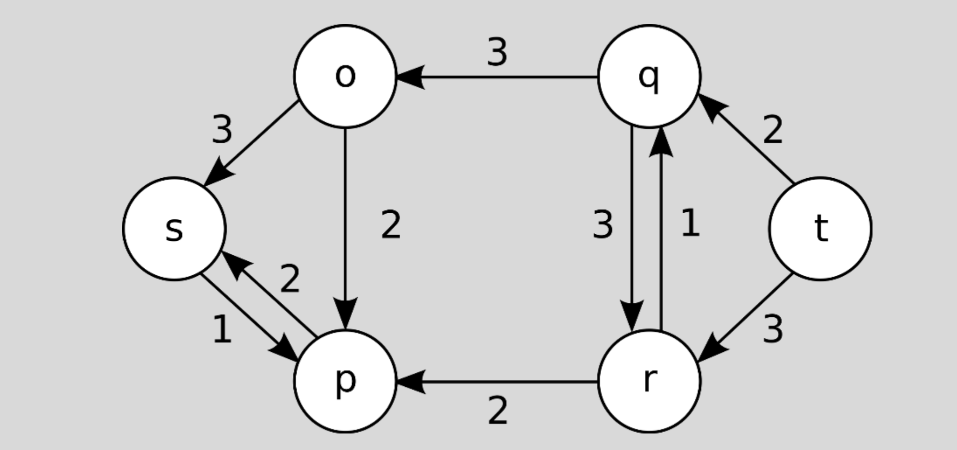
Lecture 9 –
Flow networks II





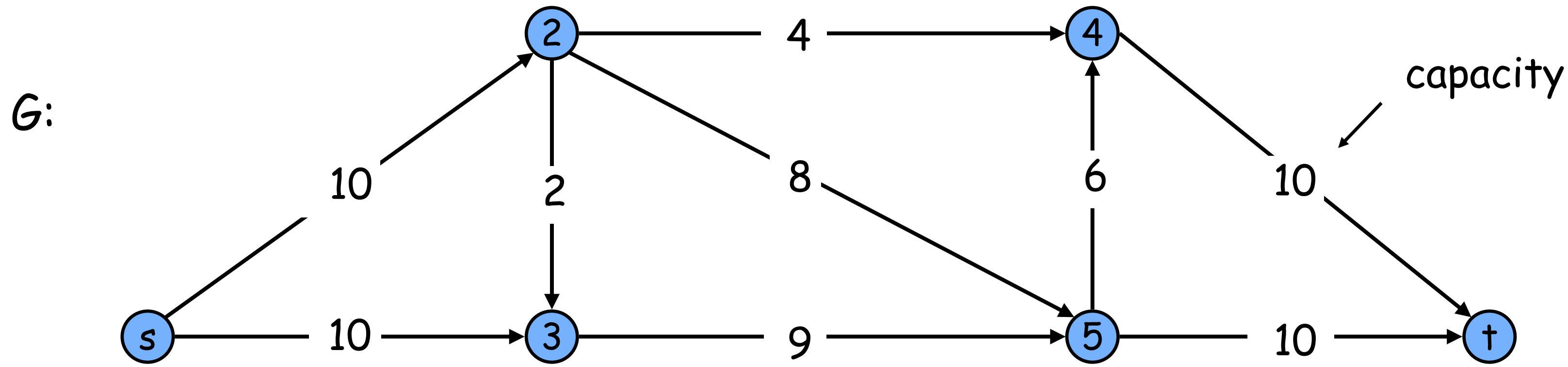
THE UNIVERSITY OF SYDNEY

Ford Fulkerson

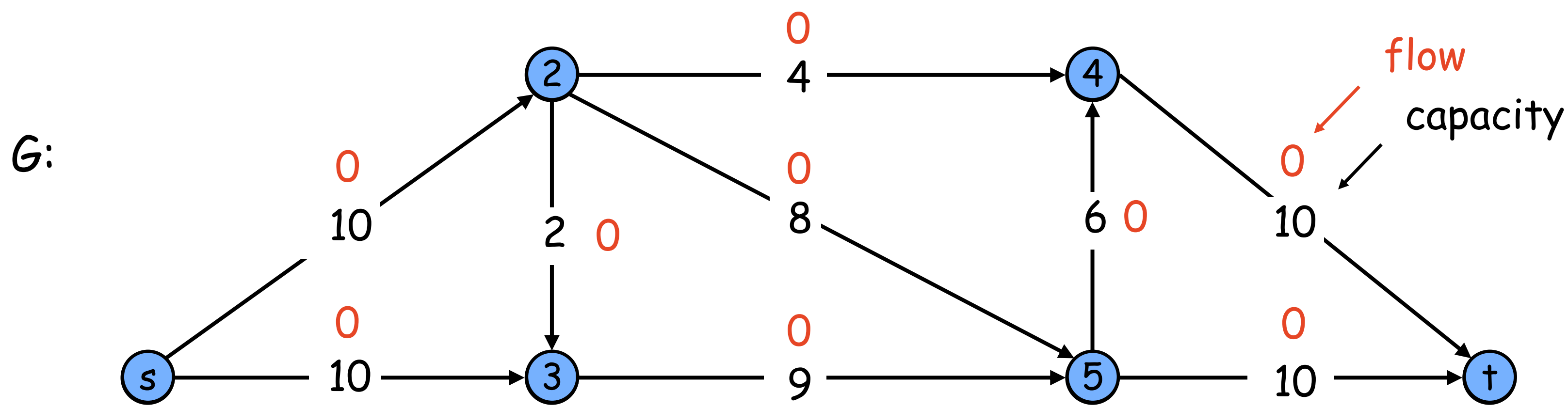
```
Ford-Fulkerson(G,s,t) {
   foreach e ∈ E
        f(e) ← 0
   Gf ← residual graph

   while (there exists augmenting path P in Gf){
      f ← Augment(f,P)
      update Gf
   }
   return f
}
```

Ford-Fulkerson Algorithm

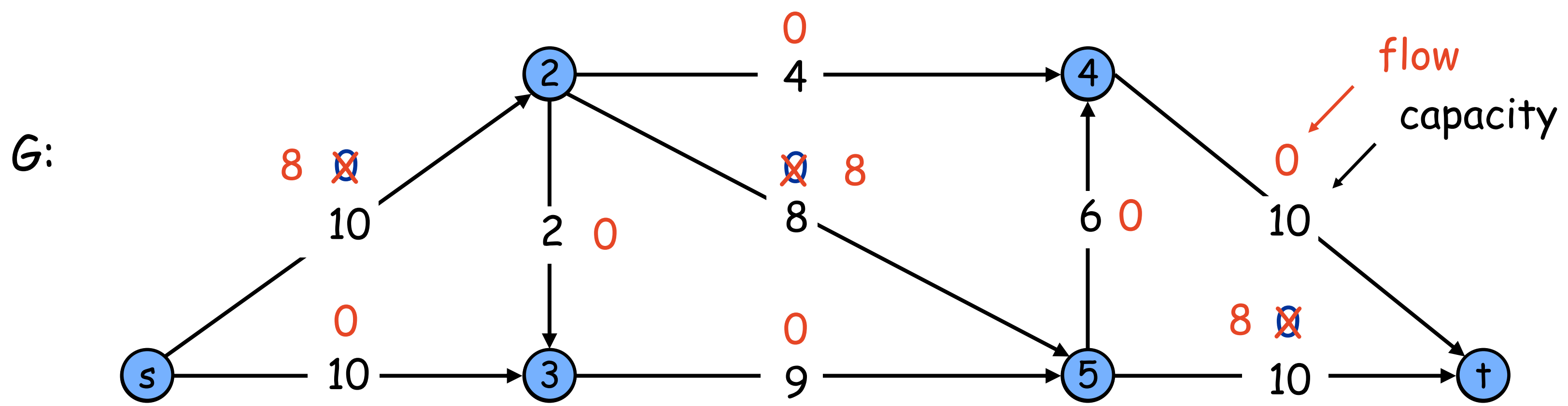


Ford-Fulkerson Algorithm

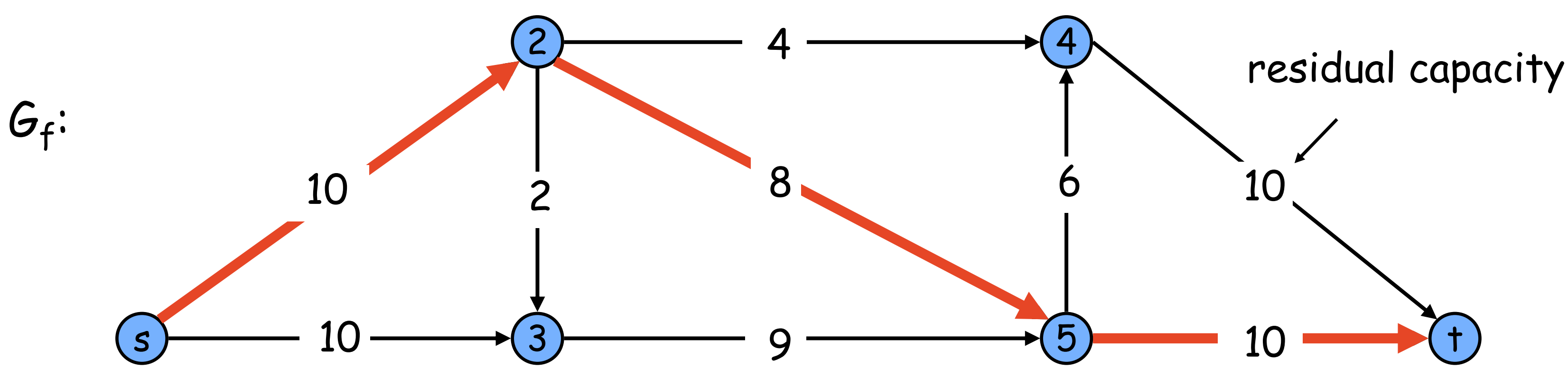


Flow value = 0

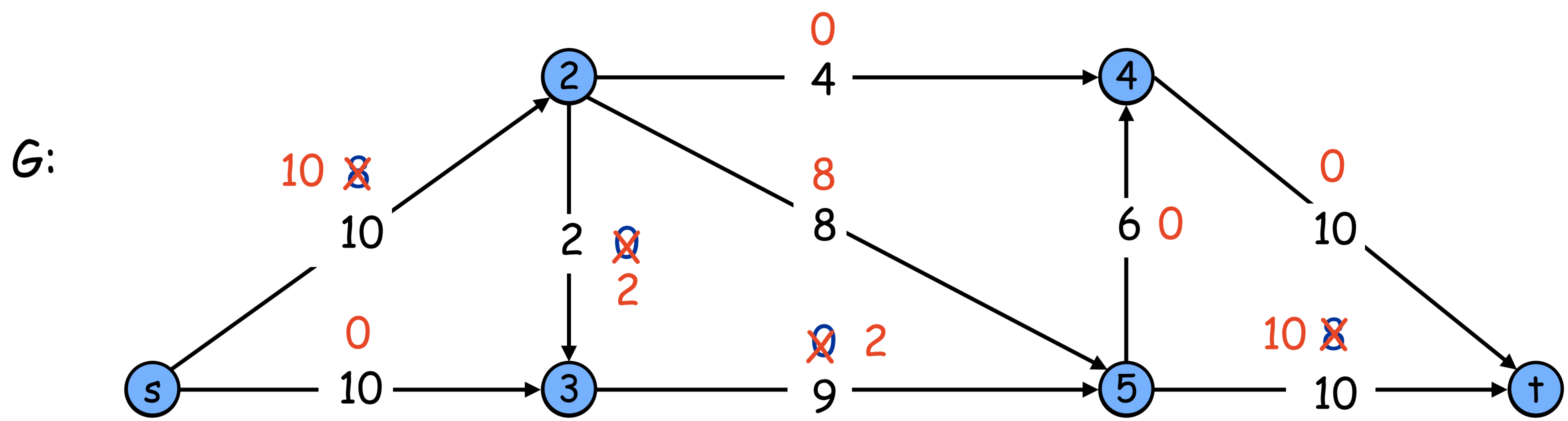
Ford-Fulkerson Algorithm



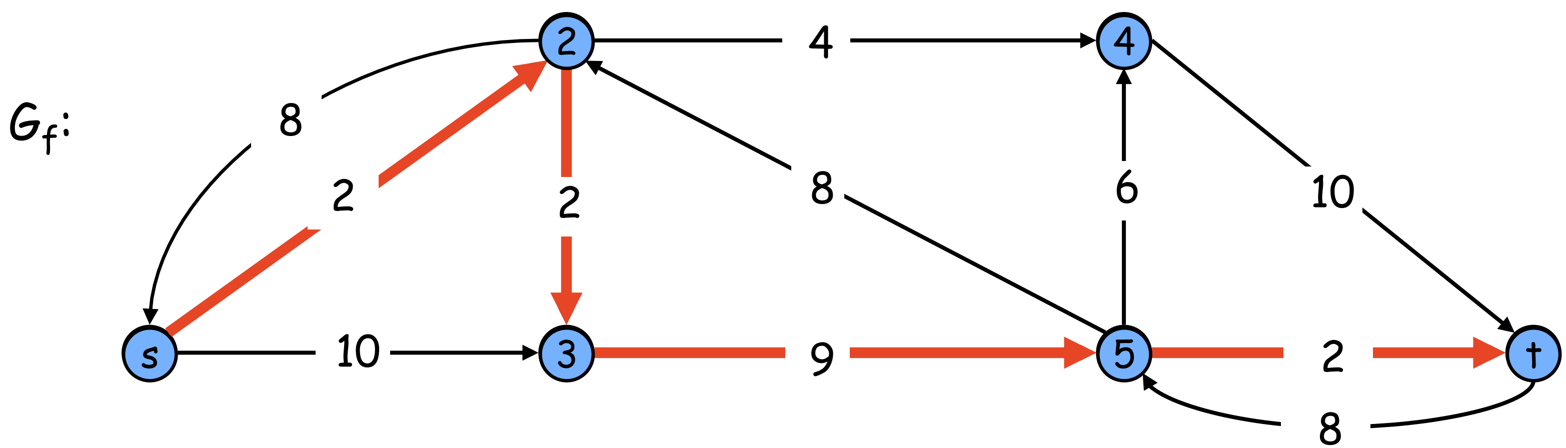
Flow value = 0



Ford-Fulkerson Algorithm

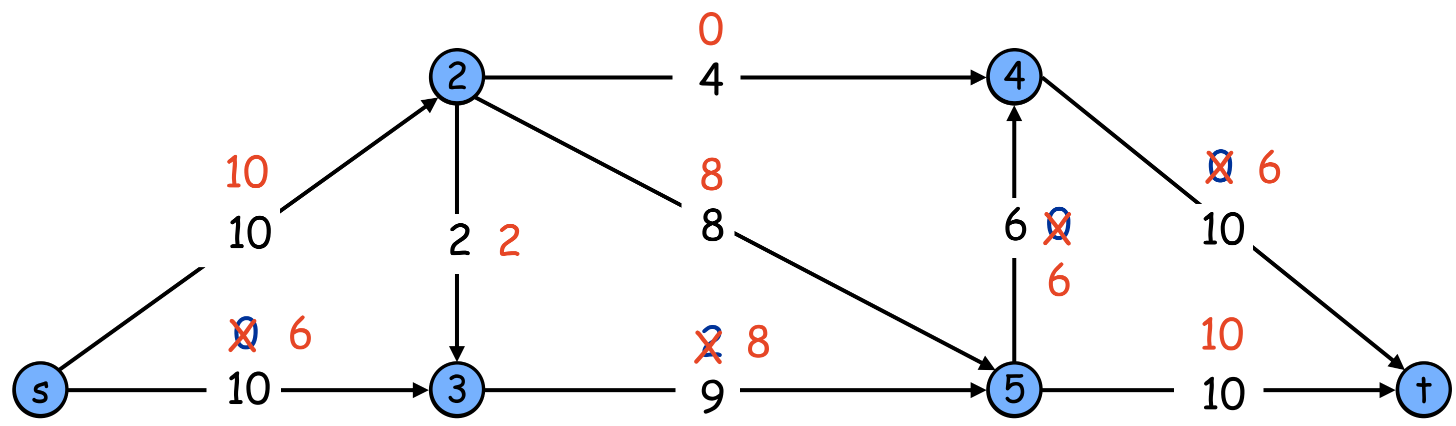


Flow value = 8



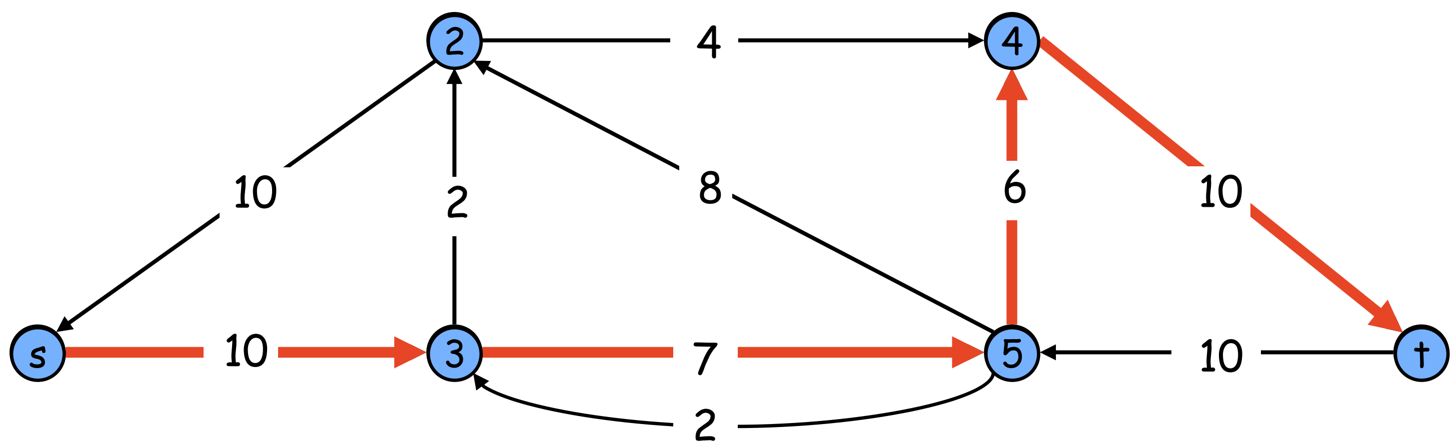
Ford-Fulkerson Algorithm

G:

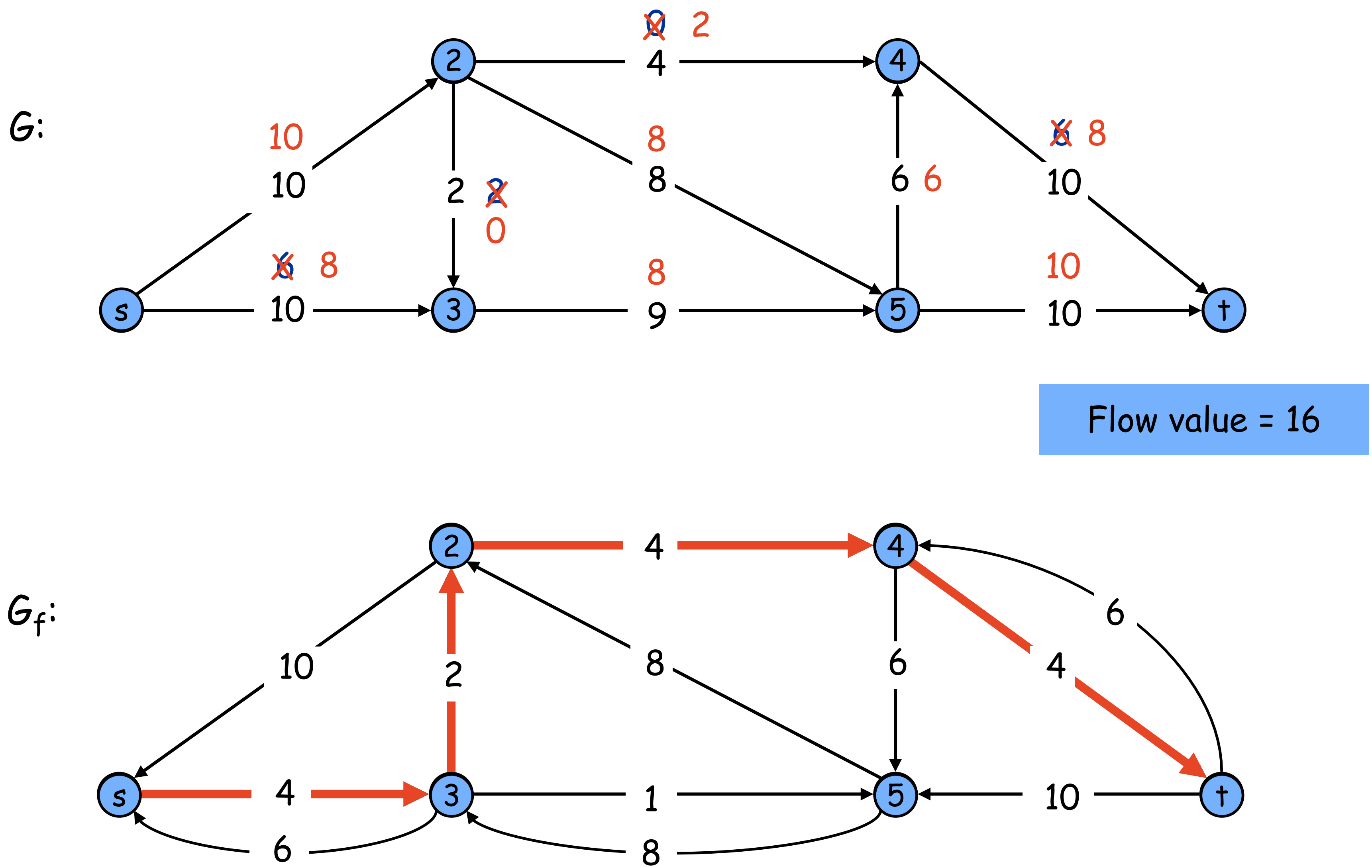


Flow value = 10

G_f:

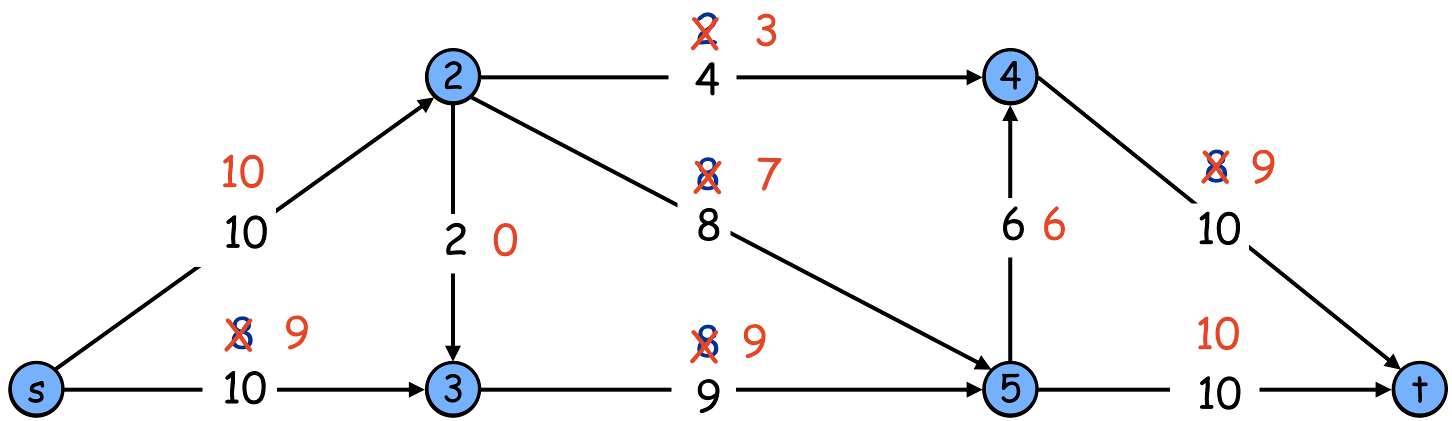


Ford-Fulkerson Algorithm



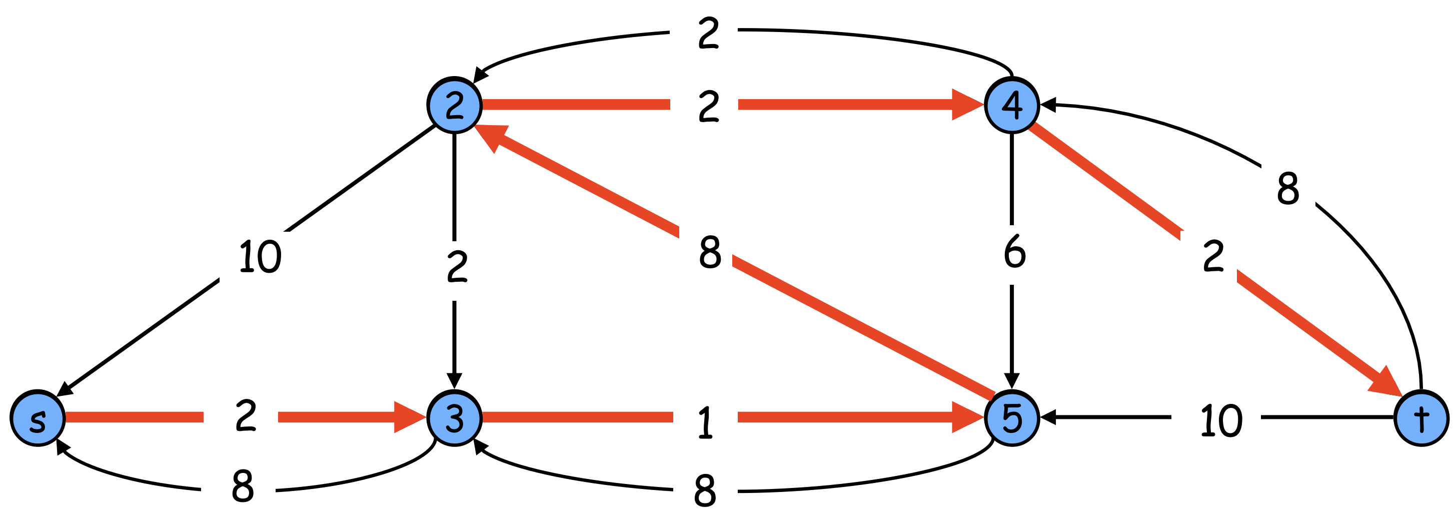
Ford-Fulkerson Algorithm

G:



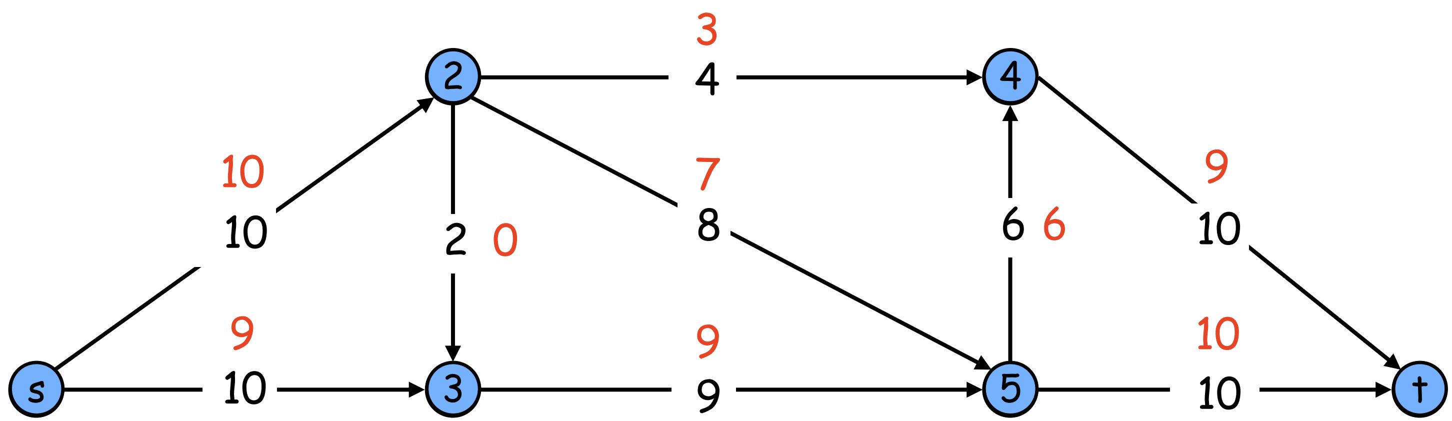
Flow value = 18

G_f:



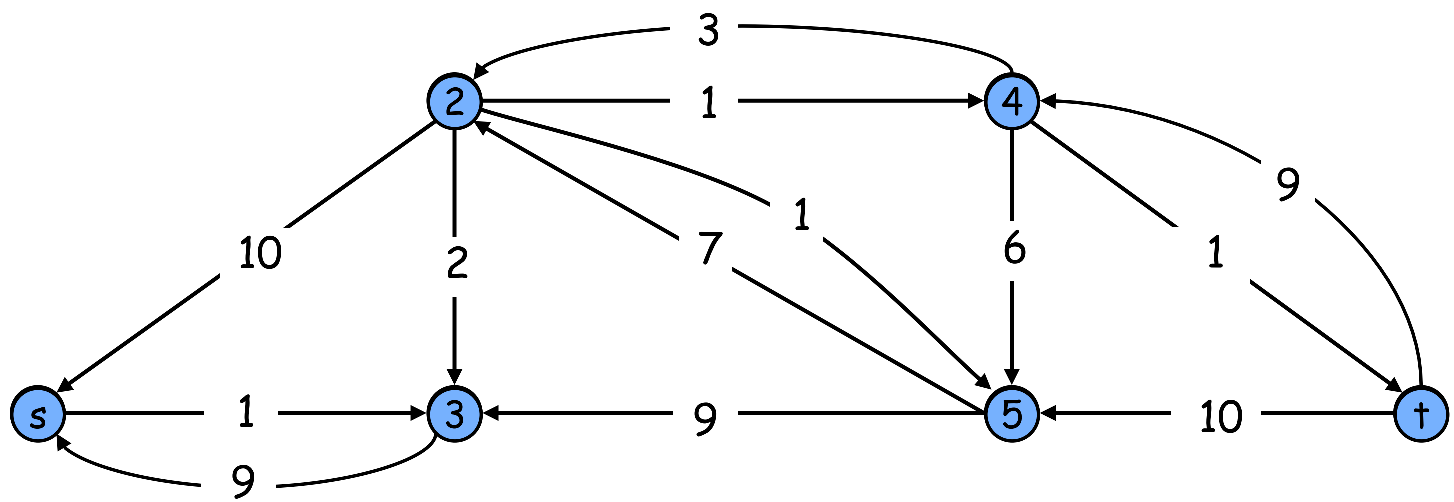
Ford-Fulkerson Algorithm

G:



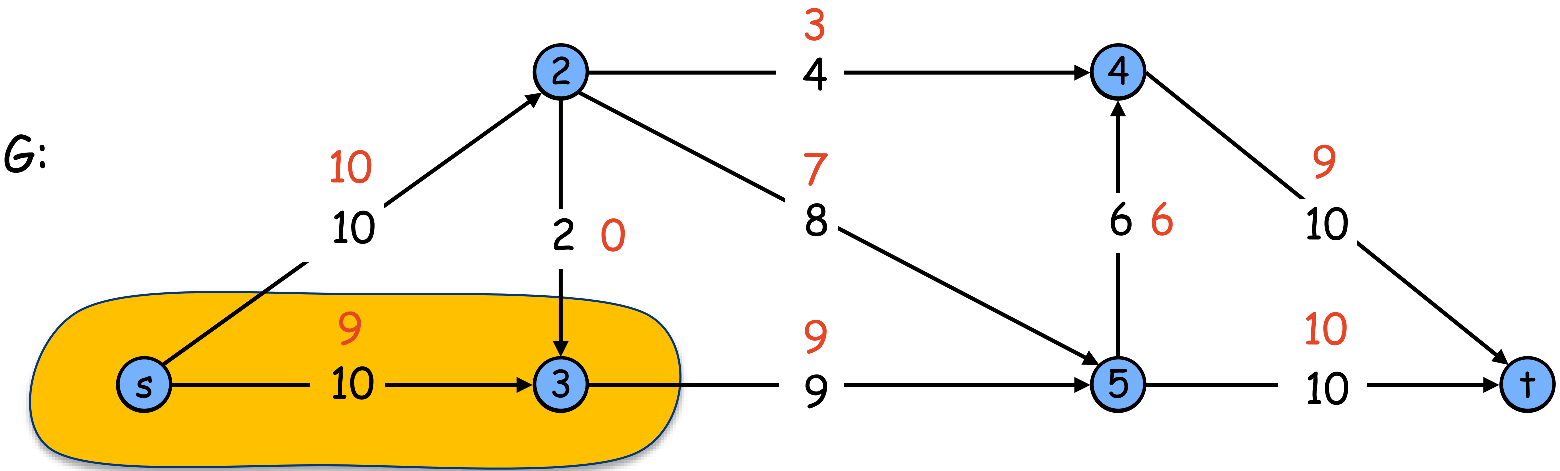
Flow value = 19

G_f:



Ford-Fulkerson Algorithm

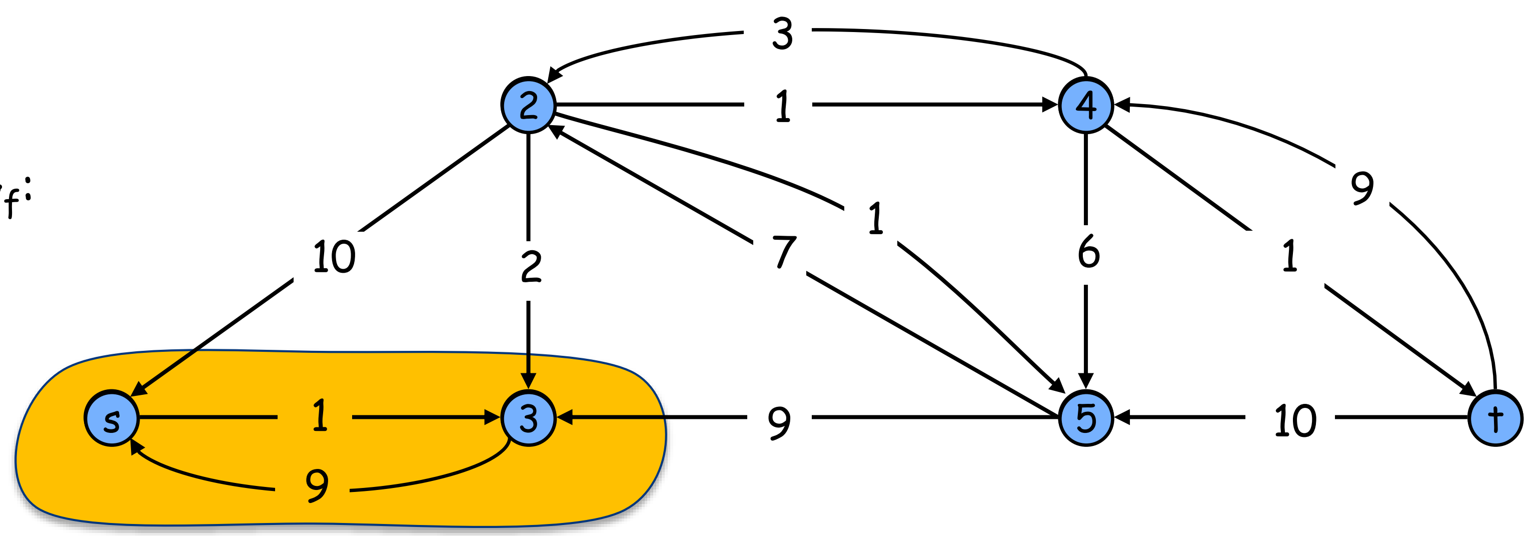
G:



Cut capacity = 19

Flow value = 19

G_f:



Augmenting Path Algorithm

```
Ford-Fulkerson(G,s,t) {
   foreach e ∈ E
        f(e) ← 0
   Gf ← residual graph

   while (there exists augmenting path P in Gf){
      f ← Augment(f,P)
      update Gf
   }
   return f
}
```

```
Augment(f,P) {
   b ← bottleneck(P,f)
   foreach e =(u,v) ∈ P {
      if e is a forward edge then
         increase f(e) in G by b
      else (e is a backward edge)
         decrease f(e) in G by b
   }
   return f
}
```

Ford-Fulkerson: Running Time

Observation:

Let f be a flow in G, and let P be a simple s-t path in G_f.

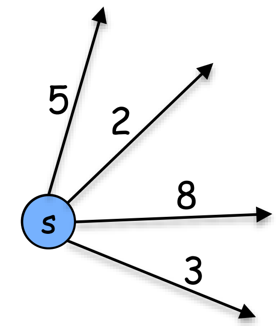
$$v(f') = v(f) + \text{bottleneck}(f,P)$$

and since bottleneck(f,P)>0

$$v(f') > v(f).$$

$\Rightarrow$ The flow value strictly increases in an augmentation

Ford-Fulkerson: Running Time



Notation: $C = \sum_{e \text{ out of } s} c(e)$

Observation: C is an upper bound on the maximum flow.

Theorem. The algorithm terminates in at most $v(f_{max}) \leq C$ iterations.

Proof: Each augmentation increase flow value by at least 1.

Ford-Fulkerson: Running Time

Corollary:

Ford-Fulkerson runs in $O((m+n)C)$ time, if all capacities are integers.

Proof: C iterations.

Path in G_f can be found in $O(m+n)$ time using BFS.

Augment(P,f) takes $O(n)$ time.

Updating G_f takes $O(m+n)$ time.

7.3 Choosing Good Augmenting Paths

Is $O(C(m+n))$ a good time bound?

- Yes, if C is small.
- If C is large, can the number of iterations be as bad as C?

Choosing Good Augmenting Paths

- Ford Fulkerson
  Choose any augmenting path (C iterations)

- Edmonds Karp #1 (m log F iterations)
  Choose max flow path

- Improved Ford Fulkerson (m log C iterations)
  Choose approximate max flow path [capacity scaling]

- Edmonds Karp #2 (nm iterations) [Edmonds-Karp 1972, Dinitz 1970]
  Choose minimum link path

Edmonds-Karp #1

Pick the augmenting path with largest capacity

[maximum bottleneck path]

Edmonds-Karp #1

Pick the augmenting path with largest capacity

[maximum bottleneck path]

Claim: If maximum flow in G is F, there must exists a path from s to t with capacity at least F/m.

Edmonds-Karp #1

Pick the augmenting path with largest capacity

[maximum bottleneck path]

Claim: If maximum flow in G is F, there must exists a path from s to t with capacity at least F/m.

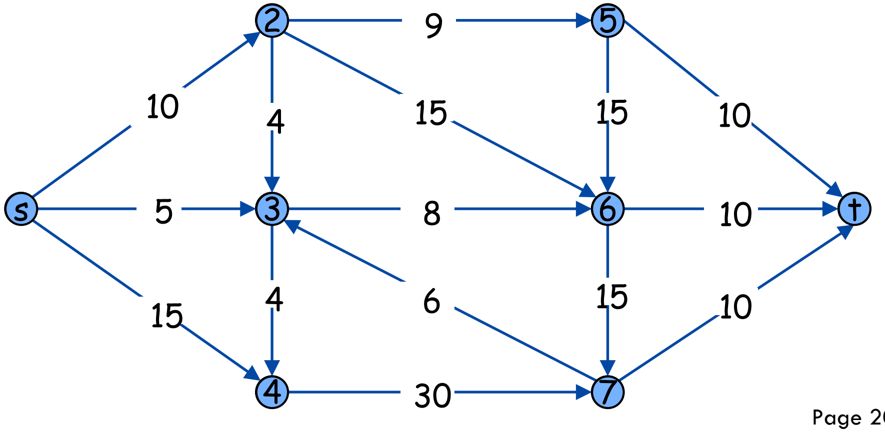
Proof:

Delete all edges of capacity less than F/m.

Is the graph still connected?

F=24

m=15



Edmonds-Karp #1

Pick the augmenting path with largest capacity

[maximum bottleneck path]

Claim: If maximum flow in G is F, there must exists a path from s to t with capacity at least F/m.

Proof:

Delete all edges of capacity less than F/m.

Is the graph still connected?

Yes, otherwise we have a cut of value less than F.
The remaining graph must have a path from s to t and since all edges have capacity at least F/m, the path itself has capacity at least F/m.

Edmonds-Karp #1

Theorem: Edmonds-Karp #1 makes at most $O(m \log F)$ iterations.

Proof:

At least $1/m$ of remaining flow is added in each iteration.

$\Leftrightarrow$

Remaining flow reduced by a factor of $(1-1/m)$ per iteration.

#iterations until remaining flow <1? $\Rightarrow$ $F \cdot (1-1/m)^x < 1$?

We know: $(1-1/m)^m < 1/e$

Set $x = m \ln F$ $\Rightarrow$ $F \cdot (1-1/m)^{m \ln F} < F \cdot (1/e)^{\ln F} < 1$

Choosing Good Augmenting Paths

- Ford Fulkerson
  Choose any augmenting path (C iterations)

- Edmonds Karp #1 (m log F iterations)
  Choose max flow path

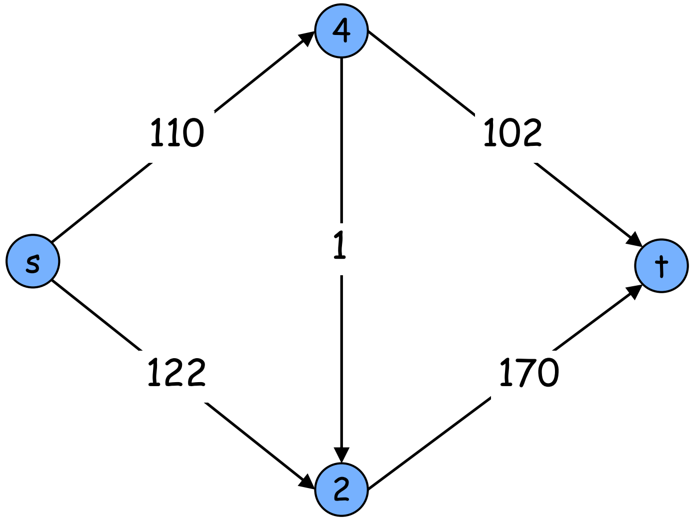
- Improved Ford Fulkerson (m log C iterations)
  Choose approximate max flow path [capacity scaling]

- Edmonds Karp #2 (nm iterations) [Edmonds-Karp 1972, Dinitz 1970]
  Choose minimum link path

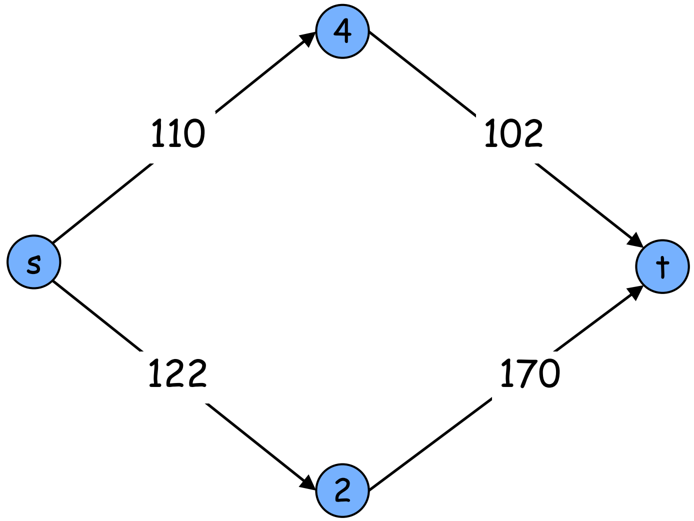
Capacity Scaling

Intuition. Choosing path with highest bottleneck capacity increases flow by max possible amount.

- Don't worry about finding exact highest bottleneck path.
- Maintain scaling parameter Δ.
- Let $G_f(\Delta)$ be the subgraph of the residual graph consisting of only arcs with capacity at least Δ.



G_f



$G_f(100)$

Capacity Scaling

```
Scaling-Max-Flow(G, s, t) {
   foreach e ∈ E
        f(e) ← 0
   Δ ← smallest power of 2 greater than or equal to C
   Gf ← residual graph

   while (Δ ≥ 1) {
      Gf(Δ) ← Δ-residual graph
      while (there exists augmenting path P in Gf(Δ)) {
         f ← augment(f, c, P)
         update Gf(Δ)
      }
      Δ ← Δ / 2
   }
   return f
}
```

Capacity Scaling: Correctness

- Assumption. All edge capacities are integers between 1 and C.

- Integrality invariant. All flow and residual capacity values are integral.

- Correctness. If the algorithm terminates, then f is a max flow.
  Proof:
  - By integrality invariant, when $\Delta = 1 \Rightarrow G_f(\Delta) = G_f$.
  - Upon termination of $\Delta = 1$ phase, there are no augmenting paths. ▪

Capacity Scaling: Running time

```
Scaling-Max-Flow(G, s, t) {
   foreach e ∈ E
        f(e) ← 0
   Δ ← smallest power of 2 greater than or equal to C
   G_f ← residual graph

   while (Δ ≥ 1) {
      G_f(Δ) ← Δ-residual graph
      while (there exists augmenting path P in G_f(Δ)) {
         f ← augment(f, c, P)
         update G_f(Δ)
      }
      Δ ← Δ / 2
   }
   return f
}
```

Capacity Scaling: Running time

Lemma 1: The outer while loop repeats $1 + \lceil \log_2 C \rceil$ times.

Proof: Initially $C \leq \Delta < 2C$. Δ decreases by a factor of 2 in each iteration. ▪

Observation: During the Δ–scaling phase each augmentation increases the flow value by at least Δ.

Capacity Scaling: Running time

```
Scaling-Max-Flow(G, s, t) {
   foreach e ∈ E
       f(e) ← 0
   Δ ← smallest power of 2 greater than or equal to C
   Gf ← residual graph

   while (Δ ≥ 1) {
      Gf(Δ) ← Δ-residual graph
      while (there exists augmenting path P in Gf(Δ)) {
         f ← augment(f, c, P)
         update Gf(Δ)
      }
      Δ ← Δ / 2
   }
   return f
}
```

log C

?

O(m) since m>n

Capacity Scaling: Running time

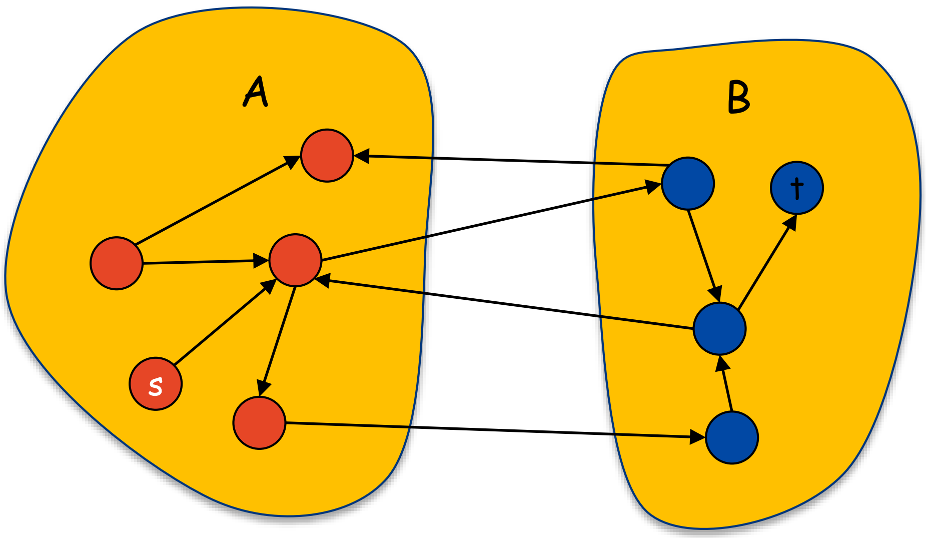
Lemma 2. Let f be the flow at the end of a Δ-scaling phase. Then value of the maximum flow is at most $v(f) + m\,\Delta$.

Proof: (similar to proof of max-flow min-cut theorem)

- We show that at the end of a Δ-phase, there exists a cut (A, B) such that $cap(A, B) \leq v(f) + m\,\Delta$.
- Choose A to be the set of nodes reachable from s in $G_f(\Delta)$.
- By definition of A, $s \in A$.
- By definition of f, $t \notin A$.

$$
\begin{aligned}
v(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\
&> \sum_{e \text{ out of } A} (c(e)-\Delta) - \sum_{e \text{ in to } A} \Delta \\
&= \sum_{e \text{ out of } A} c(e) - \sum_{e \text{ out of } A} \Delta - \sum_{e \text{ in to } A} \Delta \\
&\geq cap(A,B) - m\Delta
\end{aligned}
$$

c(e)<f(e)+Δ
and
f(e)<Δ



original network

Capacity Scaling: Running time

Lemma 3. There are at most 2m augmentations per scaling phase.

- Let f be the flow at the end of the previous scaling phase.
- Lemma 2 $\Rightarrow$ $v(f^*) \leq v(f) + m\,(2\Delta)$.
- Each augmentation in a Δ-phase increases v(f) by at least Δ. ▪

Theorem. The scaling max-flow algorithm finds a max flow in $O(m \log C)$ augmentations. It can be implemented to run in $O(m^2 \log C)$ time.

Capacity Scaling: Running time

Theorem. The scaling max-flow algorithm finds a max flow in $O(m \log C)$ augmentations. It can be implemented to run in $O(m^2 \log C)$ time. ▪

```
Scaling-Max-Flow(G, s, t) {
   foreach e ∈ E
      f(e) ← 0
   Δ ← smallest power of 2 greater than or equal to C
   Gf ← residual graph

   while (Δ ≥ 1) {
      Gf(Δ) ← Δ-residual graph
      while (there exists augmenting path P in Gf(Δ)) {
         f ← augment(f, c, P)
         update Gf(Δ)
      }
      Δ ← Δ / 2
   }
   return f
}
```

log C
(Lemma 1)

2m
(Lemma 3)

O(m) since m>n

Choosing Good Augmenting Paths

- Ford Fulkerson
  Choose any augmenting path (C iterations)

- Edmonds Karp #1 (m log F iterations)
  Choose max flow path

- Improved Ford Fulkerson (m log C iterations)
  Choose approximate max flow path [capacity scaling]

- Edmonds Karp #2 (nm iterations) [Edmonds-Karp 1972, Dinitz 1970]
  Choose minimum link path

Edmonds-Karp #2

Pick the augmenting path smallest number of edges.

How do we find such a path?

Use BFS – running time $O(n+m)$

Edmonds-Karp #2

Pick the augmenting path smallest number of edges.

Theorem: Edmonds-Karp #2 makes at most nm iterations.

Proof idea:

Let d be the distance from s to t in the current residual graph.

1. d never decreases
2. Every m iterations, d has to increase by at least 1

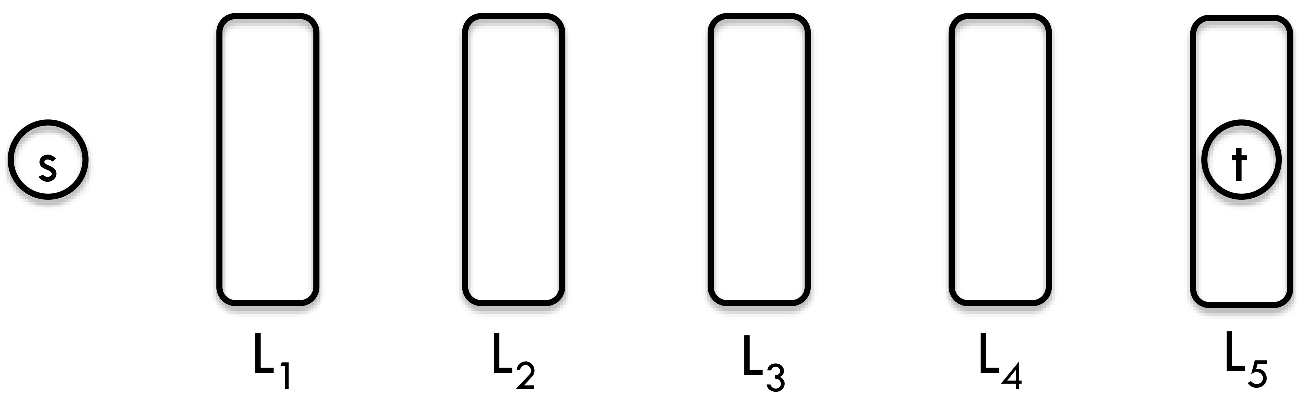
   [which can happen at least m times]

Edmonds-Karp #2

Pick the augmenting path smallest number of edges.

Lemma 1: Step 1- d never decreases

Proof idea: Consider the BFS levels starting from s.

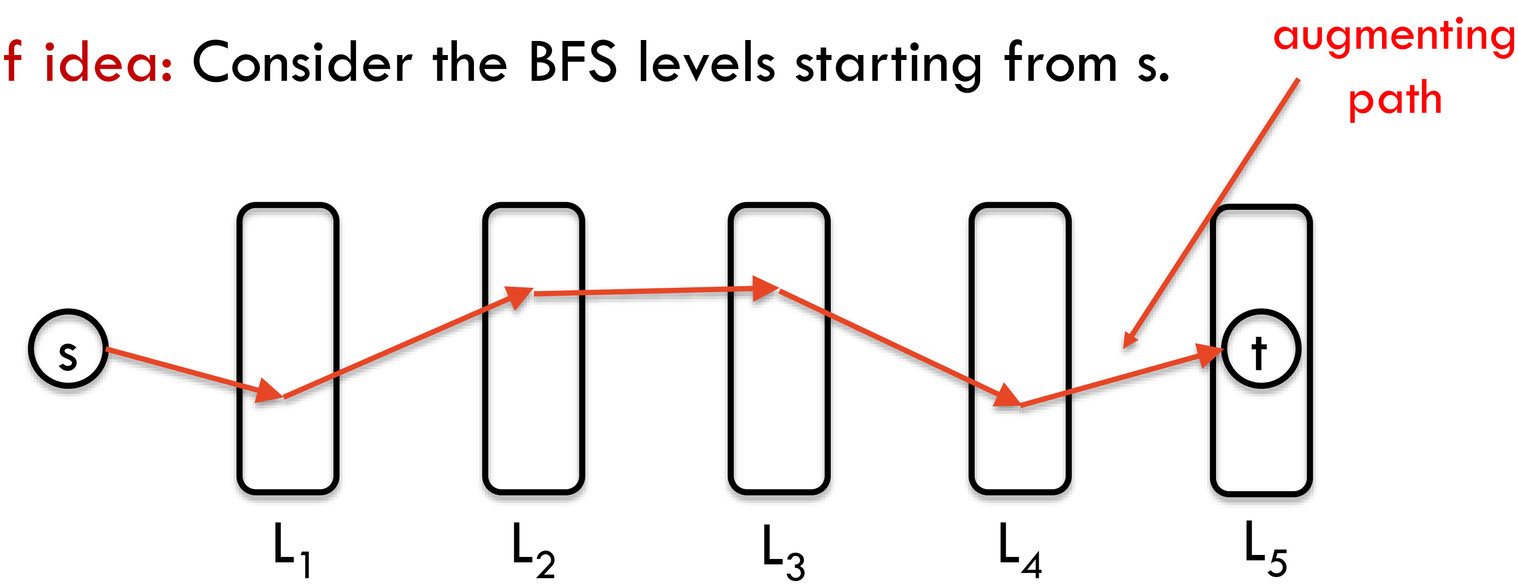


Edmonds-Karp #2

Pick the augmenting path smallest number of edges.

Lemma 1: Step 1- d never decreases

Proof idea: Consider the BFS levels starting from s.



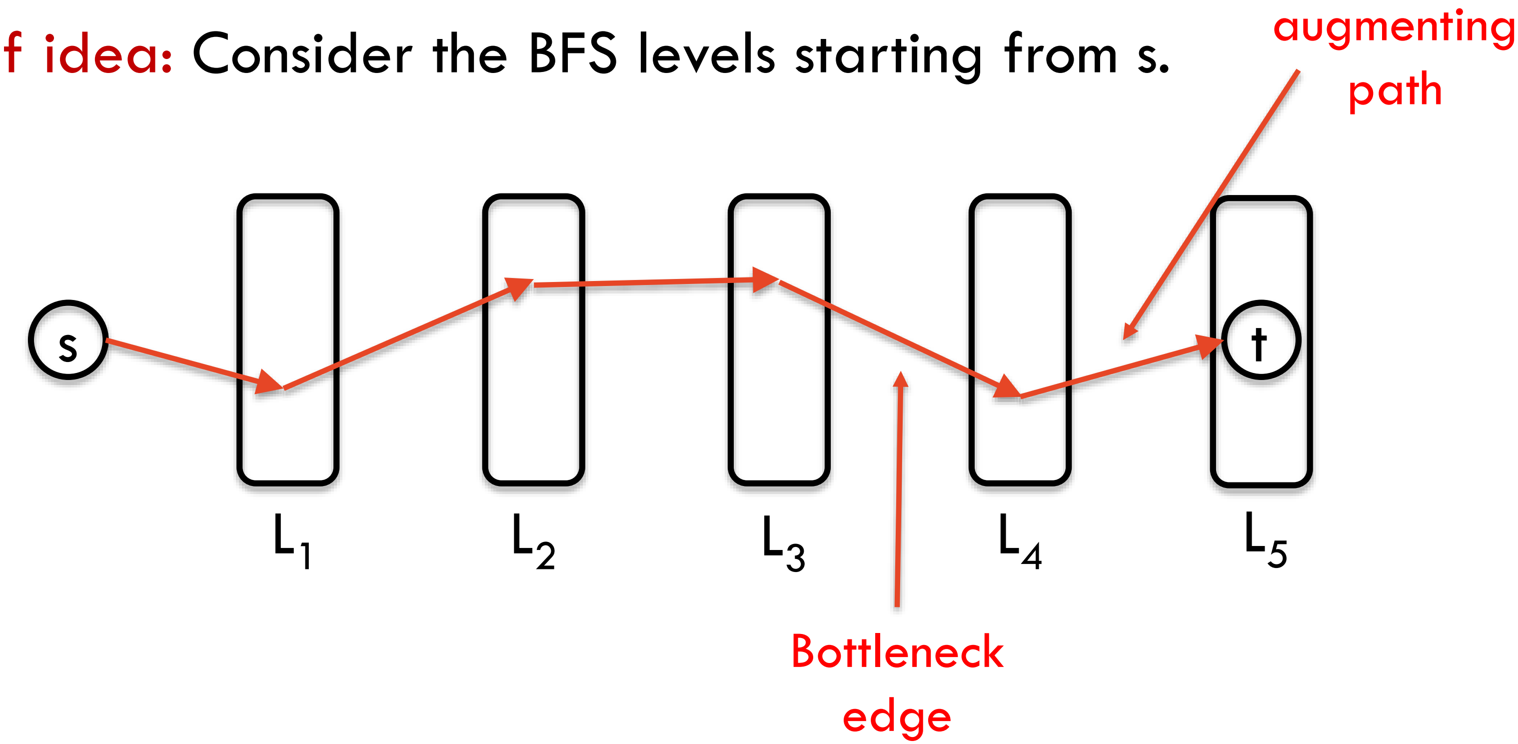
What happens when we augment flow with an st-path?

Edmonds-Karp #2

Pick the augmenting path smallest number of edges.

Lemma 1: Step 1- d never decreases

Proof idea: Consider the BFS levels starting from s.

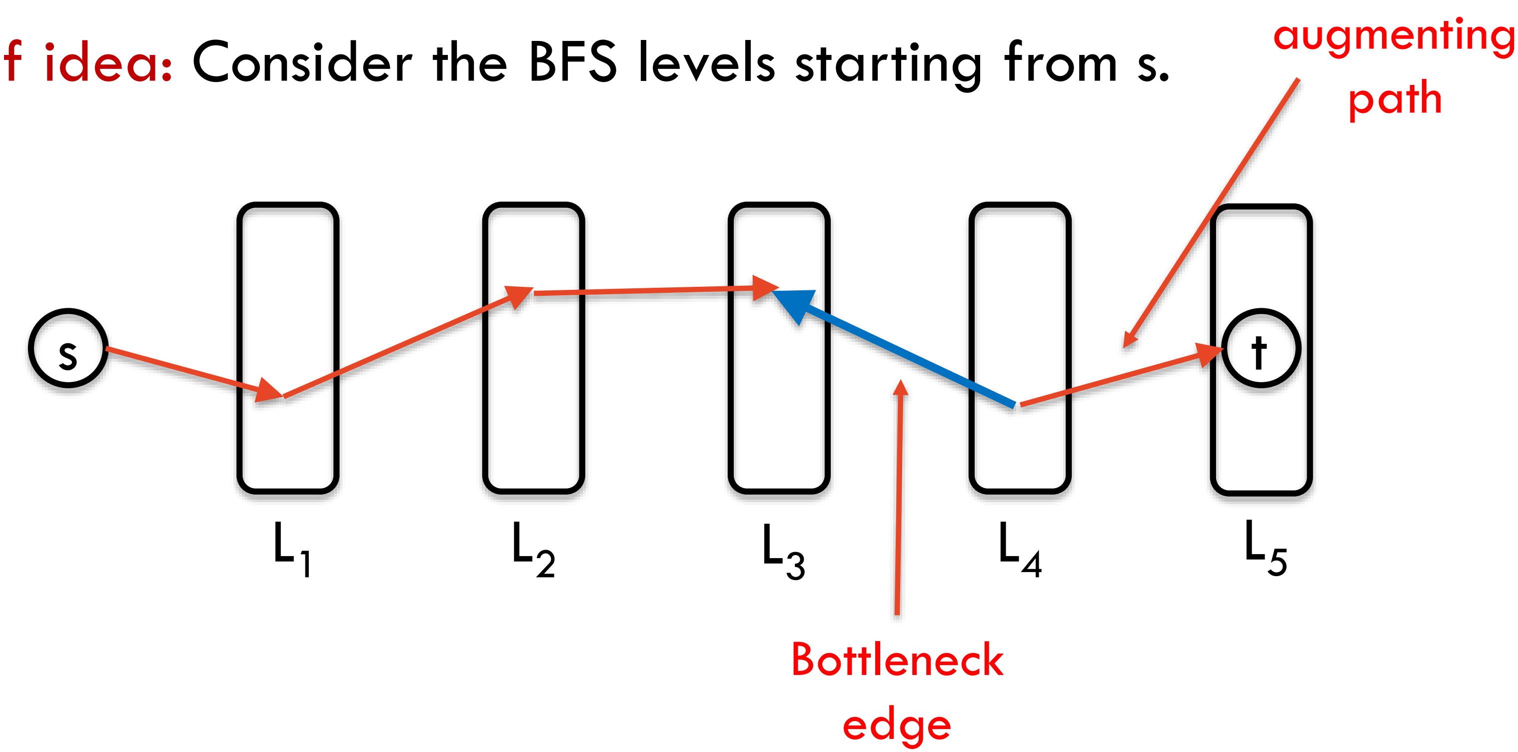


Edmonds-Karp #2

Pick the augmenting path smallest number of edges.

Lemma 1: Step 1- d never decreases

Proof idea: Consider the BFS levels starting from s.

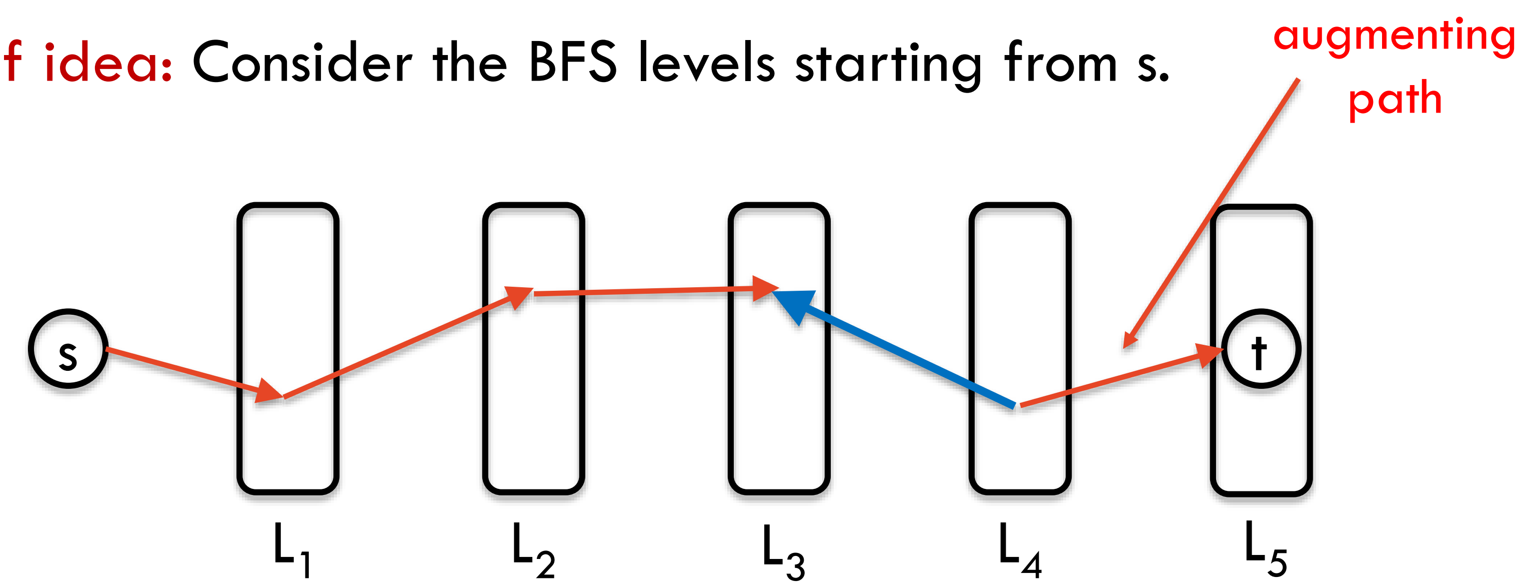


Edmonds-Karp #2

Pick the augmenting path smallest number of edges.

Lemma 1: Step 1- d never decreases

Proof idea: Consider the BFS levels starting from s.



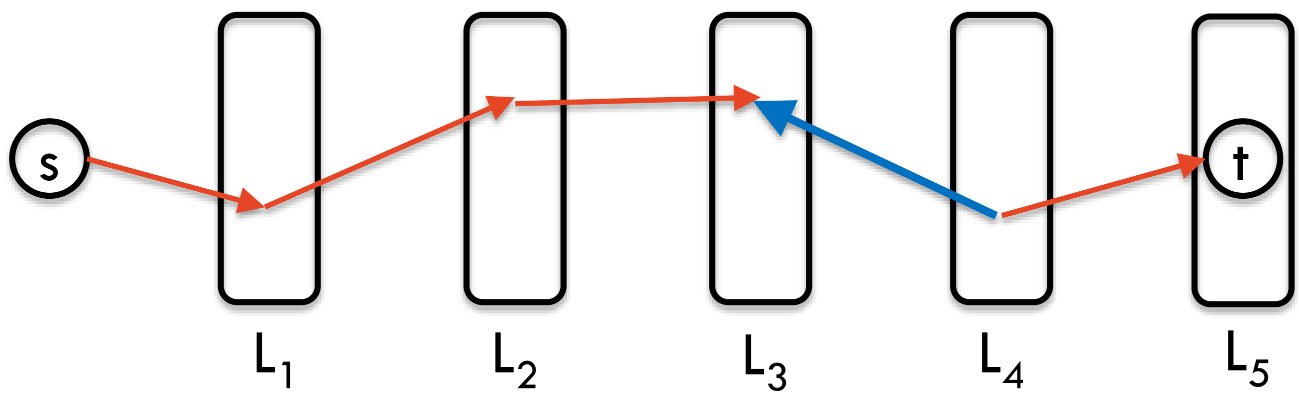
$\Rightarrow$ d can never decrease

Edmonds-Karp #2

Pick the augmenting path smallest number of edges.

Lemma 2: Every m iterations, d has to increase by at least 1

Proof idea:



What happens if d does not increase?

If d did not increase then ≥ 1 forward edge removed

How many time can this happen? m times!

Edmonds-Karp #2

Pick the augmenting path smallest number of edges.

Theorem: Edmonds-Karp #2 makes at most nm iterations.

Proof idea:

Let d be the distance from s to t in the current residual graph.

1. d never decreases [Lemma 1]
2. Every m iterations, d has to increase by at least 1 [Lemma 2]

   [which can happen at most n times]

Done!

Summary

1. Max flow problem
2. Min cut problem
3. Ford-Fulkerson:
   1. Residual graph
   2. correctness
   3. complexity
4. Max-Flow Min-Cut theorem
5. Capacity scaling
6. Edmonds-Karp