v1.0: Seth Gilbert (2 parts)
v1.6: Steven Halim (merged into 1+live demo)

CS4234
Optimiz(s)ation Algorithms

L5 – `Max-Flow/Min-Cut` + `Analysis`

They are Primal-Dual LPs

Please read all e-Lecture slides at
https://visualgo.net/en/maxflow?slide=1 to end
before attending this class

Types of Graph Problems (so far)

0. Connectivity: *How is my network connected?*

- P: (Strongly) Connected Component , https://visualgo.net/en/dfsbfs

1. Distances: *How to get from here to there?*

- P: Single-Source Shortest Paths, https://visualgo.net/en/sssp
- P: All-Pairs Shortest Paths not just Floyd-Warshall :O

2. Spanning Trees: *How do I design a network?*

- P: Min-Spanning-Tree, https://visualgo.net/en/mst
- NP-hard: Steiner-Tree, https://visualgo.net/en/steinertree
  (A TSP cycle – an edge = A Spanning Tree)
- NP-hard: Travelling-Salesman, https://visualgo.net/en/tsp

3. Network Flows: *How is my network connected?*

- Our topic today

Not all graph problems are in P
Not all graph problems are NP-hard/complete
Not all optimization problems are graph problems

Roadmap (Flipped Classroom)

Network Flows

a. **Definition (with VA, quick recap)**
b. Ford-Fulkerson Algorithm (with VA)
c. Max-Flow/Min-Cut Theorem
d. Ford-Fulkerson (FF) Analysis
    a. Analysis of Basic **Ford-Fulkerson**: $O(m^2 U)$
    b. FF with Shortest-Path v1/**Edmonds-Karp**: $O(m^2 n)$
    c. FF with Shortest-Path v2/**Dinic's**: $O(m n^2)$
e. Live solve a (simple) Max Flow problem

Roadmap (Flipped Classroom)

Network Flows

a. Definition (with VA)

b. Ford-Fulkerson Algorithm (with VA)

c. Max-Flow/Min-Cut Theorem

d. Ford-Fulkerson (FF) Analysis

   a. Analysis of Basic **Ford-Fulkerson**: $O(m^2 U)$

   b. FF with Shortest-Path v1/**Edmonds-Karp**: $O(m^2 n)$

   c. FF with Shortest-Path v2/**Dinic's**: $O(m n^2)$

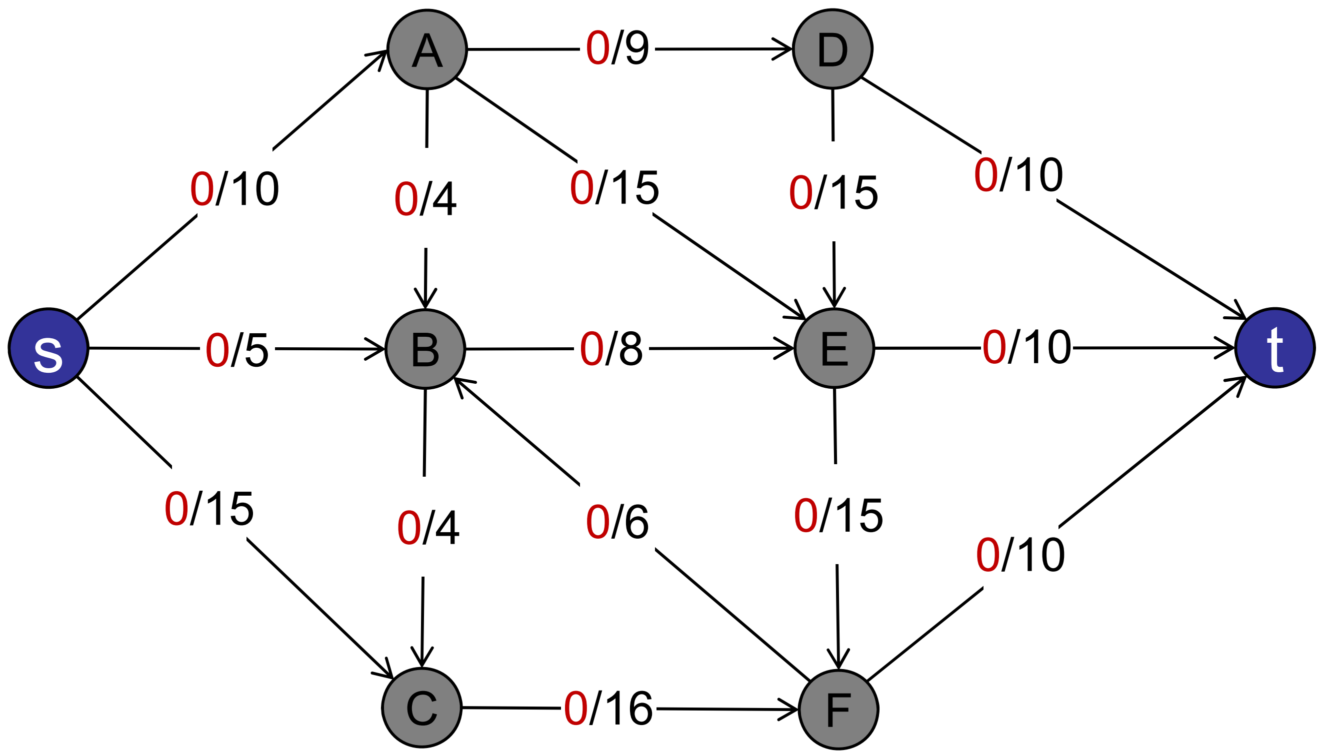
e. Live solve a (simple) Max Flow problem

Ford-Fulkerson (1)

Initially:

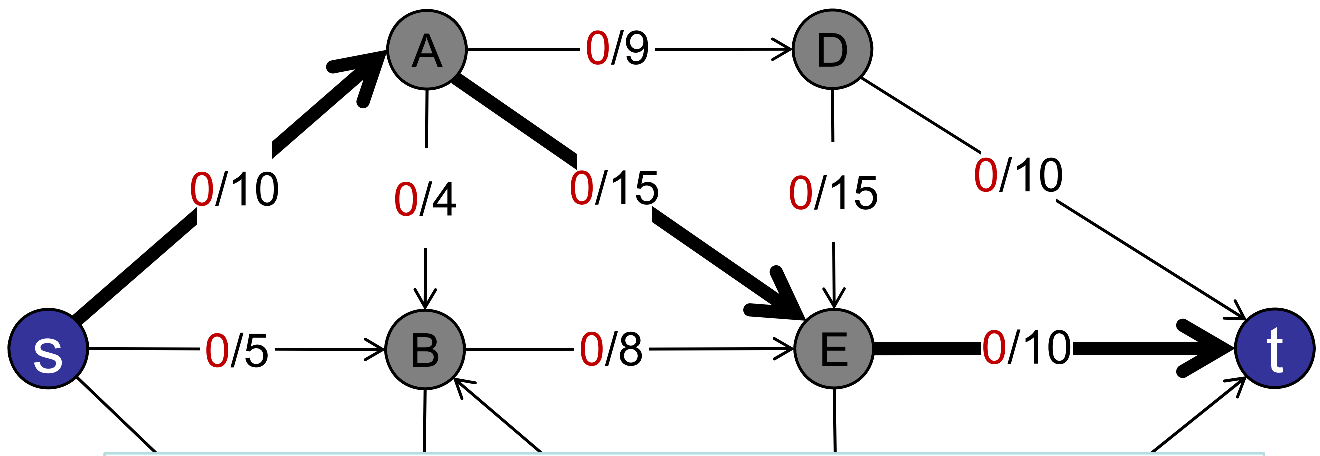
All flows are 0.

value = 0



Ford-Fulkerson (2)

Idea: Find an <u>augmenting path</u> (*path from s to t that goes through edges with positive weight residual capacity (c(e)-f(e)) left*) along which we can increase the flow.



value = 0

<u>Watch the animation live in VisuAlgo</u>
(chosen augmenting paths may be slightly different but the final answer is the same, max flow = 28)
(you can also try other flow graphs)

Ford-Fulkerson (Basic Idea)

Ford-Fulkerson Algorithm
Start with 0 flow. While there exists an augmenting path: // iterative algorithm • Find an augmenting path. • Compute bottleneck capacity. • Increase flow on the path by the bottleneck capacity.

There are still a few missing details:

- How to find an augmenting path?
- If it terminates, does it always find a max-flow?
- Does Ford-Fulkerson always terminate? How fast?

How best to find an augmenting path in the residual graph?

For now, any O(V+E) graph traversal algorithm will do (BFS, DFS, 'fattest path first', etc.)

Any path from s→t in the residual graph is an augmenting path.

We will learn more about this later

Ford-Fulkerson (More Complete)

Ford-Fulkerson Algorithm

Start with 0 flow.

Build residual graph:

- For every edge (u,v) add edge (u,v) with w(u,v) = capacity.
- For every edge (u,v) add (a new) edge (v,u) with w(v,u) = 0.

While there exists an augmenting path:

- Find an augmenting path **via DFS (the 'wrong one first')** in residual graph.
- Compute bottleneck capacity.
- Increase flow on the path by the bottleneck capacity:
    - For every edge (u,v) on the path, subtract the flow from w(u,v).
    - For every edge (u,v) on the path, add the flow to w(v,u).

Compute final flow by inverting residual flows.

Be careful of potential bug(s) here

Ford-Fulkerson

Ford-Fulkerson Algorithm
Start with 0 flow. While there exists an augmenting path: • Find an augmenting path. • Compute bottleneck capacity. • Increase flow on the path by the bottleneck capacity.

Details:

- ✓ How to find an augmenting path?
- – If it terminates, does it always find a max-flow?
- – Does Ford-Fulkerson always terminate? How fast?

Roadmap (Flipped Classroom)

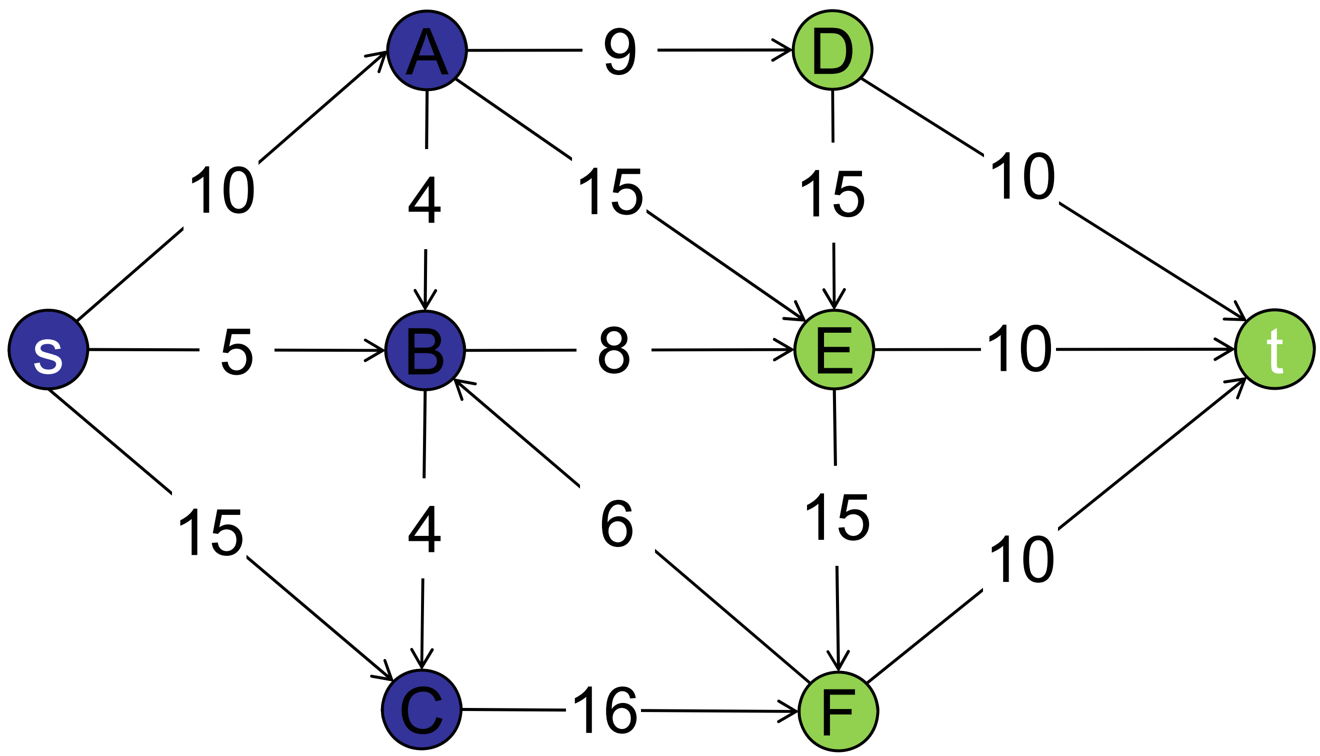
Network Flows

a. Definition (with VA)
b. Ford-Fulkerson Algorithm (with VA)
c. **Max-Flow/Min-Cut Theorem**
d. Ford-Fulkerson (FF) Analysis
    a. Analysis of Basic **Ford-Fulkerson**: $O(m^2 U)$
    b. FF with Shortest-Path v1/**Edmonds-Karp**: $O(m^2 n)$
    c. FF with Shortest-Path v2/**Dinic's**: $O(m n^2)$
e. Live solve a (simple) Max Flow problem

Cuts and Flows – Definition (1/4)

Definition:

An <u>st-cut</u> partitions the vertices of a graph into two disjoint sets S and T where source s $\in$ S and sink t $\in$ T.

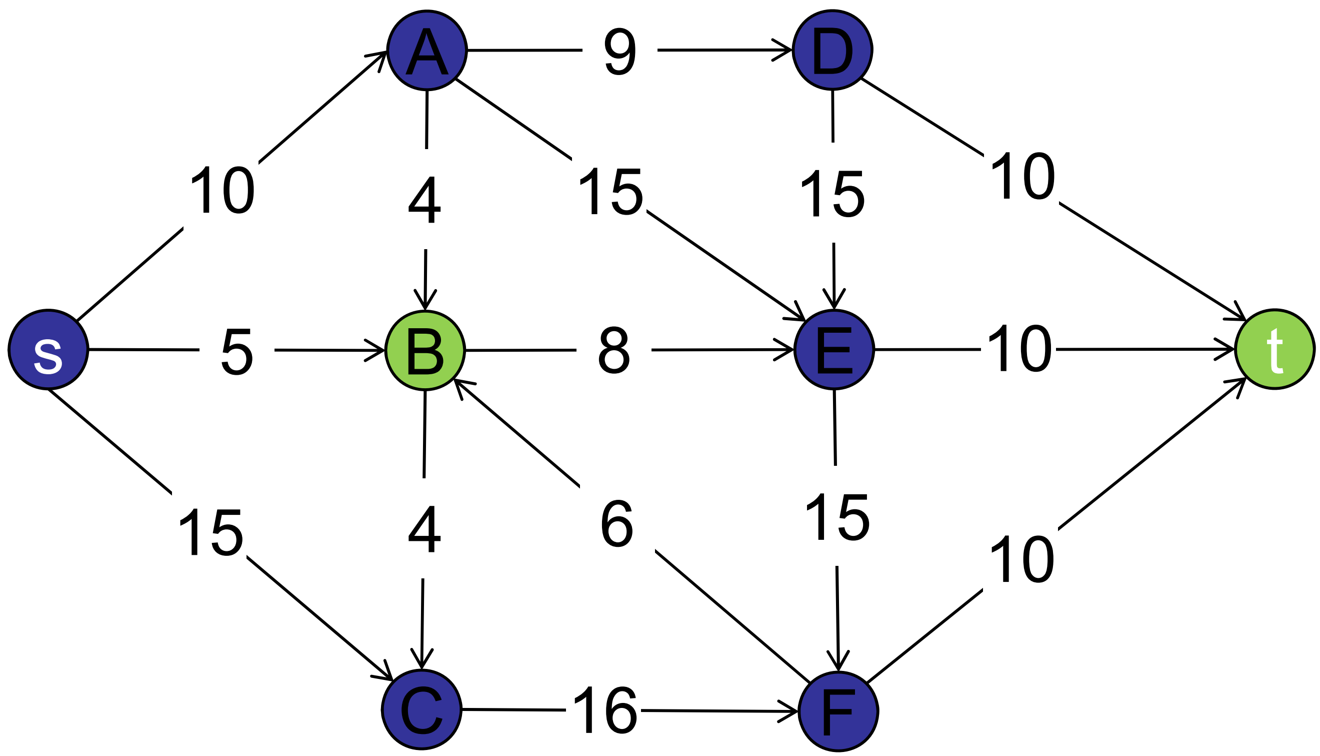


One possible st-cut, set S = blue, set T = green

Cuts and Flows – Definition (2/4)

Definition:

An st-cut partitions the vertices of a graph into two disjoint sets S and T where source s ∈ S and sink t ∈ T.

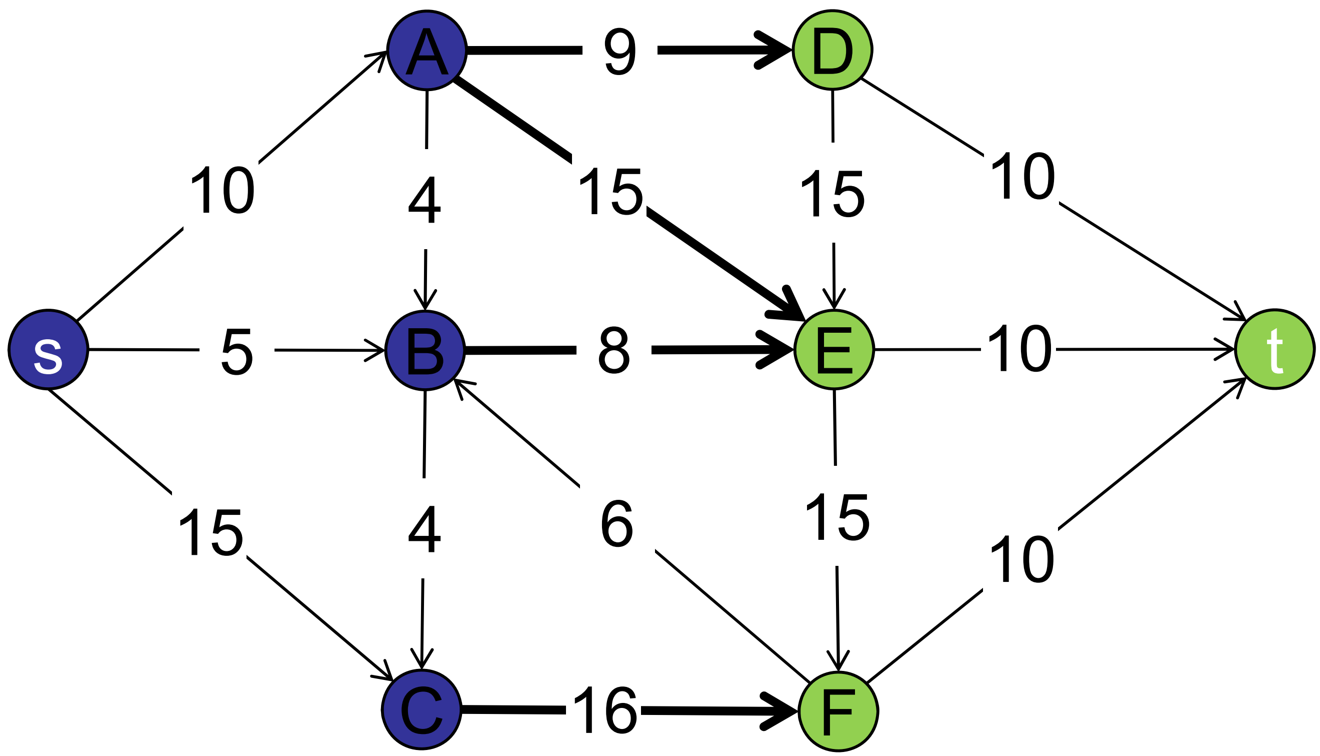


Another possible st-cut, set S = blue, set T = green

Cuts and Flows – Definition (3/4)

Definition:

The capacity of an st-cut is the sum of the capacities of the edges that cross the cut **from S to T**.

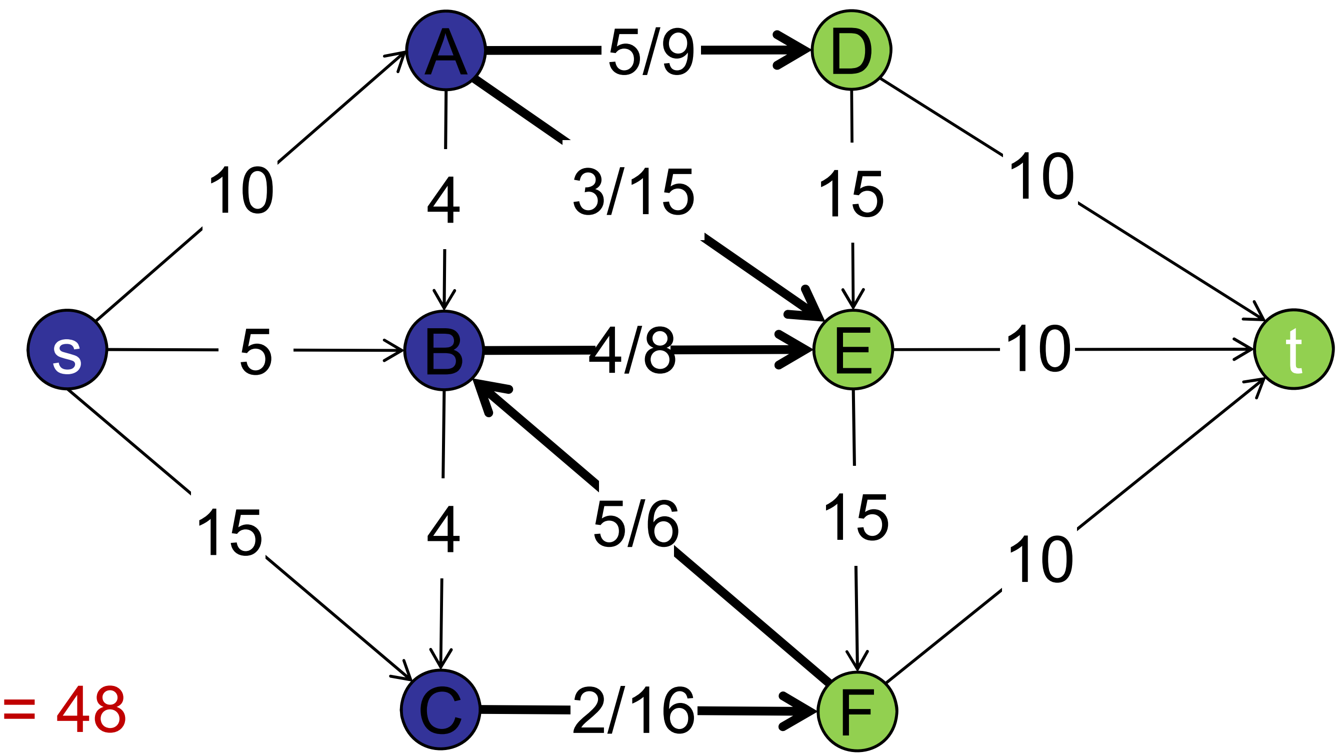


Capacity of this st-cut = 48

Cuts and Flows – Definition (4/4)

Definition:

The net flow across an st-cut is the sum of the flows on edges from S→T **minus the flows from T→S**.



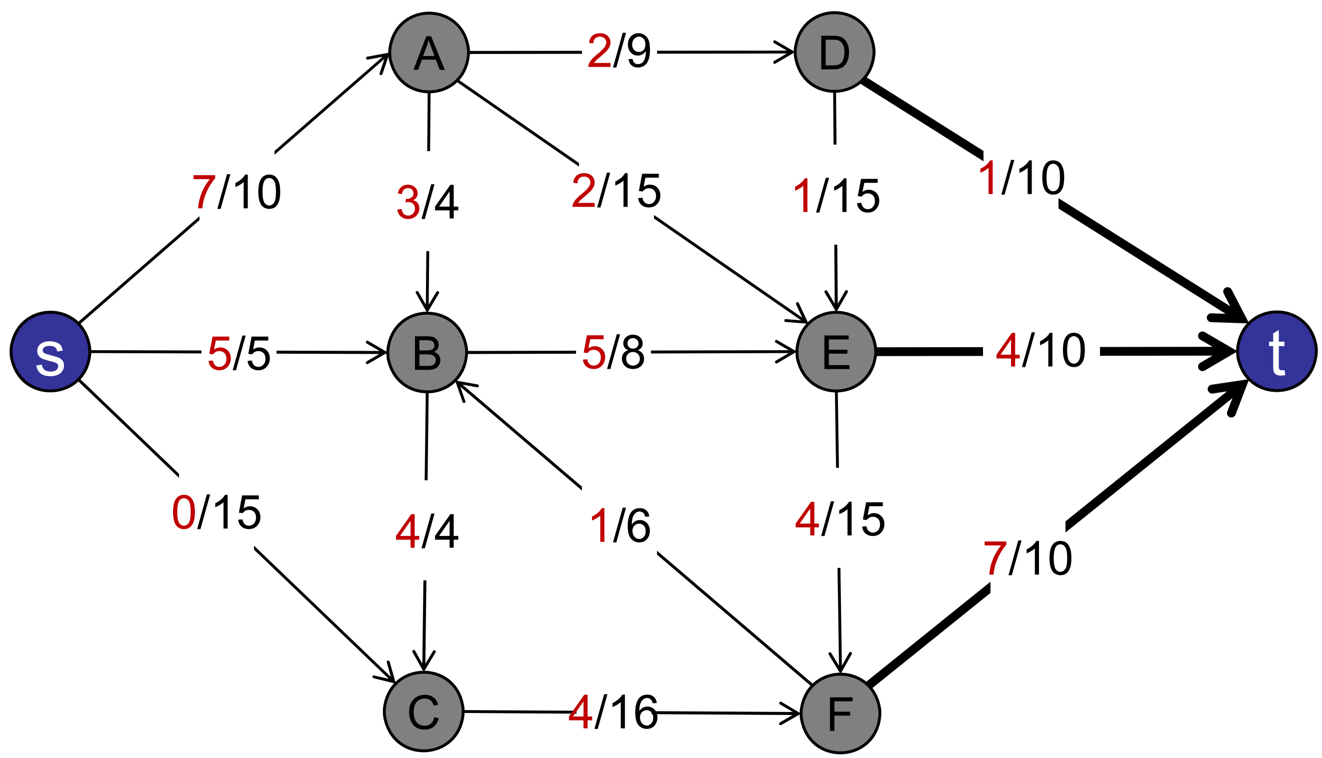
Capacity = 48
Net flow = 9

Cuts and Flows – Equal (1/4)

Proposition:

Let f be a flow, and let (S,T) be an st-cut.

Then the net flow across (S,T) equals the value of f.



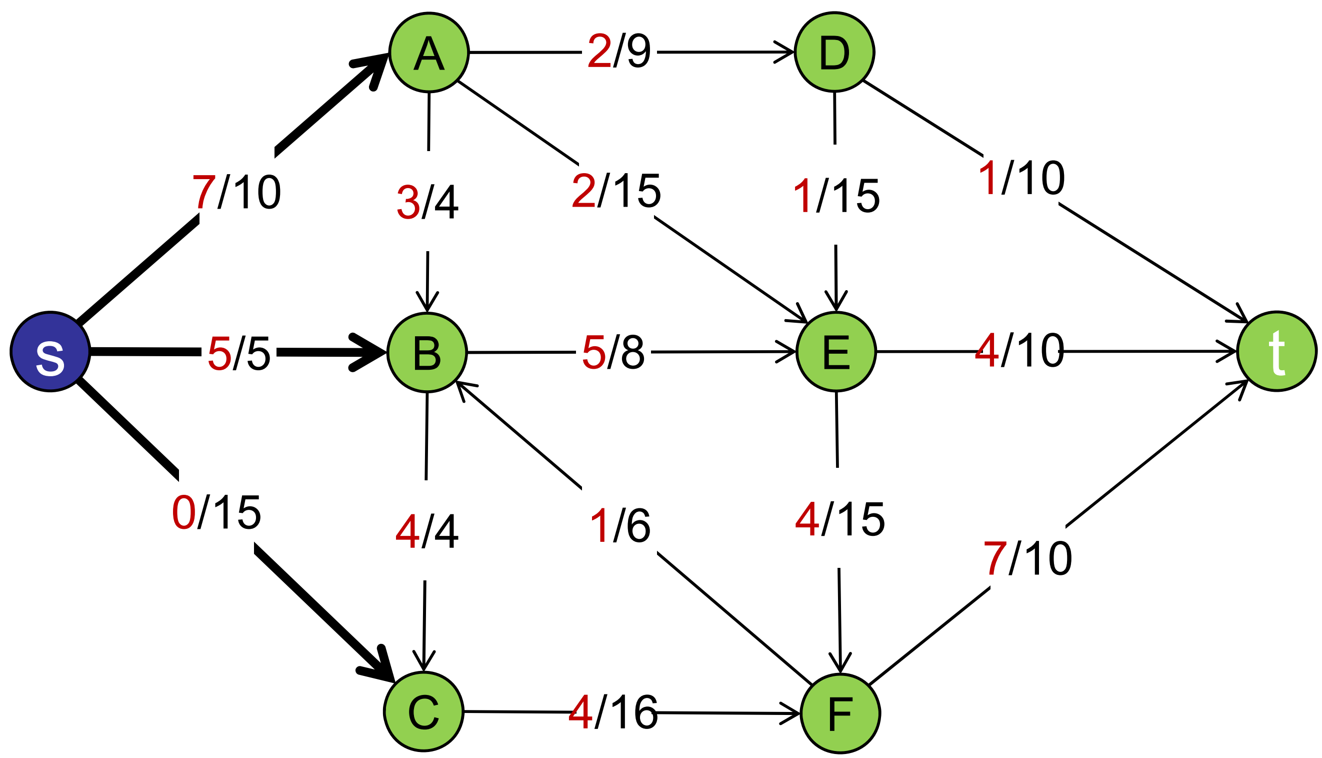
Value of flow = 12

Cuts and Flows – Equal (2/4)

Proposition:

Let f be a flow, and let (S,T) be an st-cut.

Then the net flow across (S,T) equals the value of f.



Flow across cut = 12

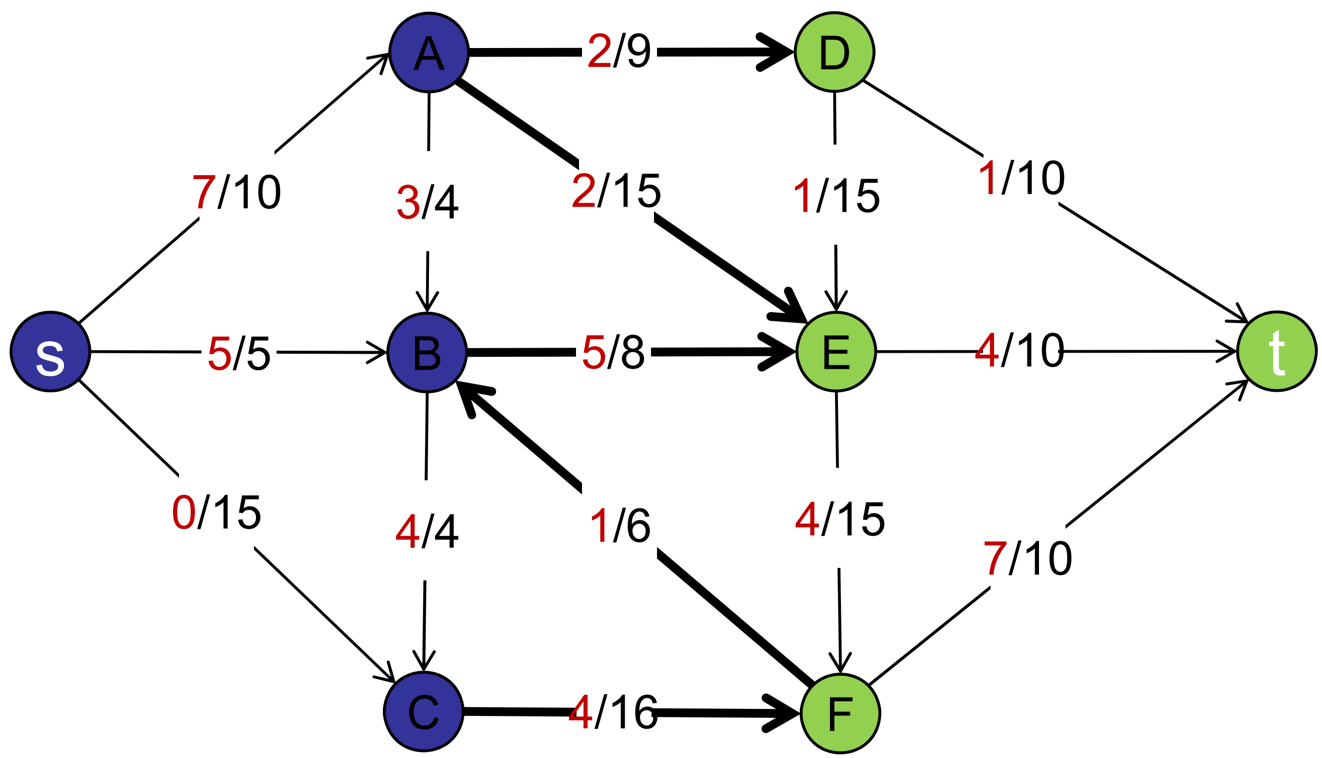
Value of flow = 12

Cuts and Flows – Equal (3/4)

Proposition:

Let f be a flow, and let (S,T) be an st-cut.

Then the net flow across (S,T) equals the value of f.



Flow across cut = 12

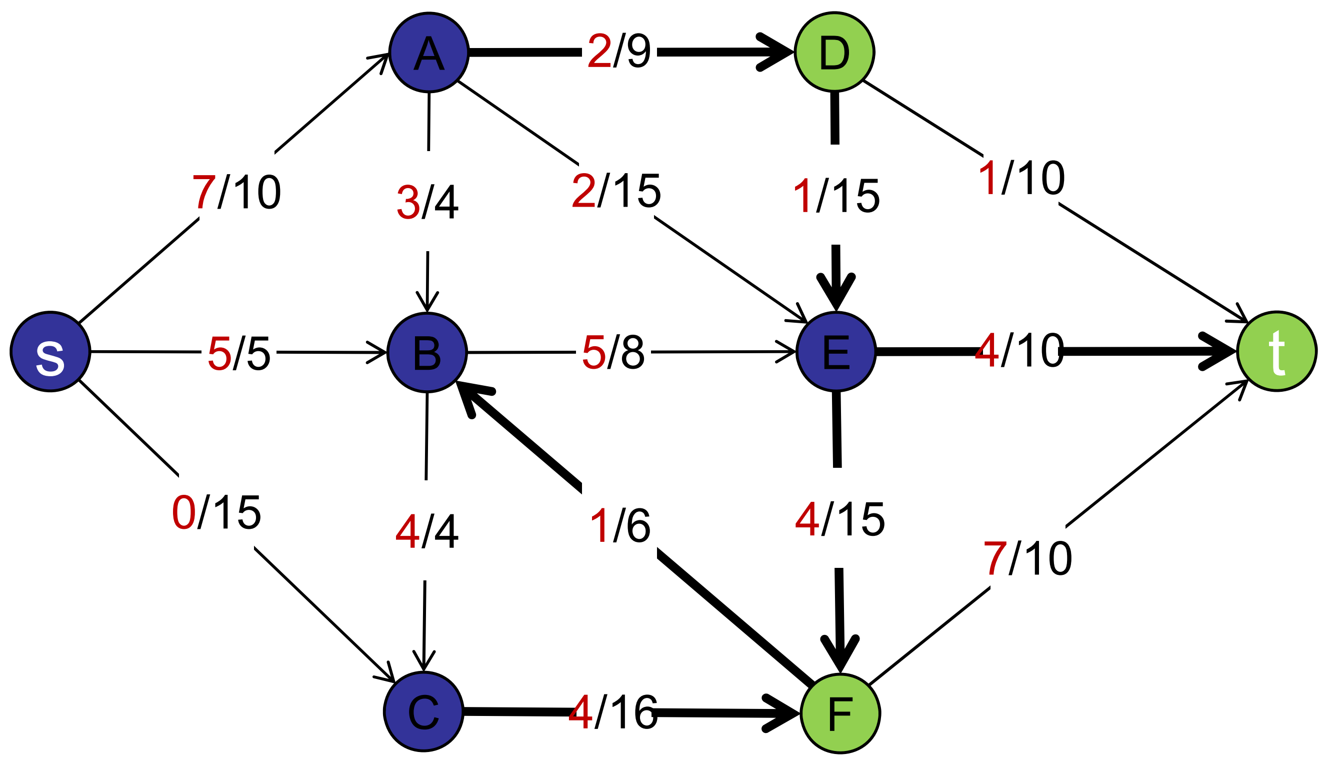
Value of flow = 12

Cuts and Flows – Equal (4/4)

Proposition:

Let f be a flow, and let (S,T) be an st-cut.

Then the net flow across (S,T) equals the value of f.



Flow across cut = 12

Value of flow = 12

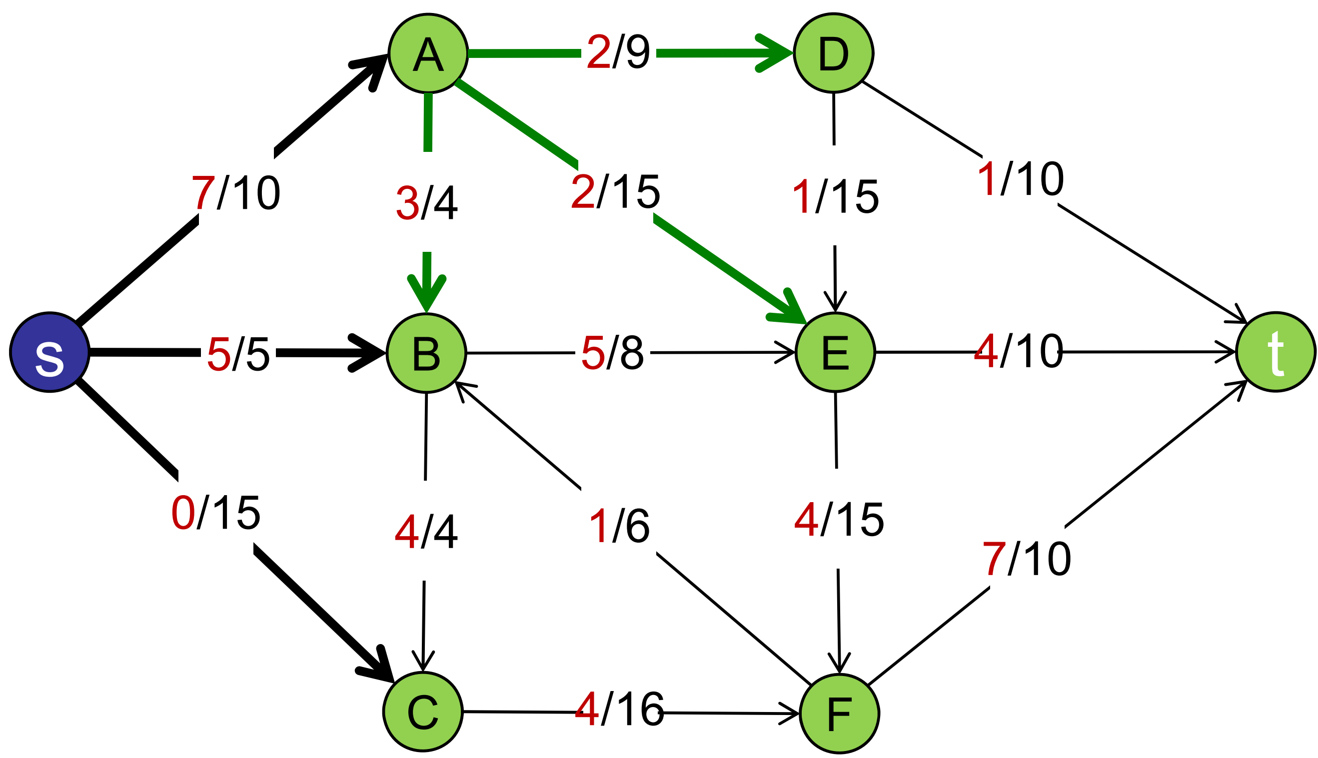
Cuts and Flows – Equal Proof (1/5)

Proof: (by induction)

Start with S = {source s}, T = V \ S.

Define F = flow across cut.

F = 12

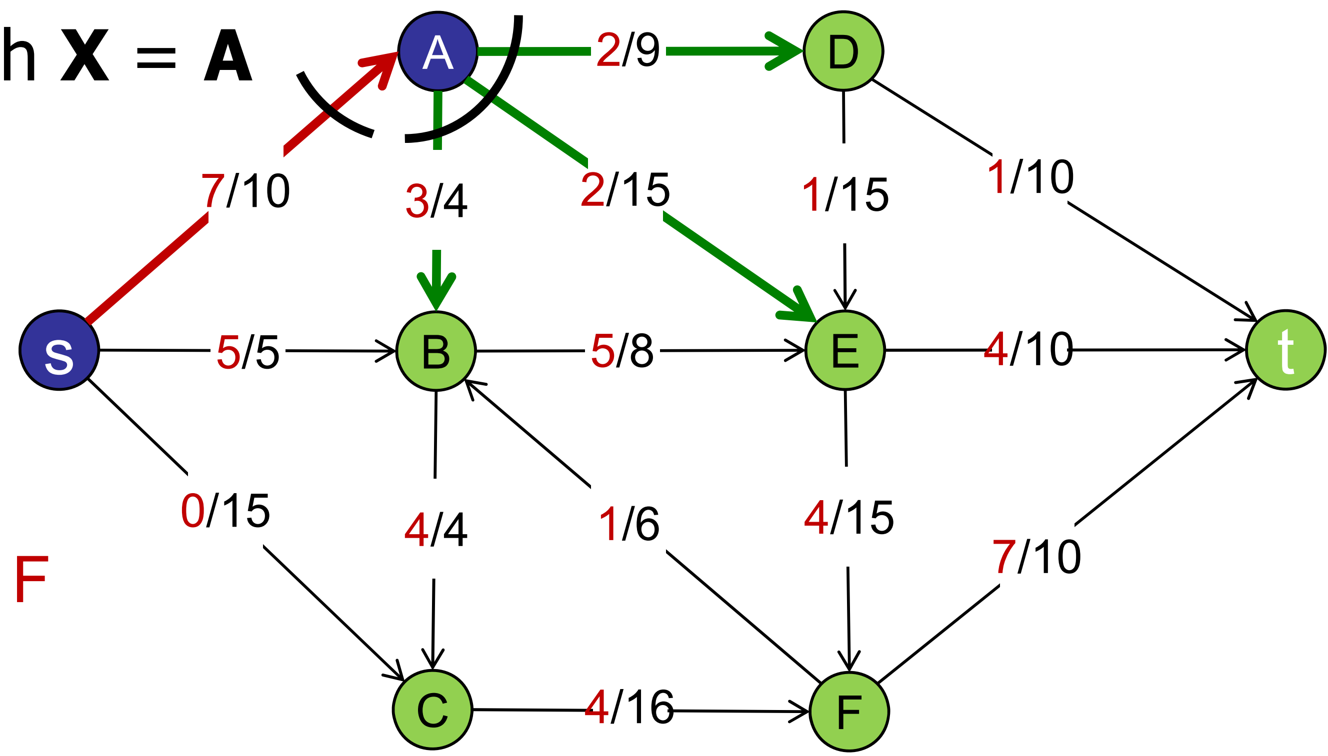


Cuts and Flows – Equal Proof (2/5)

Inductive step:

Take one node X that is reachable from S and add it to S.

- Add new outgoing edges that cross new cut.
- Subtract new incoming edges that cross new cut.
- Subtract/add edges from X to S.
- See example with **X** = **A**



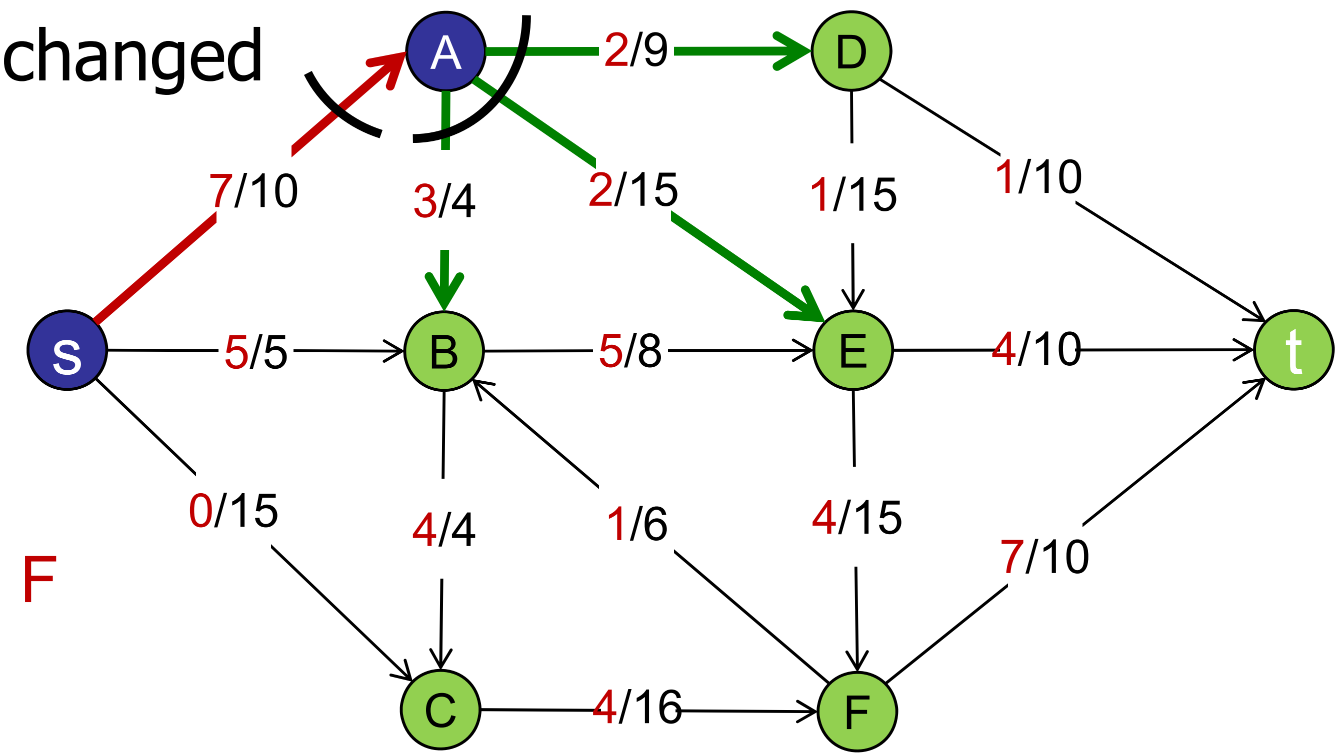
F + (3 + 2 + 2) - 7 - 0 = F

Cuts and Flows – Equal Proof (3/5)

Inductive step:

Conservation of flow: (Equilibrium constraint)

- See example with **X** = **A**
- Flow into **A** equals flow out of **A**
- Flow that crossed (old set **S**)→**A** == **A**→(old set **T**)
- So F remains unchanged

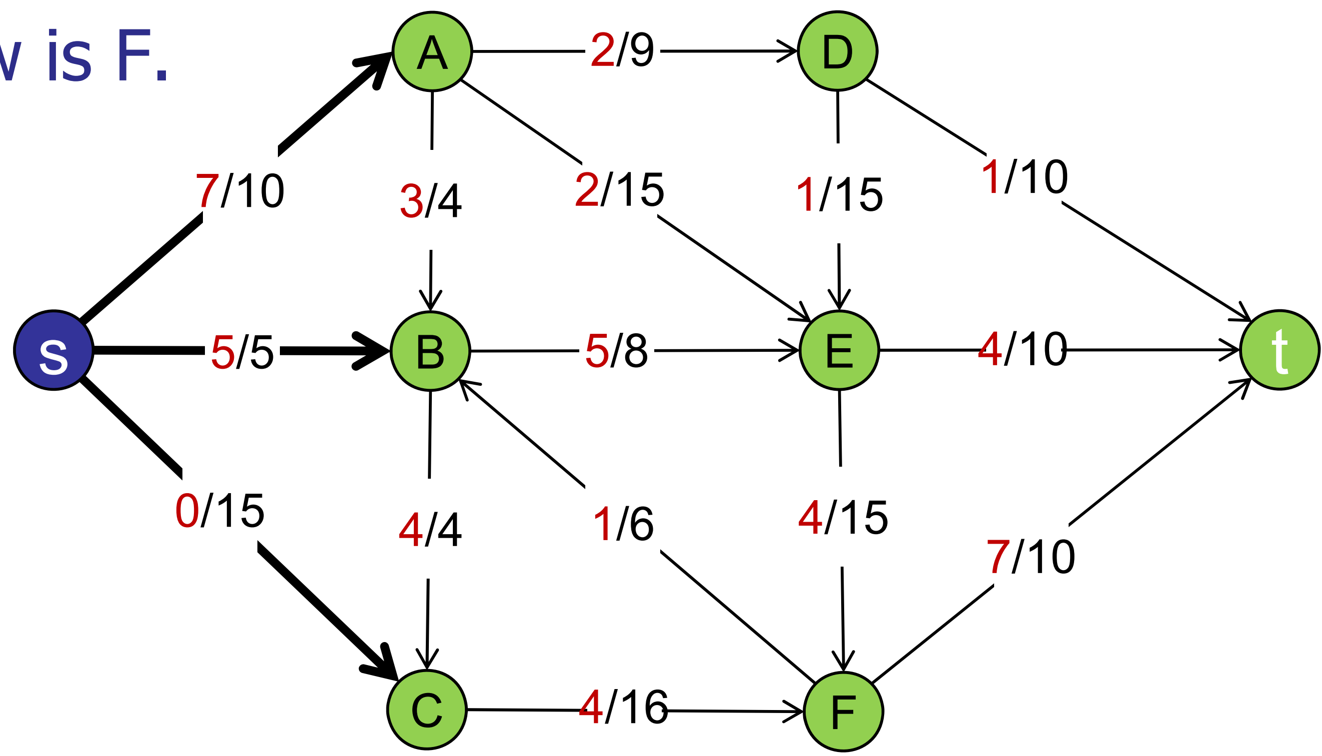


F + (3 + 2 + 2) - 7 - 0 = F

Cuts and Flows – Equal Proof (4/5)

Proof: (by induction)

- Start with S = {source s}, T = V \ S.
- Define F = flow across cut.
- Move vertices one at a time from T to S.
- At every step, F remains unchanged.
- Thus for <u>all cuts</u>, flow is F.

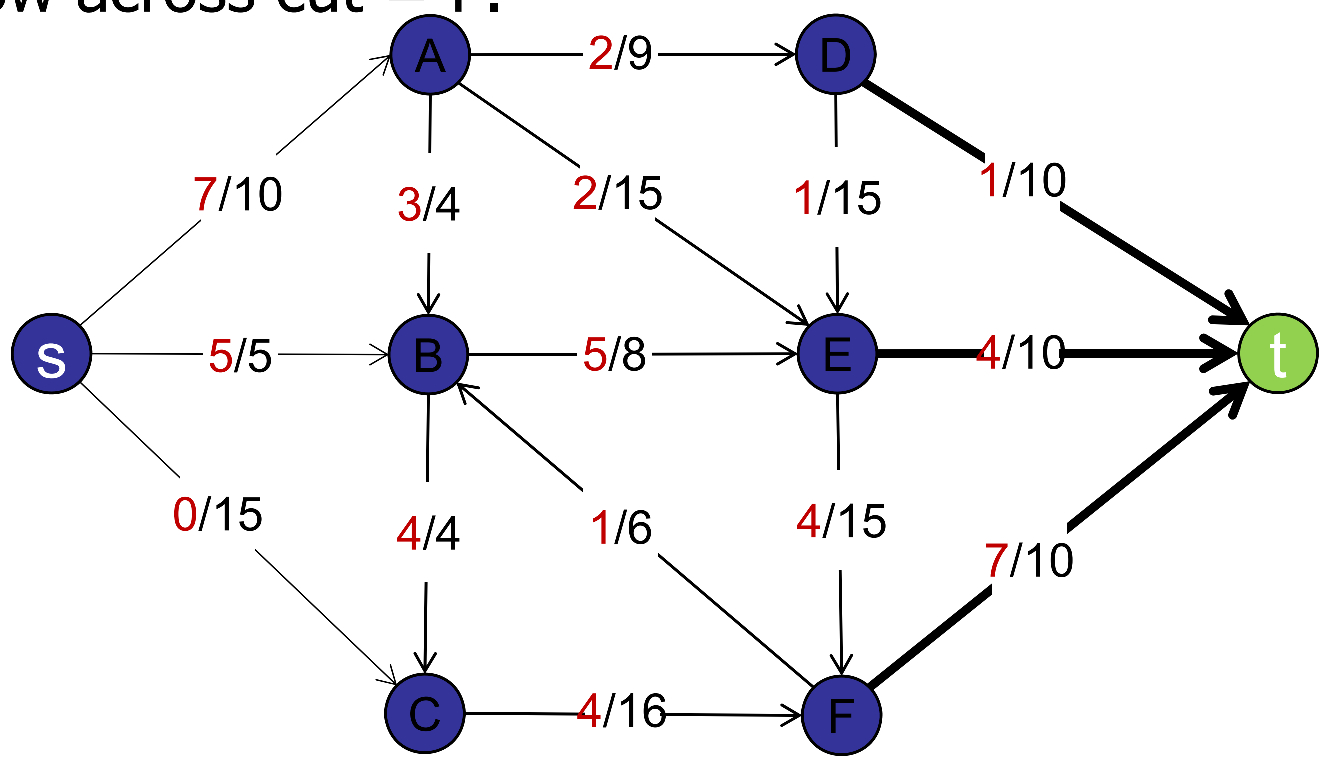


Cuts and Flows – Equal Proof (5/5)

Proof: (by induction)

How to easily compute F?

- Consider cut S = V \ {sink t}, T = {sink t}.
  - One other easy way exist, S = {source s}, T = V \ {source s}.
- All edges crossing the cut go to sink t.
- Value of flow = flow across cut = F.

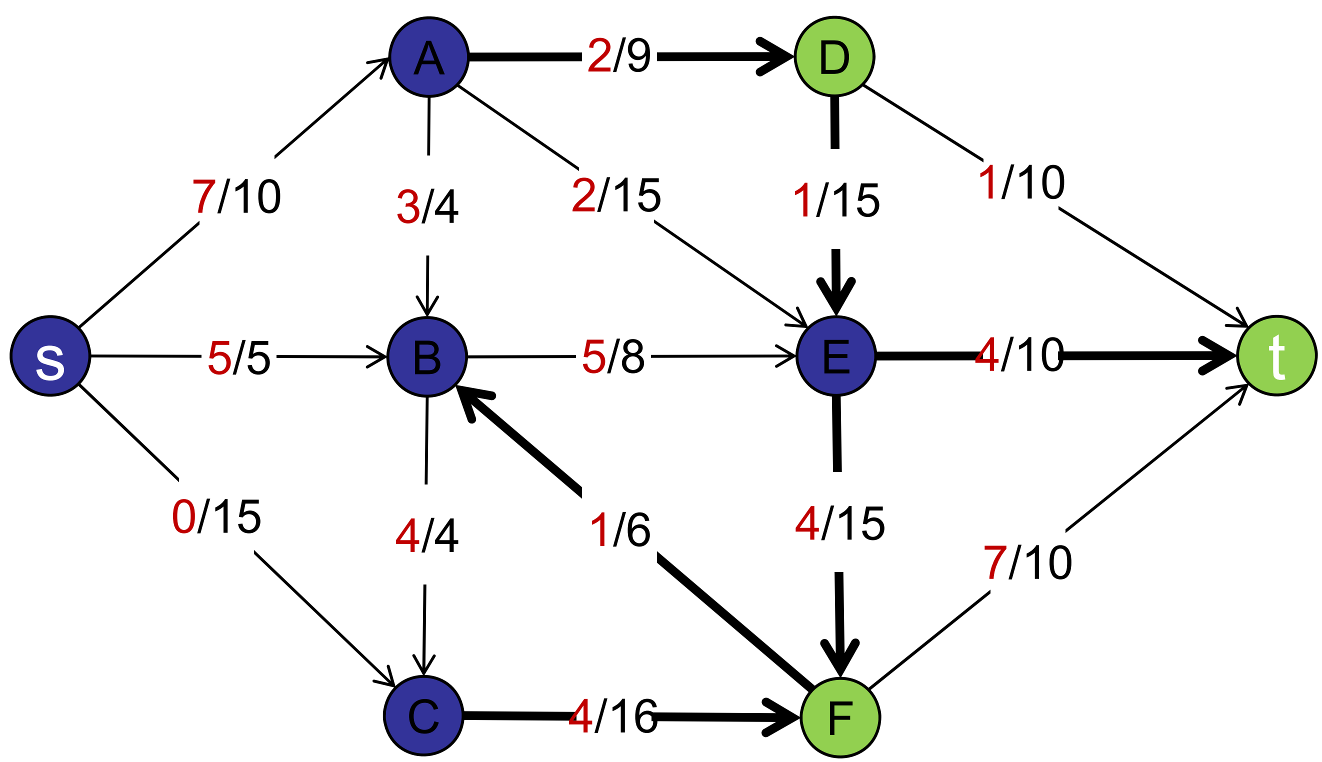


Cuts and Flows – Weak Duality (1/2)

Weak duality:

Let f be a flow, and let (S,T) be an st-cut.

Then value(f) ≤ capacity(S,T).



Cuts and Flows – Weak Duality (2/2)

Weak duality:

Let f be a flow, and let (S,T) be an st-cut.

Then value(f) $\leq$ capacity(S,T).

Proof:

value(f) = flow across cut (S,T) $\leq$ capacity(S,T).

flow value proposition

flow is bounded by the capacity

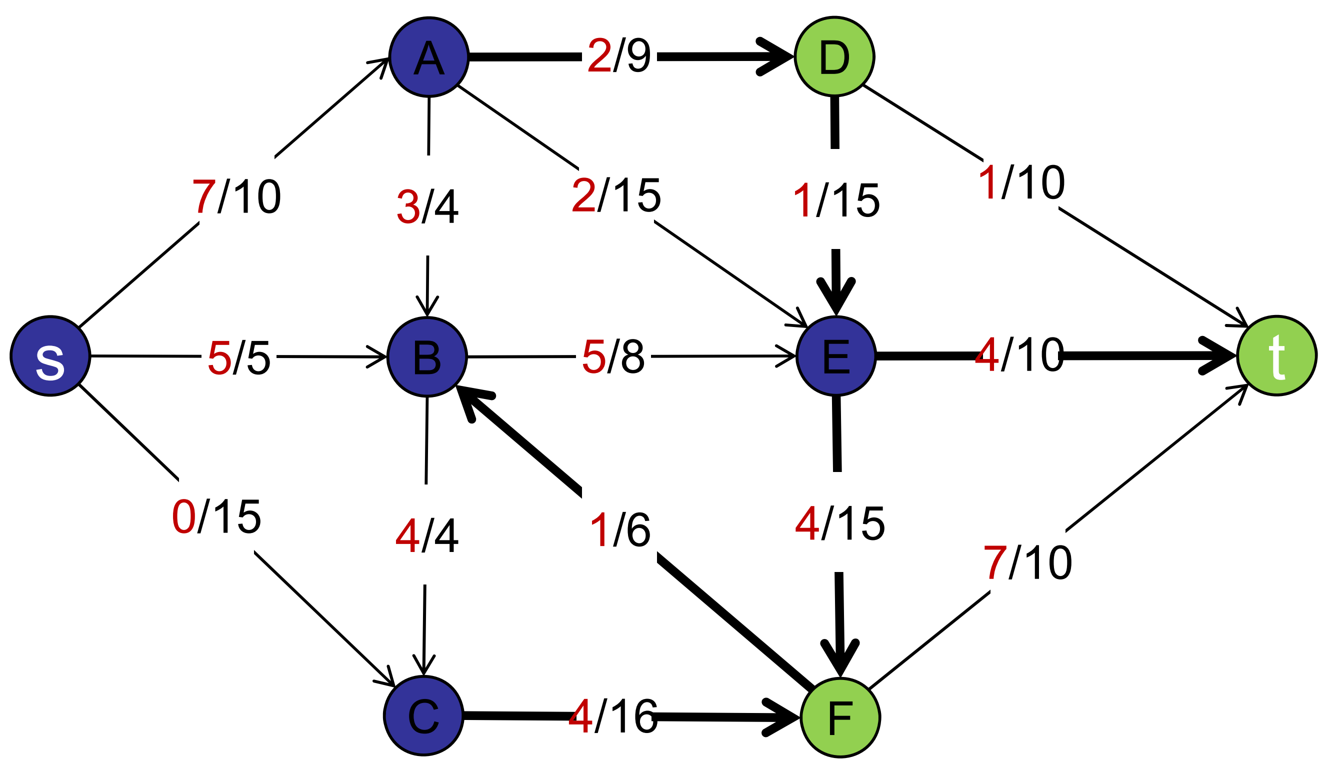
MaxFlow-MinCut Theorem

MaxFlow-MinCut Theorem:

Let f be a maximum flow.

Let (S,T) be an st-cut with minimum capacity.

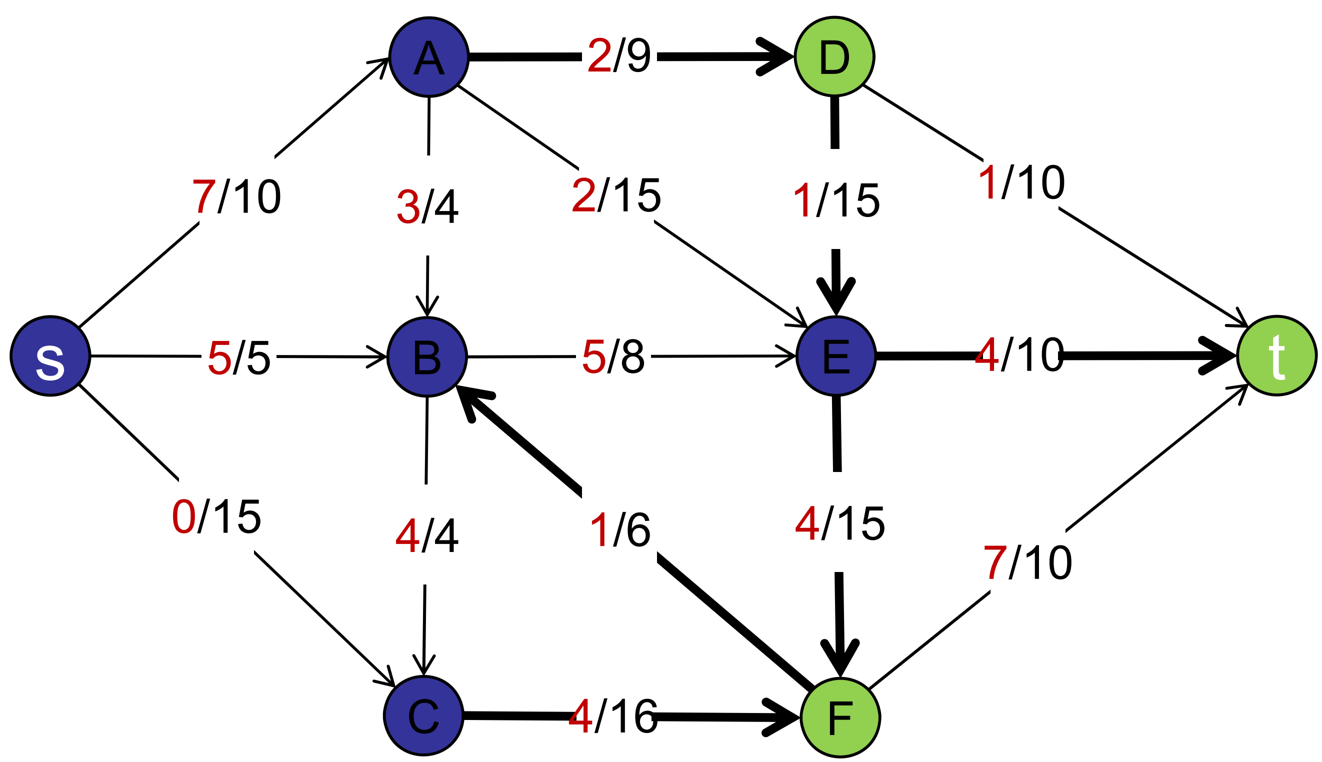
Then value(f) = capacity(S,T).



Augmenting Path Theorem

Augmenting Path Theorem:

Flow f is a maximum flow if and only if there are no augmenting paths in the residual graph.



Cuts and Flows – 3 Statements

Proof:

The following three statements are equivalent for flow f:

1. There exists a cut whose capacity equals the value of f.
2. f is a maximum flow.
3. There is no augmenting path with respect to f.

Cuts and Flows – Statement 1→2

1 → 2: There exists an $\boxed{\text{f-capacity cut}}$ ➔ f is maximum

Assume (S,T) is an st-cut with <u>minimum capacity</u> equal to f.

- We will change this "assumption" into a constructive proof in Step 3 → 1

weak duality

- For all flows g: value(g) ≤ capacity(S,T)
- For all flows g: value(g) ≤ value(f)
- f is a maximum flow

Cuts and Flows – Statement 2→3

2 → 3: f is maximum flow ➔ no augmenting paths

Assume there IS at least 1 more augmenting path:

- Improve flow by sending flow on augmenting path.
- Augmenting path has bottleneck capacity > 0.
- f was NOT a maximum flow.
- Contradiction

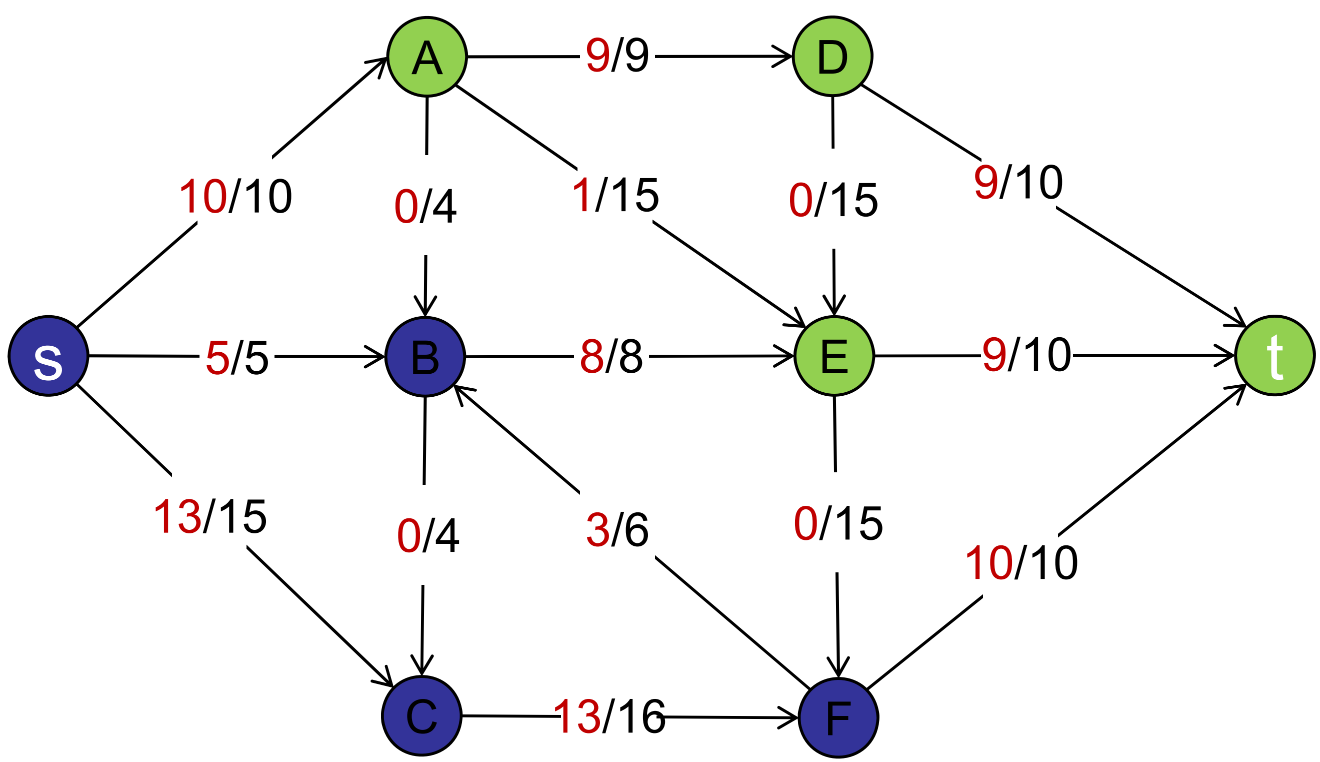
Conclusion: after we find f (max flow),
there is no more augmenting path

Cuts and Flows – Statement 3→1 (1/4)

3 → 1: no augmenting paths ➔ exists f-capacity cut

Assume there is no augmenting path:

- Let S be the vertices reachable from the source in the residual graph.
- Let T be the remaining vertices, i.e. T = V \ S.



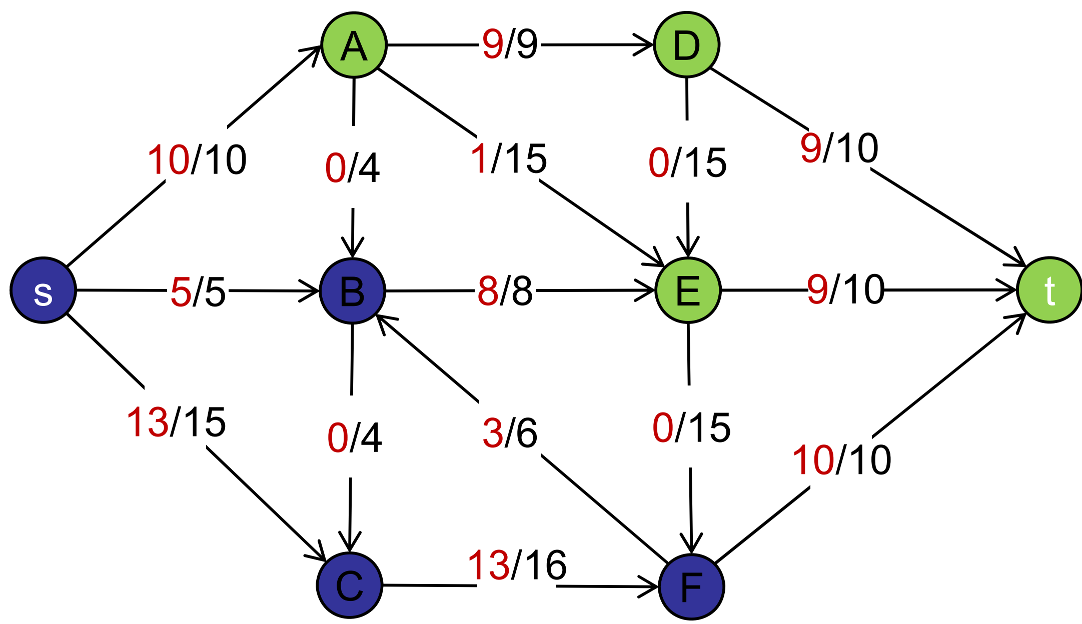
Cuts and Flows – Statement 3→1 (2/4)

3 → 1: no augmenting paths ➔ exists f-capacity cut

Assume there is no augmenting path:

- Let S = reachable vertices. T = remaining vertices.
- S contains the source s and T contains the target/sink t.
- source s cannot reach sink t anymore

Otherwise, if sink t was reachable from source s in the residual graph, there would be another augmenting path.

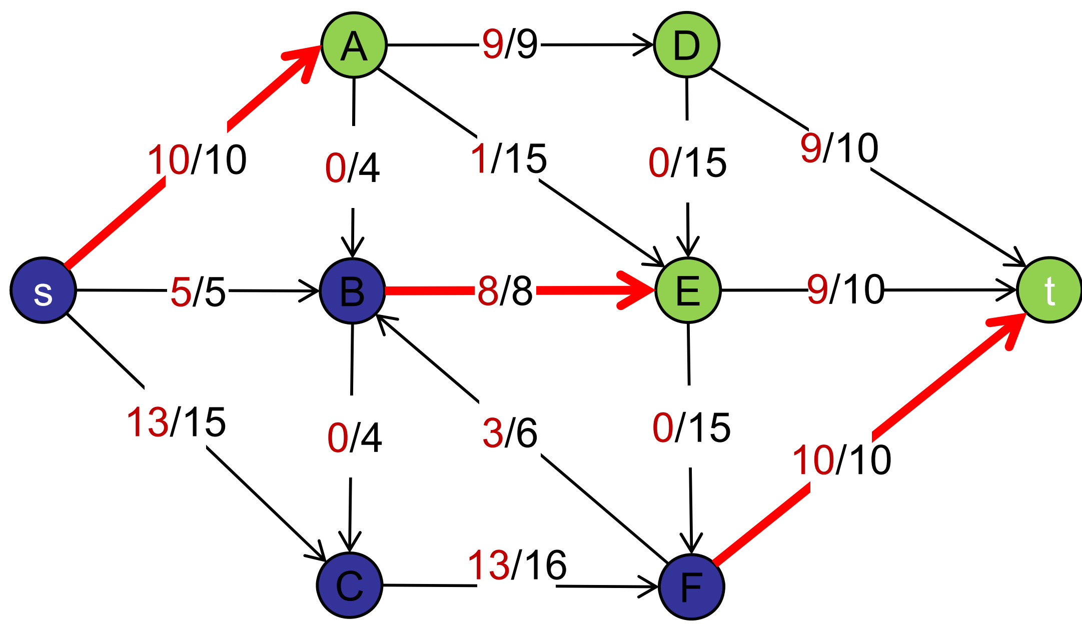


Cuts and Flows – Statement 3→1 (3/4)

3 → 1: no augmenting paths ➔ exists f-capacity cut

Assume there is no augmenting path:

- Let S = reachable vertices. T = remaining vertices.
- (S,T) is an st-cut.
- All edges from T→S are **empty**.
- All edges from S→T are **saturated**.



Cuts and Flows – Statement 3→1 (4/4)

3 → 1: no augmenting paths ➔ exists f-capacity cut

Assume there is no augmenting path:

- Let S = reachable vertices. T = remaining vertices.
- (S,T) is an st-cut.
- All edges from T→S are **empty**.
- All edges from S→T are **saturated**.
- value(f) = net flow across (S,T) = capacity of cut

flow value proposition

Notice ≤ changed to =
S→T saturation
T→S empty

Cuts and Flows – Statement 1→2 (again)

1 → 2: There exists an $\boxed{\text{f-capacity cut}}$ ➔ f is maximum

We can now constructively show how to get (S,T), an st-cut with minimum capacity equal to f.

- For all flows g: value(g) ≤ capacity(S,T)
- For all flows g: value(g) ≤ value(f)
- f is a maximum flow

Summary

Augmenting Path Theorem:

Flow f is a maximum flow if and only if there are no augmenting paths in the residual graph.

➔ If Ford-Fulkerson terminates, then there is no augmenting path (left).

➔ Thus, the resulting flow is maximum.

Ford-Fulkerson (Yet More Details 1/3)

Ford-Fulkerson Algorithm
Start with 0 flow. While there exists an augmenting path: • Find an augmenting path. • Compute bottleneck capacity. • Increase flow on the path by the bottleneck capacity.

We have just seen that if FF terminates, it has found max flow

- ✓ How to find an augmenting path?
- ✓ If it terminates, does it always find a max-flow?
- Does Ford-Fulkerson always terminate? How fast?

Ford-Fulkerson (Yet More Details 2/3)

Ford-Fulkerson Algorithm
Start with 0 flow. While there exists an augmenting path: • Find an augmenting path. • Compute bottleneck capacity. • Increase flow on the path by the bottleneck capacity.

Termination: FF always terminates <u>if the capacities are integers</u>.

- Every iteration finds a new augmenting path.
- Each augmenting path has bottleneck capacity at least 1.
- So each iteration increases the flow of at least one edge by at least 1.
- Finite number of edges, finite max capacity per edge → termination.

Ford-Fulkerson (Yet More Details 3/3)

Ford-Fulkerson Algorithm
Start with 0 flow. While there exists an augmenting path: • Find an augmenting path. • Compute bottleneck capacity. • Increase flow on the path by the bottleneck capacity.

Termination: Ford-Fulkerson always terminates.

- ✓ How to find an augmenting path?
- ✓ If it terminates, does it always find a max-flow?
- How fast does Ford-Fulkerson terminate? Can we do better?
  - **After the break…**

Roadmap (Flipped Classroom)

Network Flows

a. Definition (with VA)
b. Ford-Fulkerson Algorithm (with VA)
c. Max-Flow/Min-Cut Theorem
d. **Ford-Fulkerson (FF) Analysis**
    a. **Analysis of Basic Ford-Fulkerson: $O(m^2 U)$**
    b. FF with Shortest-Path v1/**Edmonds-Karp**: $O(m^2 n)$
    c. FF with Shortest-Path v2/**Dinic's**: $O(m n^2)$
e. Live solve a (simple) Max Flow problem

Ford-Fulkerson Analysis

The basic algorithm runtime:

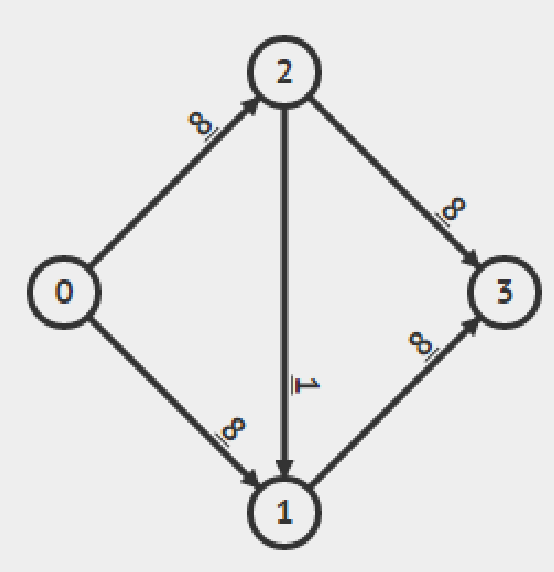
- Each iteration: O(**m**) for running DFS to find **p** in **R**
  - And O(**n**) to update capacities in **R** along **p,** but **m > n** in **R**
- So, the main question is: How many iterations will FF run?
  - Important assumption for the next line: **Capacities are integers**
  - Bottleneck edge = min capacity edge on **P**, it has min capacity of 1
  - So each iteration will increase flow value by ≥ 1 unit
  - Let **U** = max capacity of outgoing edge connected to source **s**
  - Max flow **MF ≤ m*U**, assuming that all **m** edges have capacity **U**
  - # iterations **≤ m*U**, as each iteration increase flow by ≥ 1
- → Total cost: O(**m*U * m**) = O(**m² U**)

Yes, this is a gross upperbound, see T04

Ford-Fulkerson Worst Case Input

Is it really so bad?

YES



This example is available at VisuAlgo maxflow visualization, try Example Graphs: Ford-Fulkerson Killer and run Ford-Fulkerson

Example of a simple Flow Network that causes bad performance of simple FF
To exaggerate the effect, assume 8 = 8B(illion) unit capacity
Assume FF takes augmenting paths 0→2→1→3 and then 0→1→2→3 alternatingly
It will only stop until ~16B steps

What went wrong?

Why did FF choose such a bad augmenting path?

PS: In practice, it is not like this, see PS3+4

Roadmap (Flipped Classroom)

Network Flows

a. Definition (with VA)

b. Ford-Fulkerson Algorithm (with VA)

c. Max-Flow/Min-Cut Theorem

d. Ford-Fulkerson (FF) Analysis

  a. Analysis of Basic **Ford-Fulkerson**: $O(m^2 U)$

  b. FF with Shortest-Path v1/Edmonds-Karp: $O(m^2 n)$

  c. FF with Shortest-Path v2/**Dinic's**: $O(m n^2)$

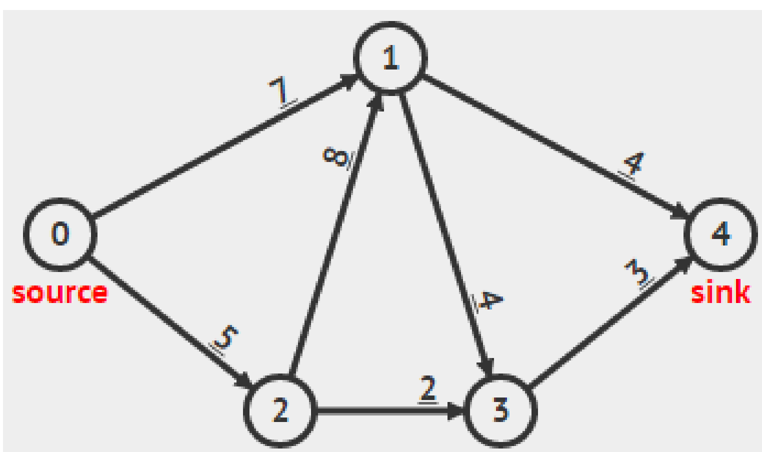
e. Live solve a (simple) Max Flow problem

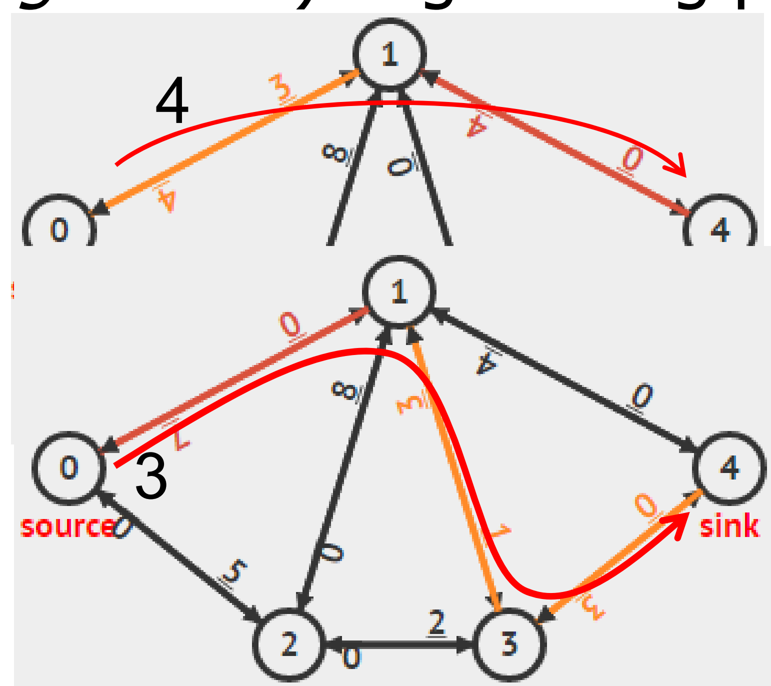
Edmonds-Karp (EK) Algorithm

Idea: What if we don't consider *any* augmenting paths but consider augmenting paths with the *smallest number of edges involved first* (so we don't put flow on more edges than necessary)

Implementation: We first ignore capacity of the edges first (assume all edges in **R** have weight 1), and we run O(**E**) BFS to find the shortest (*in terms of # of edges used*) augmenting path

[VA link]()





EK – Claims

1. Distance from source vertex **s** to any other vertices (including to sink vertex **t**) never decreases
   - Augmenting path(s) push **s** and **t** further apart in **R**
2. EK will use at most **m*n** iterations

If we can show this, it means that

➔EK runs at most in O(**mn * m**) = O(**m² n**) time

- Yey, our max flow algorithm is no longer dependant on **U** (or **F**) → this is called: *strongly polynomial* algorithm
- Max-Flow is **NOT** an NP-hard optimization problem

This AY, the proof is skipped so that we can do live-demo (but they are left in the slides)

- Instead, we see the works done by your seniors

Proof of Claim no 1 (1/2)

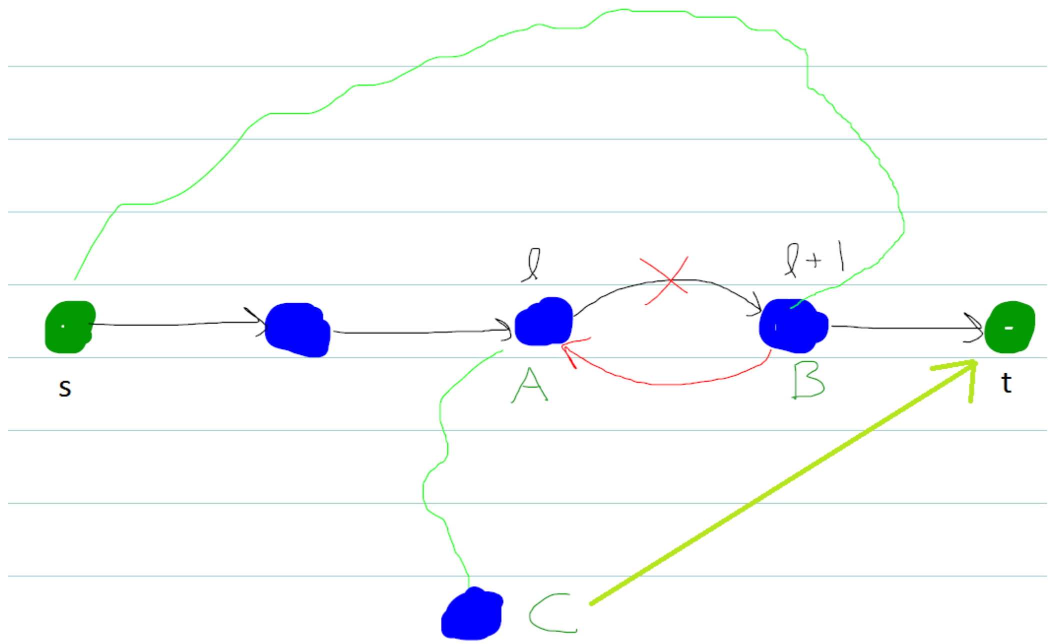
Distance from source vertex **s** (to **t**) never decreases

On augment, we have two possible outcomes:

1. We "delete" at least 1 (or more) bottleneck edge(s) in **R**
   - No problem, this outcome cannot shorten **s**→**t** path
2. We may add backward/reverse edges in **R**
   - Those from initial capacity 0 to +**f** upon pushing flow **f** on the opposite forward edges along augmenting path
   - Notice that such addition of backward edges can only happen along the shortest **s**→**t** augmenting path in **R** that we are processing
   - Will it cause problem?

Proof of Claim no 1 (2/2)

2. We may add backward/ reverse edges in **R**



- Will it cause problem?
- Every step on shortest path **s→t** increases distance
- Thus, a shorter path **s→t**, if exist, must cross the newly created edge; Suppose we have new backward edge **B→A**
- We see that **d(s, B)** cannot get shorter
- So **d(s, A)** over new edge = **d(s, B) + 1 ≥ (l+1) + 1 = l+2 > l**
  - Shortest path **s→A** cannot cross new edge, i.e., **d(s, A)** doesn't decrease
- So **d(s, C)** cannot cross edge **B→A** as it won't make it shorter
- A→C~→t could not be shorter than A→B~→t previously
- So s~→B→A→C~→t cannot shorten **s→t** path

Proof of Claim no 2 (1/3)

After finding an augmenting path **p**, every bottleneck edge (**A**, **B**) along path **p** will be "deleted" from R

There is ≥ 1 bottleneck edge at each augmenting path

We will show that each of the **m** edges in **R** can become bottleneck edge at most **n/2** times

Let **A** and **B** be two vertices that are connected by an edge in **R** and since any augmenting path **p** in EK is shortest path, when (**A**, **B**) becomes bottleneck edge for the first time, we have **dist(s, B) = dist(s, A) + 1**

After augmentation of **p**, edge (**A**, **B**) is deleted from **R**

Proof of Claim no 2 (2/3)

Can edge (**A**, **B**) reappear in **R** again?

Yes, it can… if the flow from **A** to **B** is <u>decreased</u>, which occurs only if the reverse/backward edge (**B**, **A**) appears on some other shortest augmenting path in latter iteration

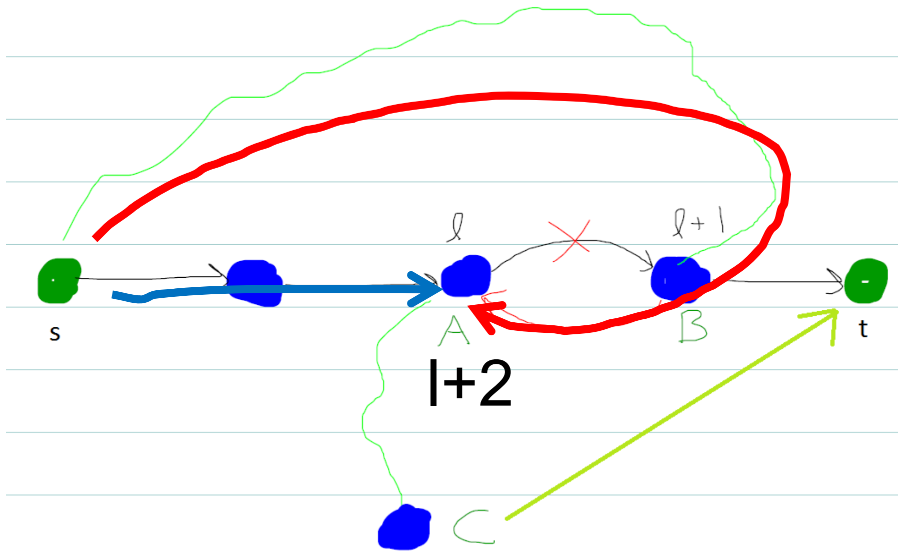
When it happens, we have **dist(s, A) = dist(s, B) + 1**

As each augmentation can never decrease shortest path from **s** (from earlier proof)

Proof of Claim no 2 (2/3)

dist(s, A) = dist(s, B)+1

$\geq$ old-dist(s, B)+1 = old-dist(s, A)+2



So, from the time edge (**A**, **B**) is the bottleneck edge, deleted due to an augmenting path, and reappears later, dist(**s**, **A**) <u>must have increased by at least 2</u>

As shortest path from **s** to **t** in **R** is at most **n**, this arbitrary edge (**A**, **B**) can only be bottleneck **n/2** times

EK – Analysis

Conclusions:

➔An arbitrary edge (**A**, **B**) can be a bottleneck edge up to **n/2** times

➔As there are at most **2m** edges in **R**, there can be at most O(**2m*n/2**) = O(**mn**) bottleneck edges

➔So EK will run at most O(**mn**) iterations

➔Total time of EK is O(**mn * m**) = O($\mathbf{m^2}$ **n**)

- ➔A polynomial time algorithm

Edmonds-Karp "Worst Case" Input

How to enforce EK to run up to O(**mn**) iterations?

In the first 5 AYs, 5 batches of your seniors tried hard to do this

PS: Possible flow graph structure hidden in the event
I still want to reuse this exercise

Record holders:

- Bui Do Hiep (AY 2016/17, 5 years ago) test case,
  - n = 102, m = 1975, nm = 201,450, AP = 16,250 (~1/12 of nm, or 8.06%)
- Gan Wei Liang (AY 2017/18, 4 years ago), ~8.3%
- Sidhant Bansal (AY 2019/20, 2 years ago), ~8.91%
- Teo Wei Zheng (AY 2020/21, last AY) = 8.967%... (~1/11 of nm)

Roadmap (Flipped Classroom)

Network Flows

a. Definition (with VA)
b. Ford-Fulkerson Algorithm (with VA)
c. Max-Flow/Min-Cut Theorem
d. **Ford-Fulkerson (FF) Analysis**
    a. Analysis of Basic **Ford-Fulkerson**: $O(m^2 U)$
    b. FF with Shortest-Path v1/**Edmonds-Karp**: $O(m^2 n)$
    c. **FF with Shortest-Path v2/Dinic's: $O(m n^2)$**
e. Live solve a (simple) Max Flow problem

Dinic's (Preview)

Dinic's algorithm is "90%" identical as Edmonds-Karp

Just that Dinic's uses BFS (shortest path) information in a "better way"

For now, a quick explanation using
https://visualgo.net/en/maxflow
with a follow-up later in T04

Roadmap (Flipped Classroom)

Network Flows

a. Definition (with VA)
b. Ford-Fulkerson Algorithm (with VA)
c. Max-Flow/Min-Cut Theorem
d. Ford-Fulkerson (FF) Analysis
    a. Analysis of Basic **Ford-Fulkerson**: O(m^2 U)
    b. FF with Shortest-Path v1/**Edmonds-Karp**: O(m^2 n)
    c. FF with Shortest-Path v2/**Dinic's**: O(m n^2)
e. **Live solve a (simple) Max Flow problem**

Live Solve

Let's cap off this lecture with a live demonstration on how to solve a (simple) max flow problem:

Steps:

1. Realizes that the given problem is really a max flow problem
2. Copy paste something...
3. Construct the required flow graph
4. Run the efficient-enough max flow algorithm (Dinic's)
5. Done