CS5234
Combinatorial and Graph Algorithms

Network Flows and Cuts

Types of Graph Problems

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

- Single-source shortest paths
- All-pairs shortest paths

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

- Minimum/maximum spanning tree
- Steiner tree
- Travelling salesman

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

Roadmap

Network Flows

a. Network flows defined
b. Ford-Fulkerson algorithm
c. Max-Flow / Min-Cut Theorem

Network Flow Problems

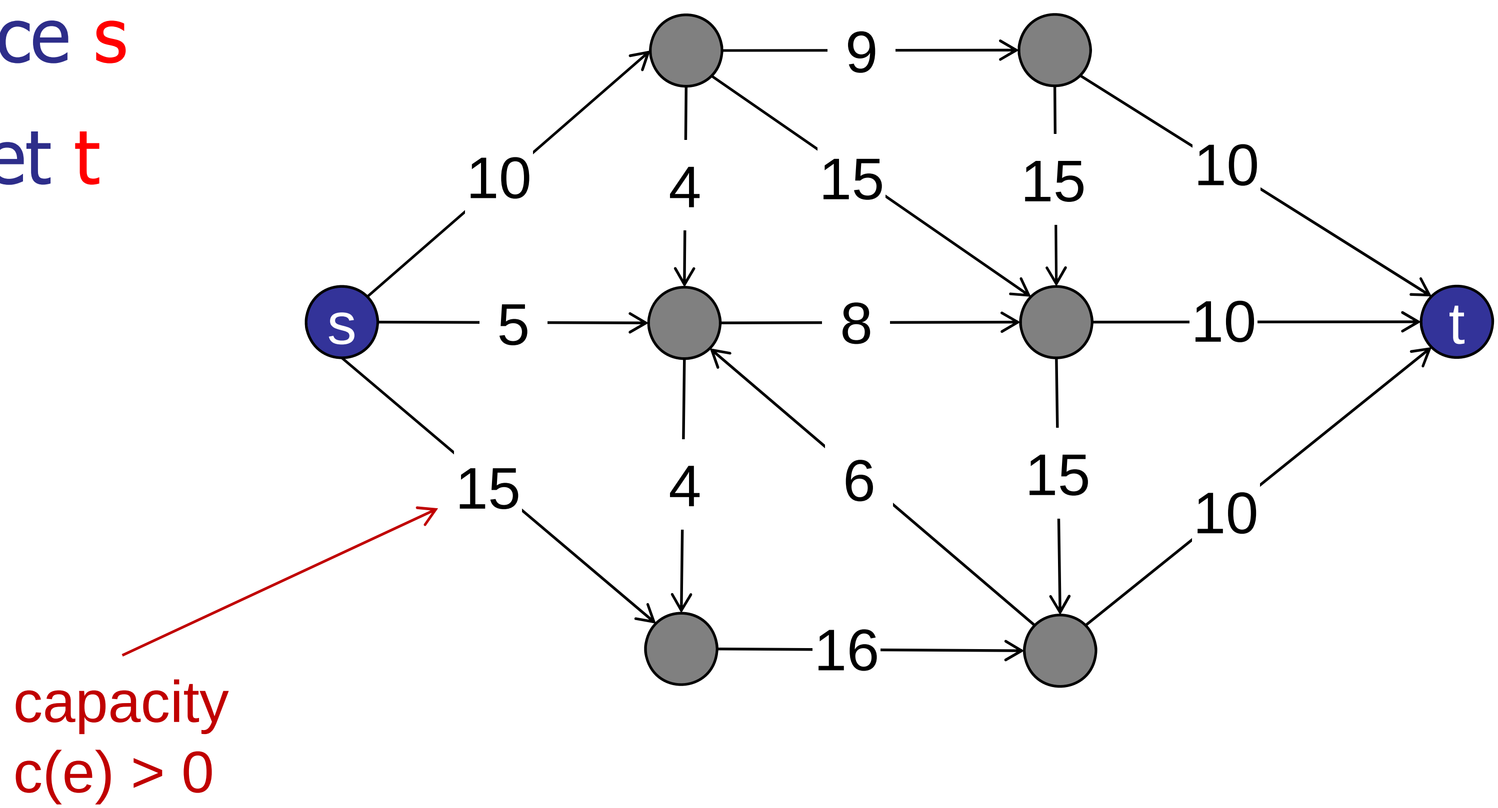
Examples:

- Transportation problems
- Distributed network reliability
- Network attacks
- Project selection
- Matching and assignment problems
- Image segmentation
- Sport’s teams prospects

Network Flow

Input:

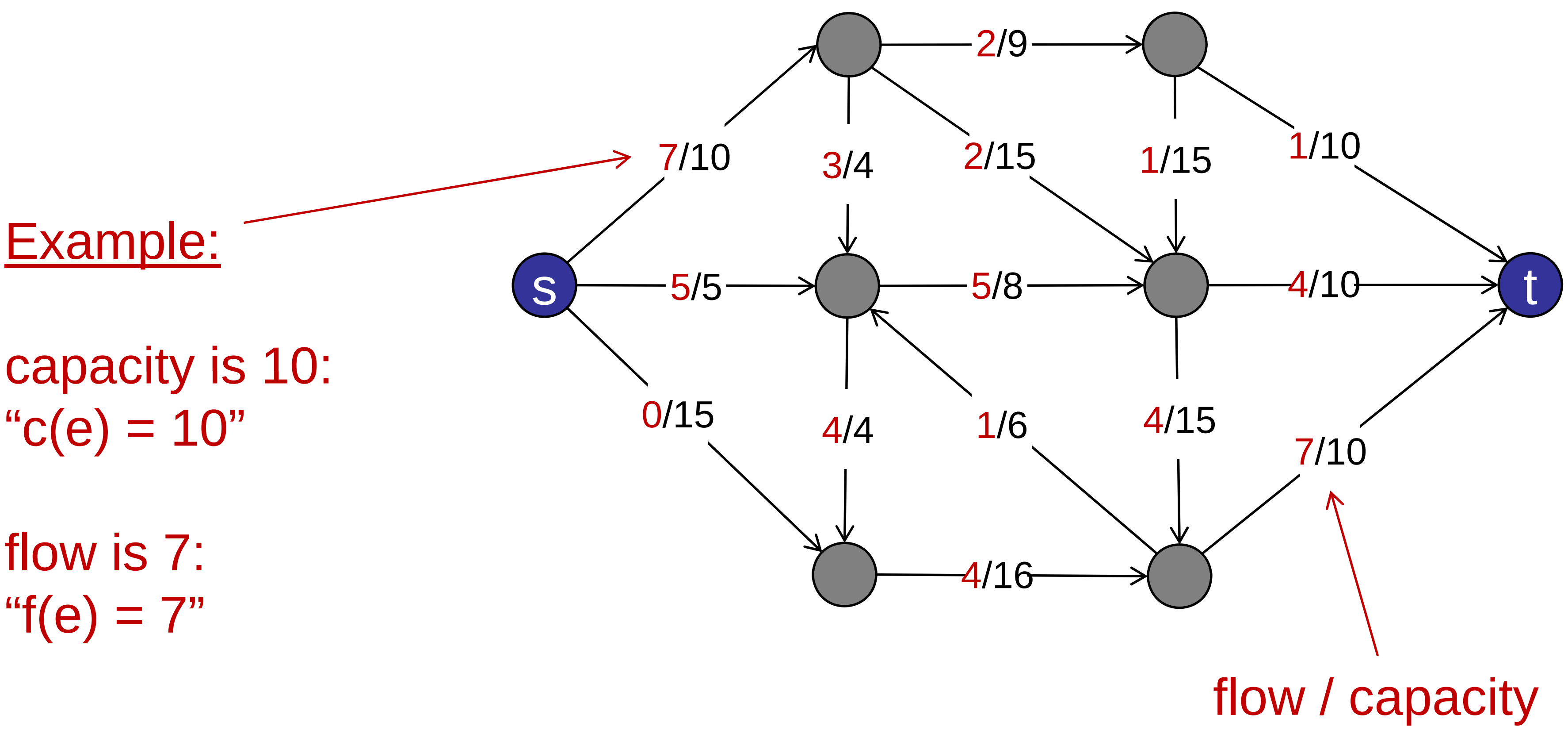
– directed graph G = (V,E)
– edge capacities c(e)
– source s
– target t



Network Flow

Output: Flow

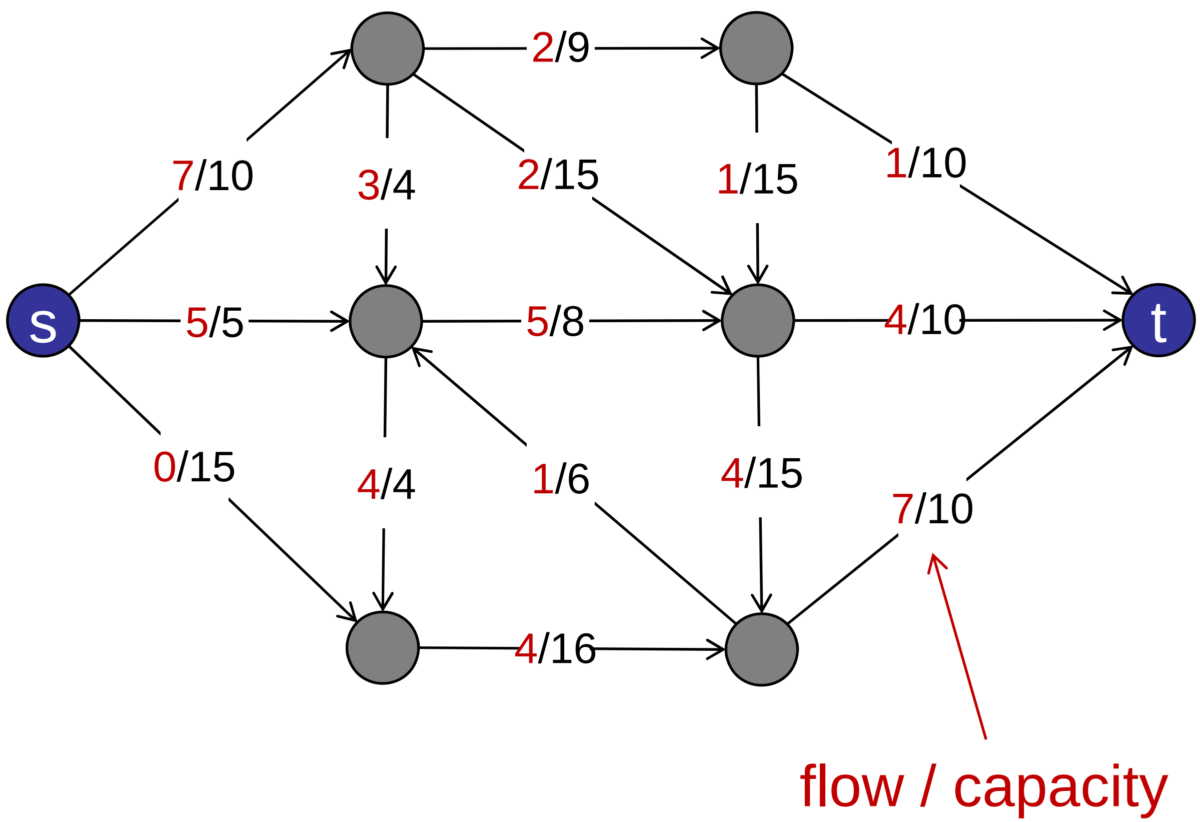
– Assignment of flow f to each edge



Network Flow

Output: Flow

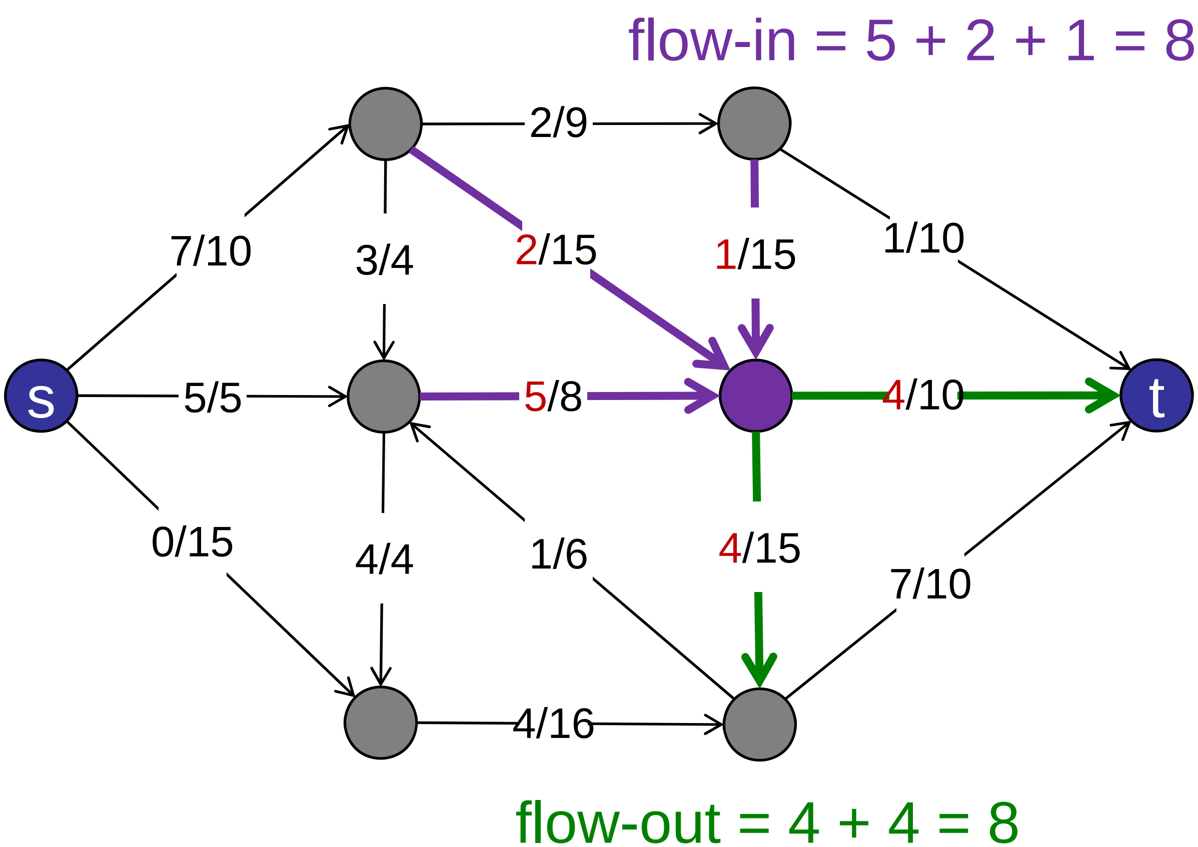
- Flow is not negative: for every edge e, $0 \le f(e)$
- Flow ≤ capacity: for every edge e, $f(e) \le c(e)$



Network Flow

Equilibrium constraint:

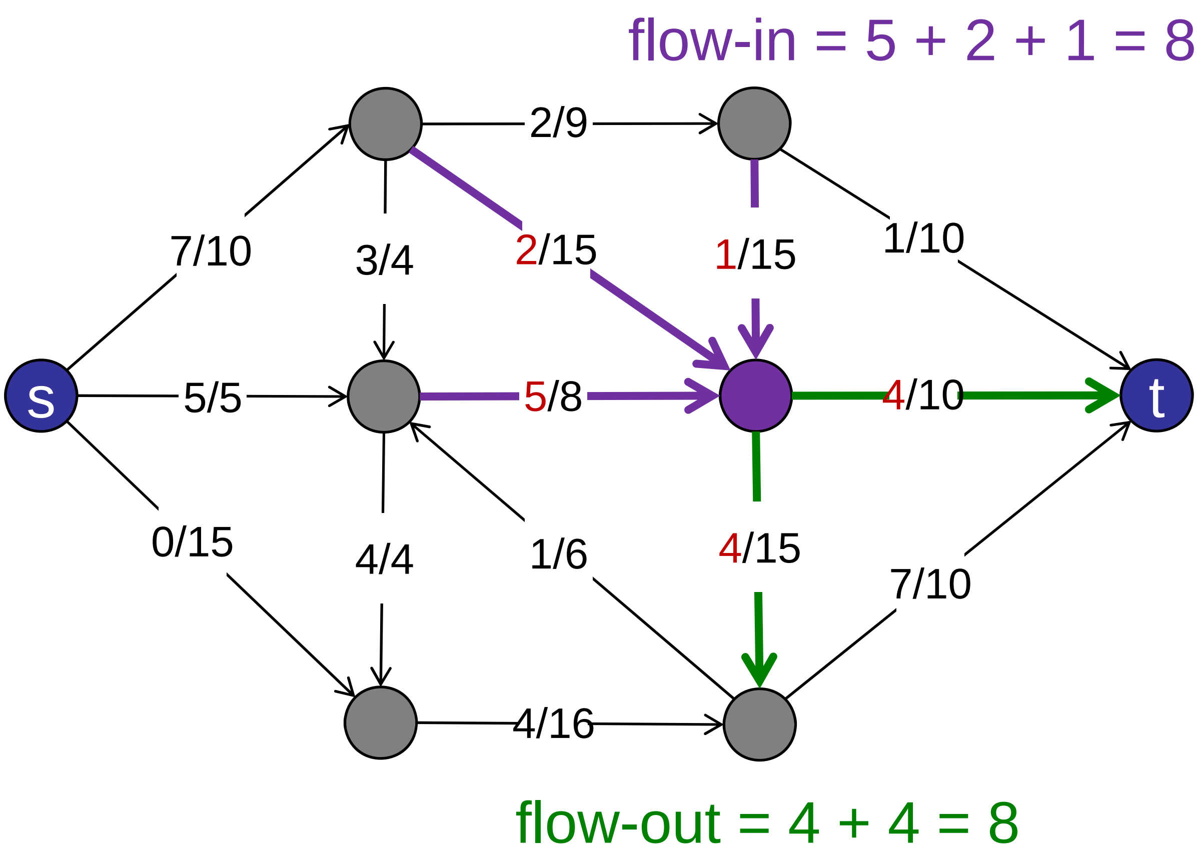
– For every node: flow-in = flow-out



Network Flow

Equilibrium constraint:

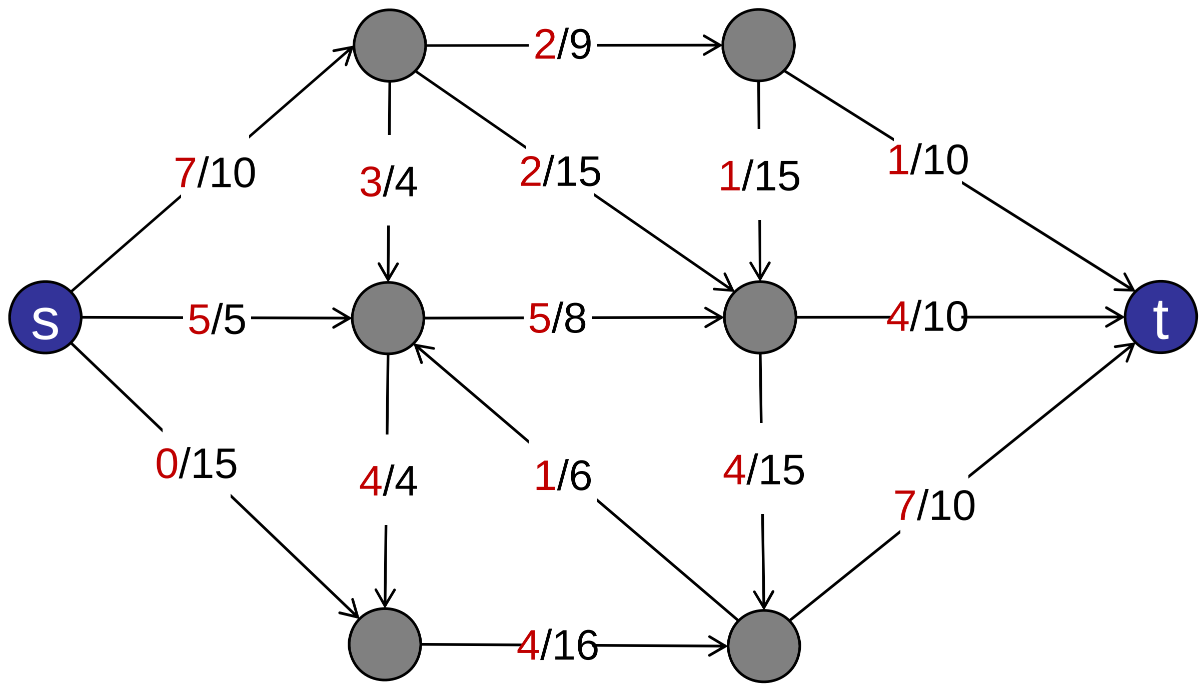
- For every node: flow-in = flow-out
- Except s and t



Network Flow

Output: st-flow

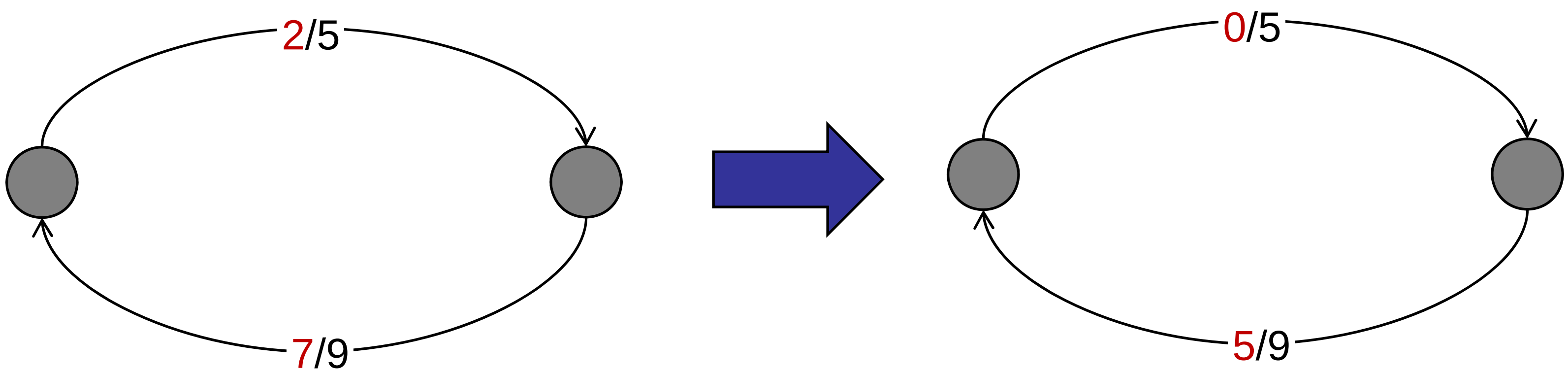
- Capacity constraint (never exceed capacity)
- Equilibrium constraint (flow-in = flow-out)



Network Flow

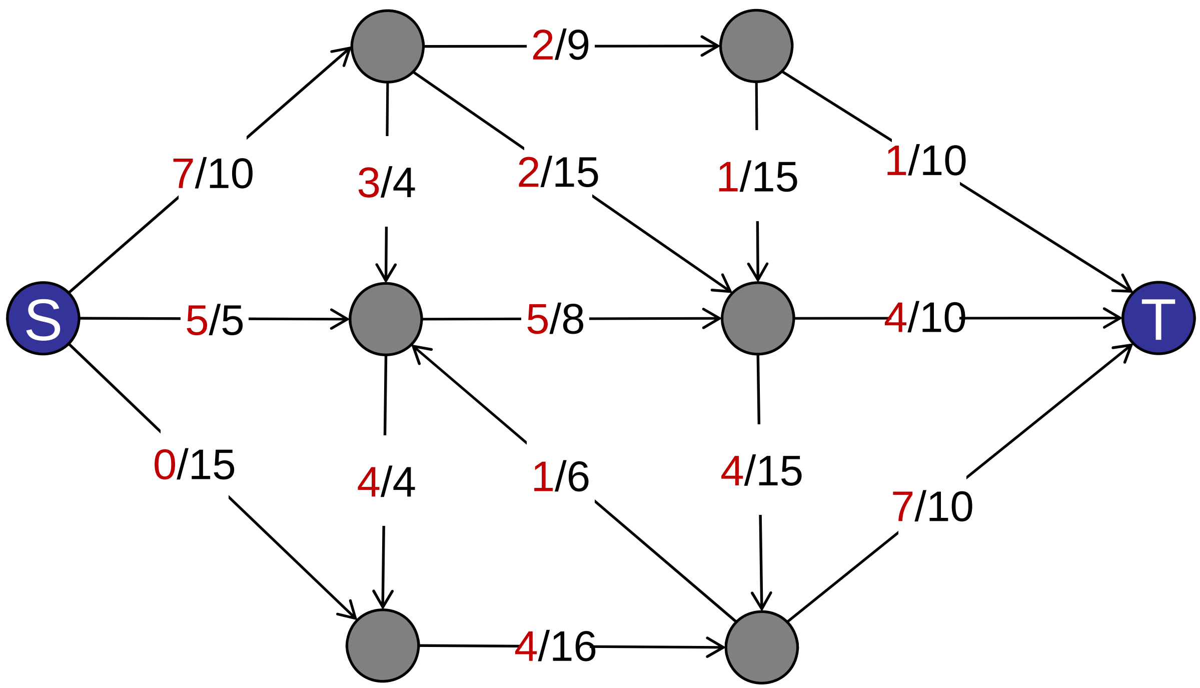
Uni-directional flow: st-flow

- If f(u,v)>0 and f(v,u)>0, then they cancel out.
- Flows only go in one direction.



Value of a Flow

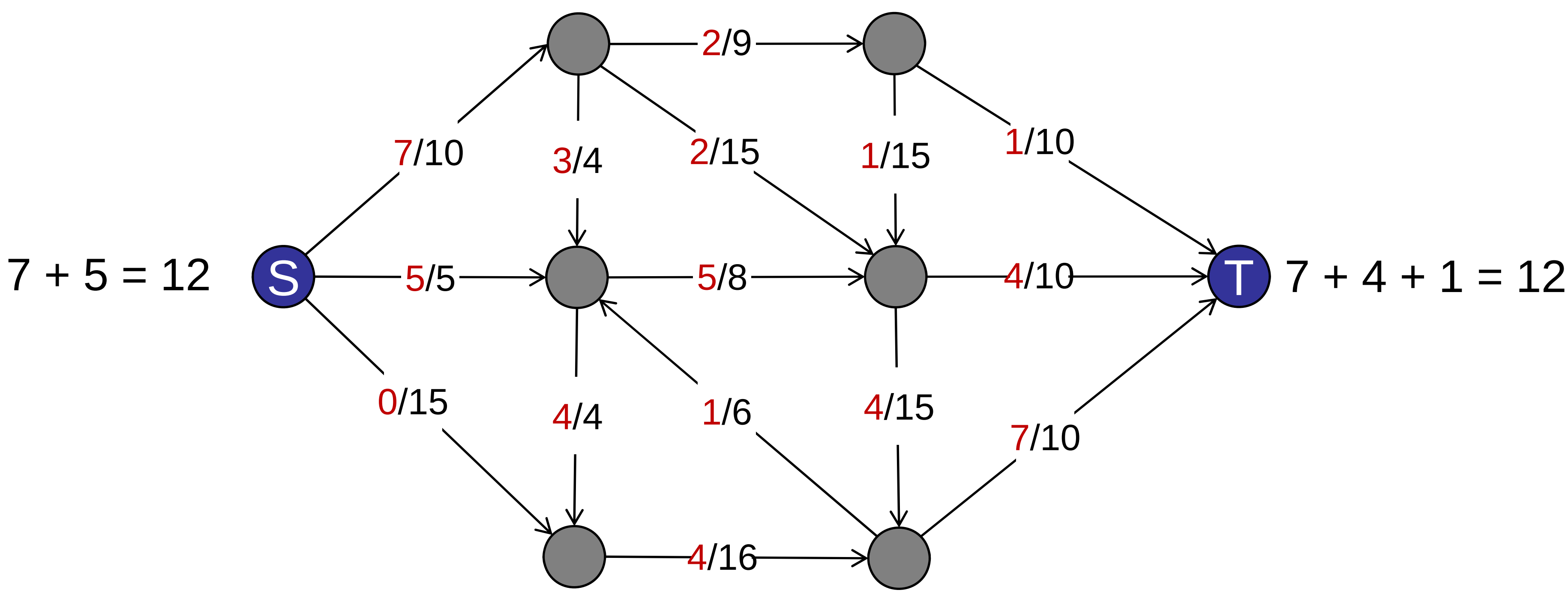
How much stuff gets from s to t?



Value of a Flow

How much stuff gets from s to t?

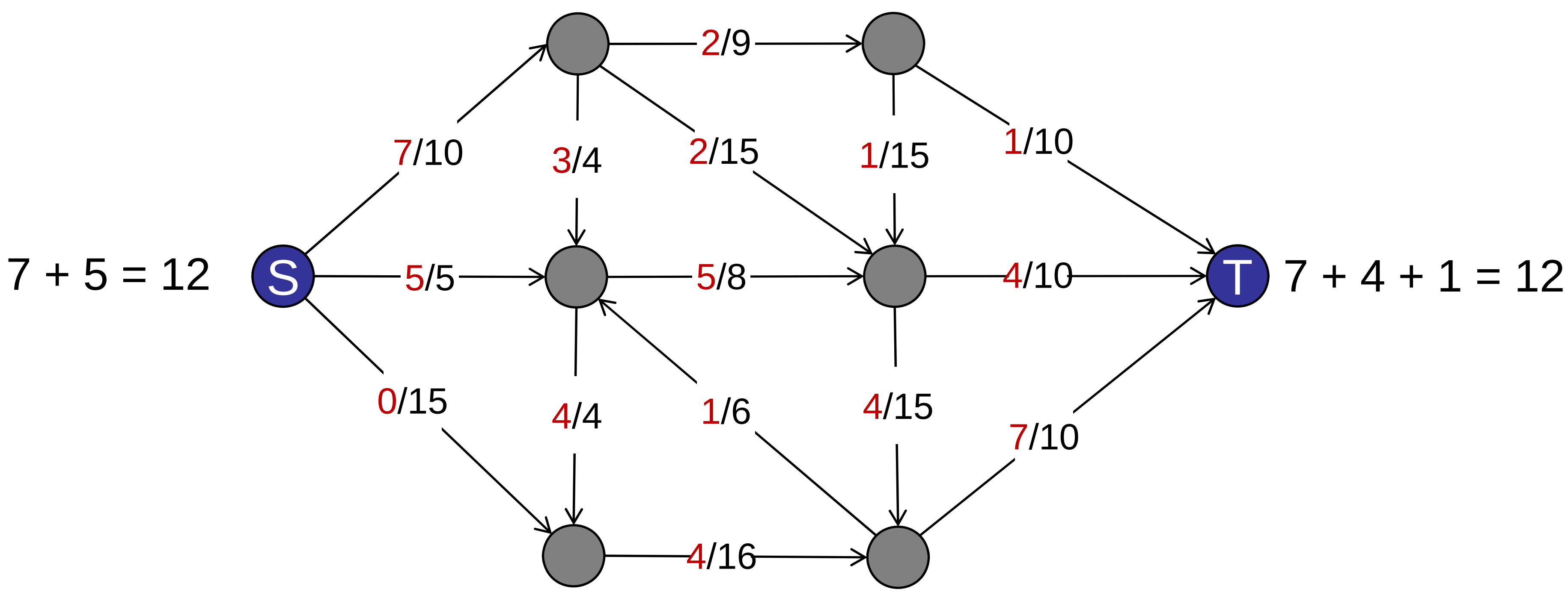
- How much leaves source?
- How much gets to target?



Value of a Flow

Definition:

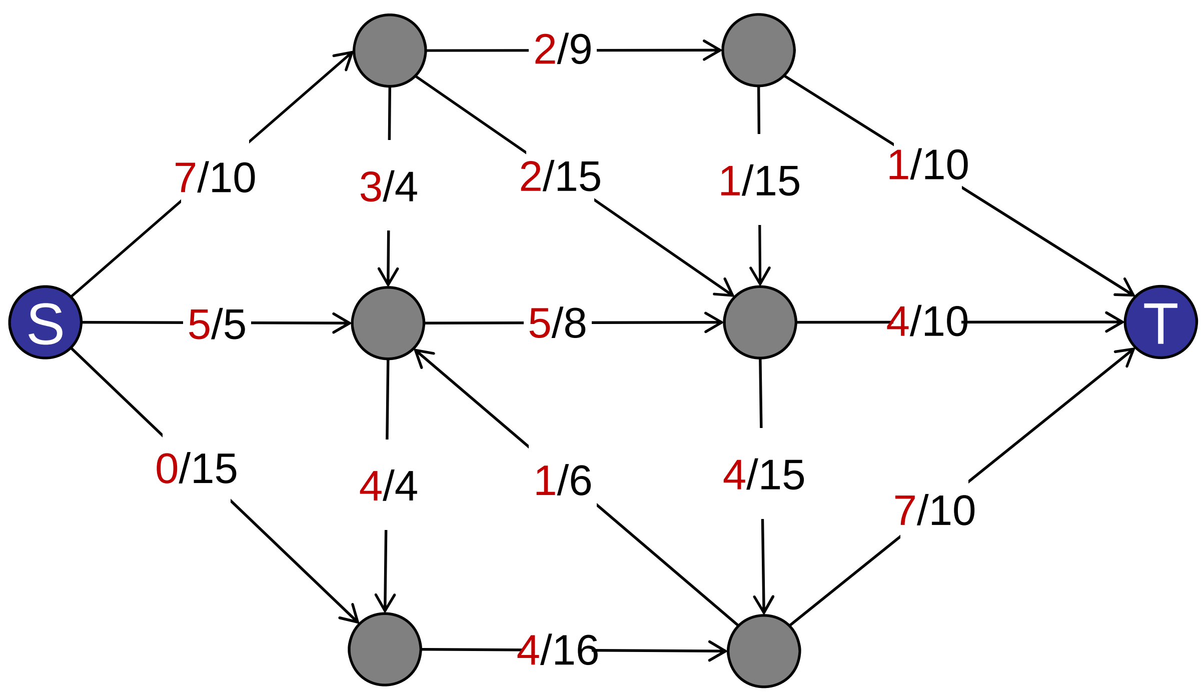
For a flow f: $\text{value}(f) = \sum_{v:(s,v)\in E} f(s,v)$



Max Flow

Goal:

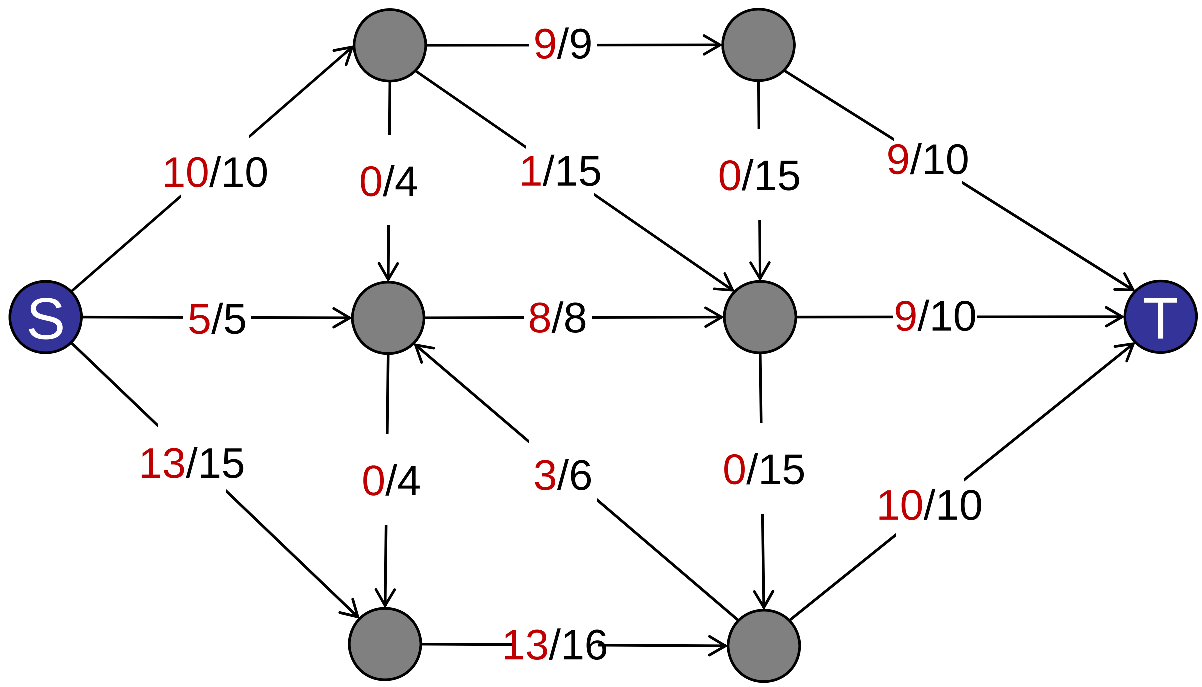
Find an st-flow with maximum value.



Max Flow

Goal:

Find an st-flow with maximum value.



value = 28

Roadmap

Network Flows

a. Network flows defined

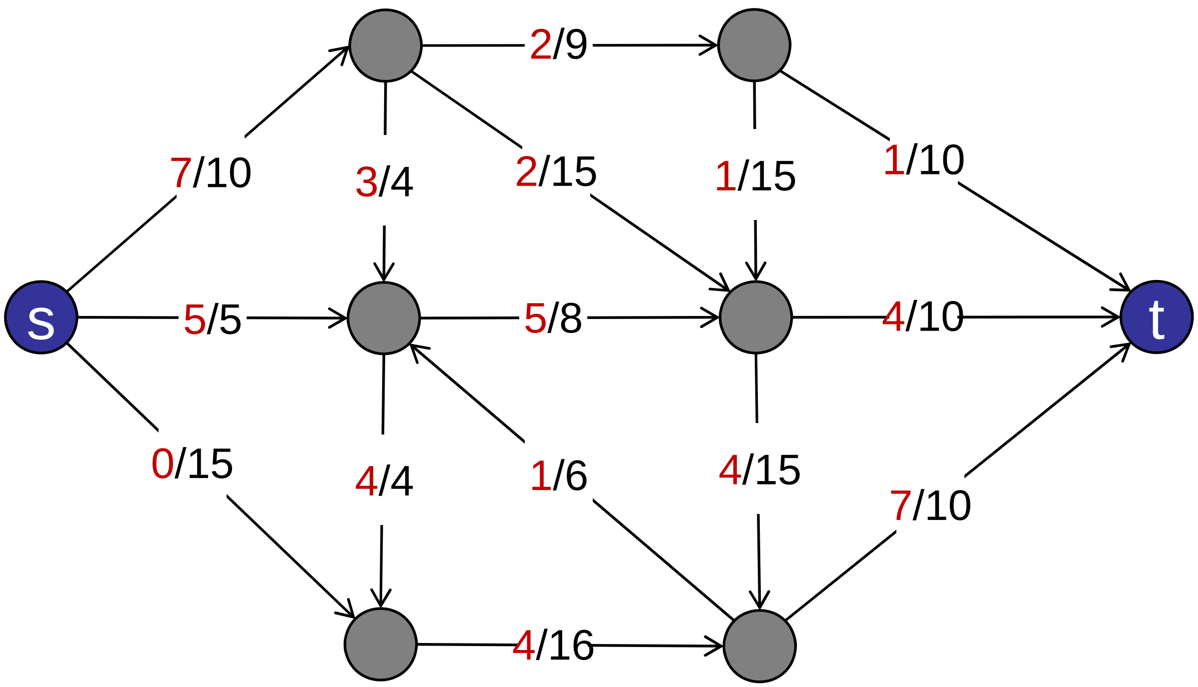
b. Ford-Fulkerson algorithm

c. Max-Flow / Min-Cut Theorem

Max Flow

Goal:

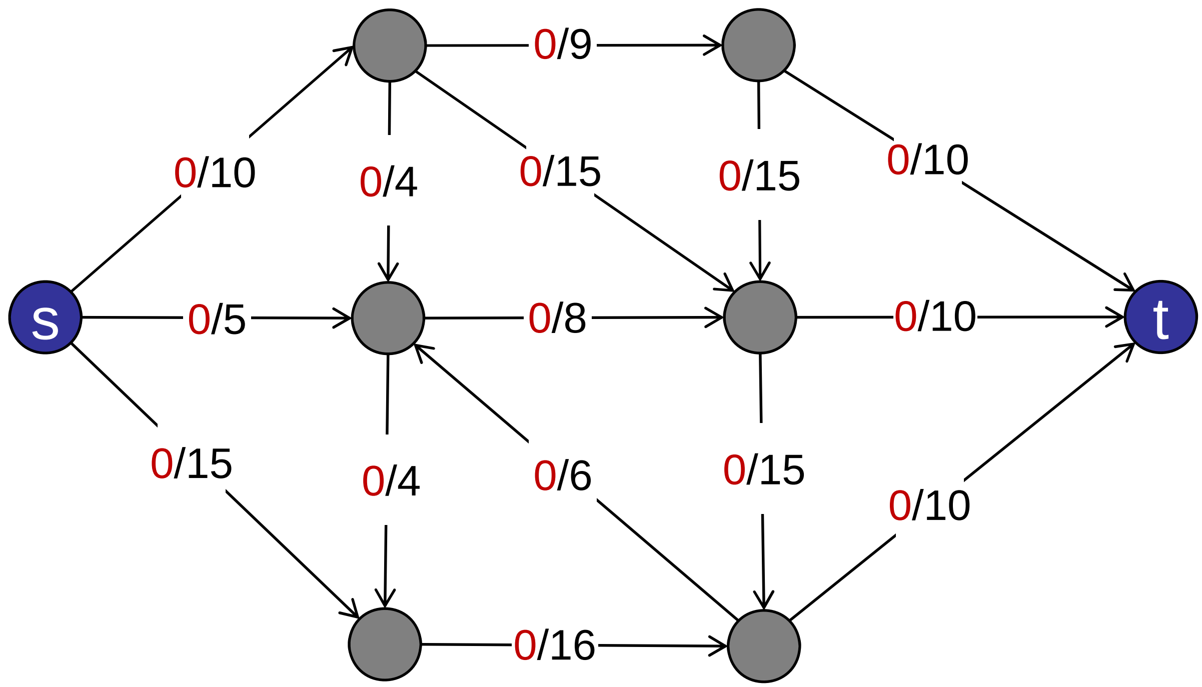
Find an st-flow with maximum value.



Ford-Fulkerson

Initially:

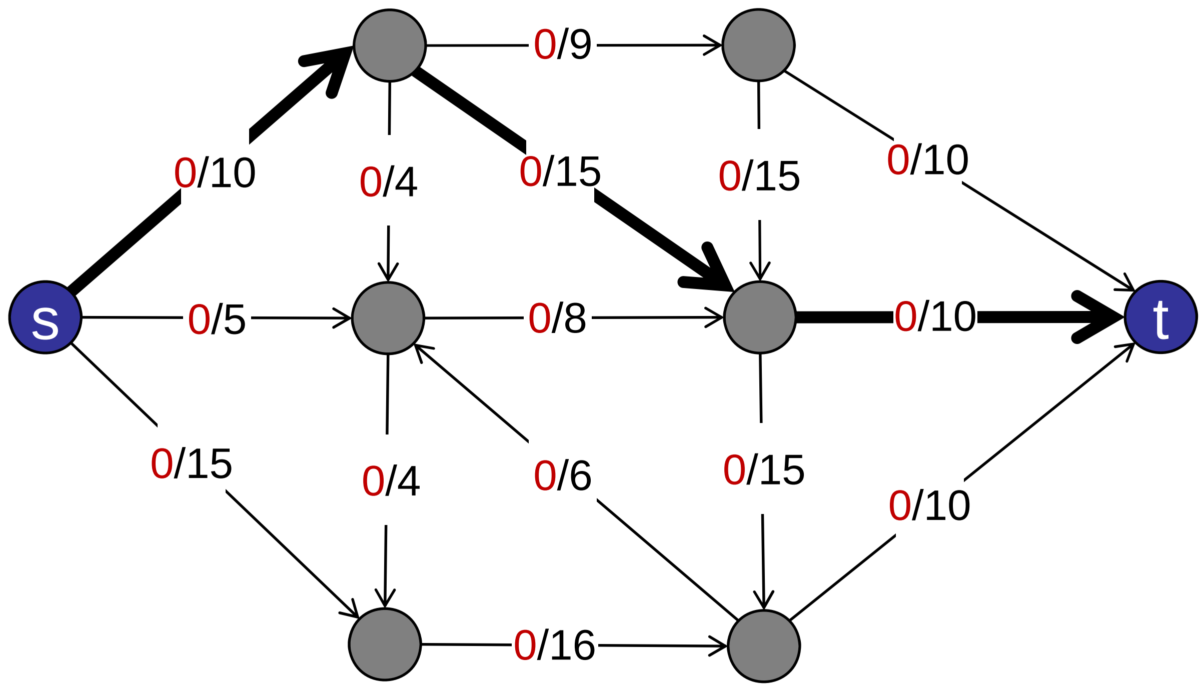
All flows are 0.



value = 0

Ford-Fulkerson

Idea: find an augmenting path along which we can increase the flow.

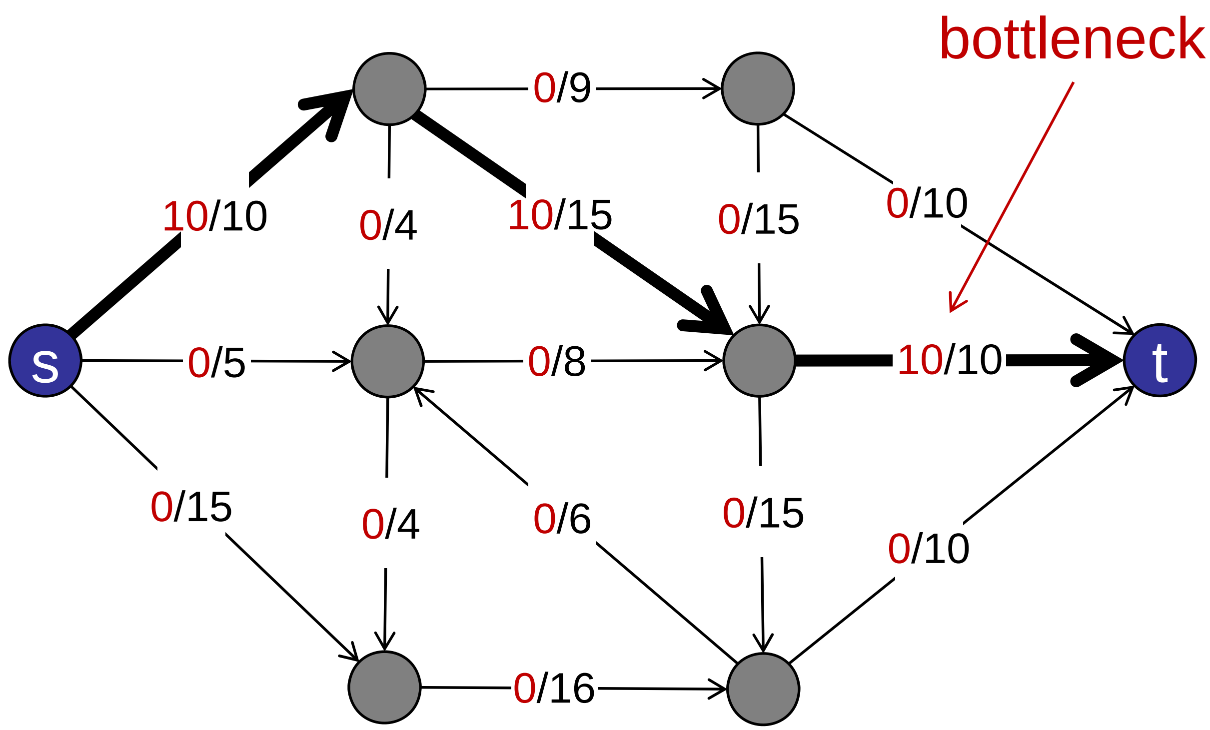


value = 0

Ford-Fulkerson

Augmenting path: directed path from s → t

- Can increase flow on all forward edges.

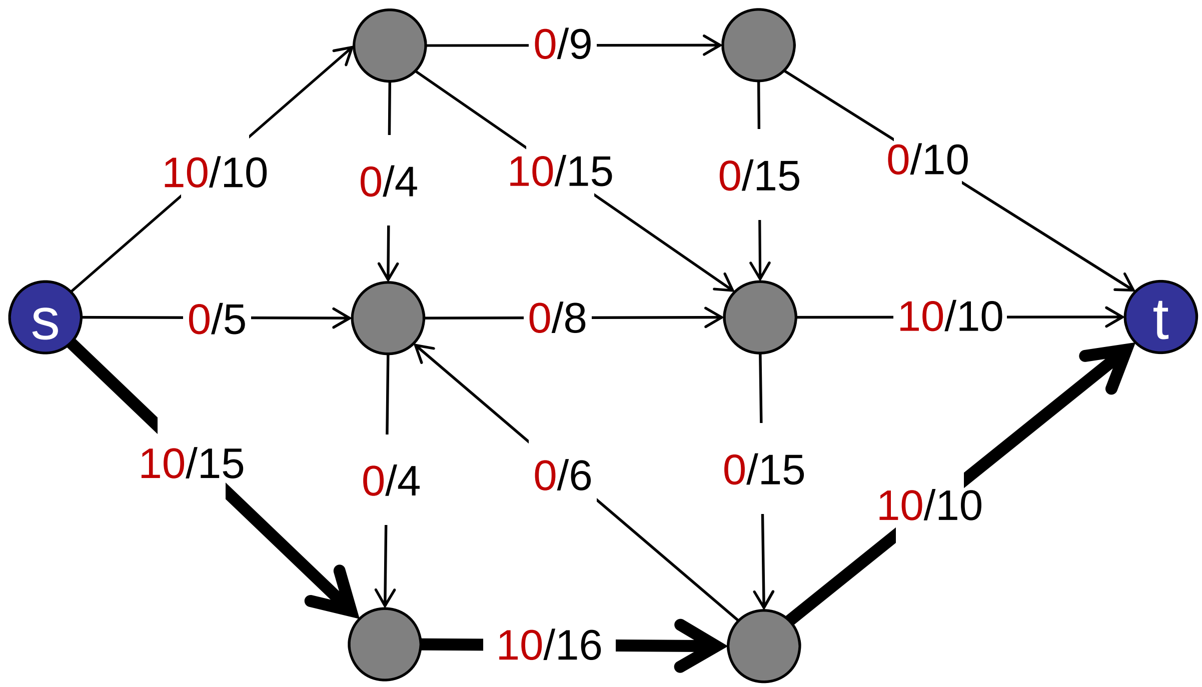


value = 0 + 10

Ford-Fulkerson

Augmenting path: directed path from s → t

- Can increase flow on all forward edges.

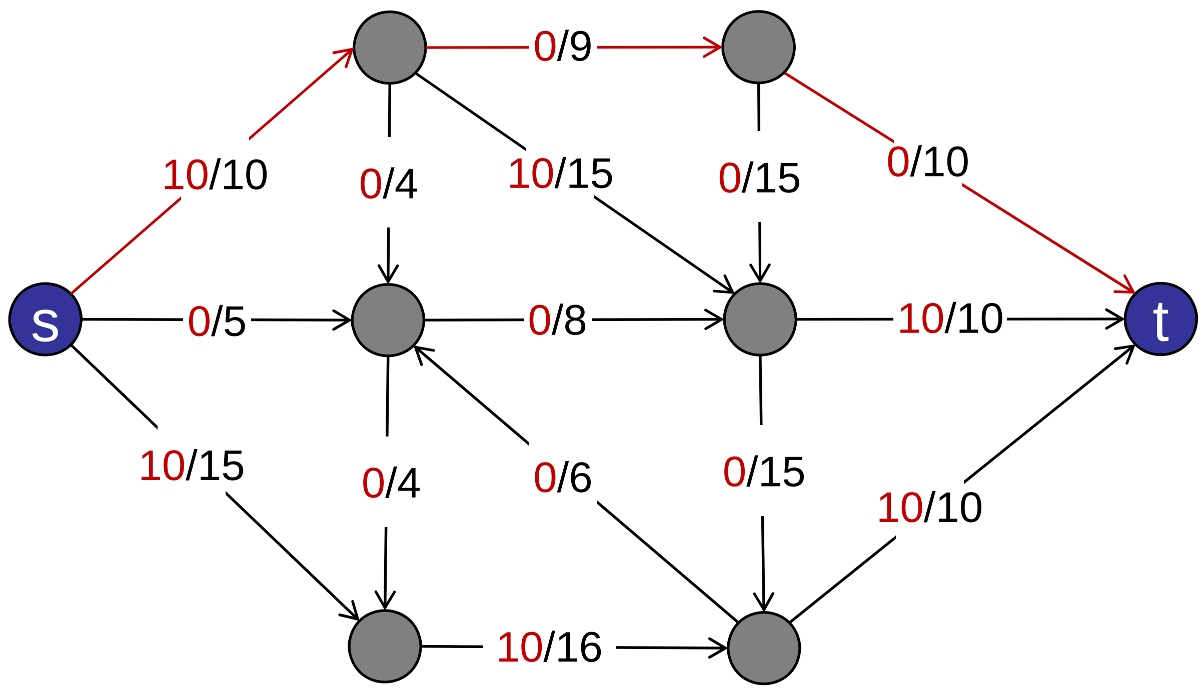


value = 0 + 10 + 10

Ford-Fulkerson

Augmenting path: directed path from s → t

- Can increase flow on all forward edges.
- No more augmenting paths?

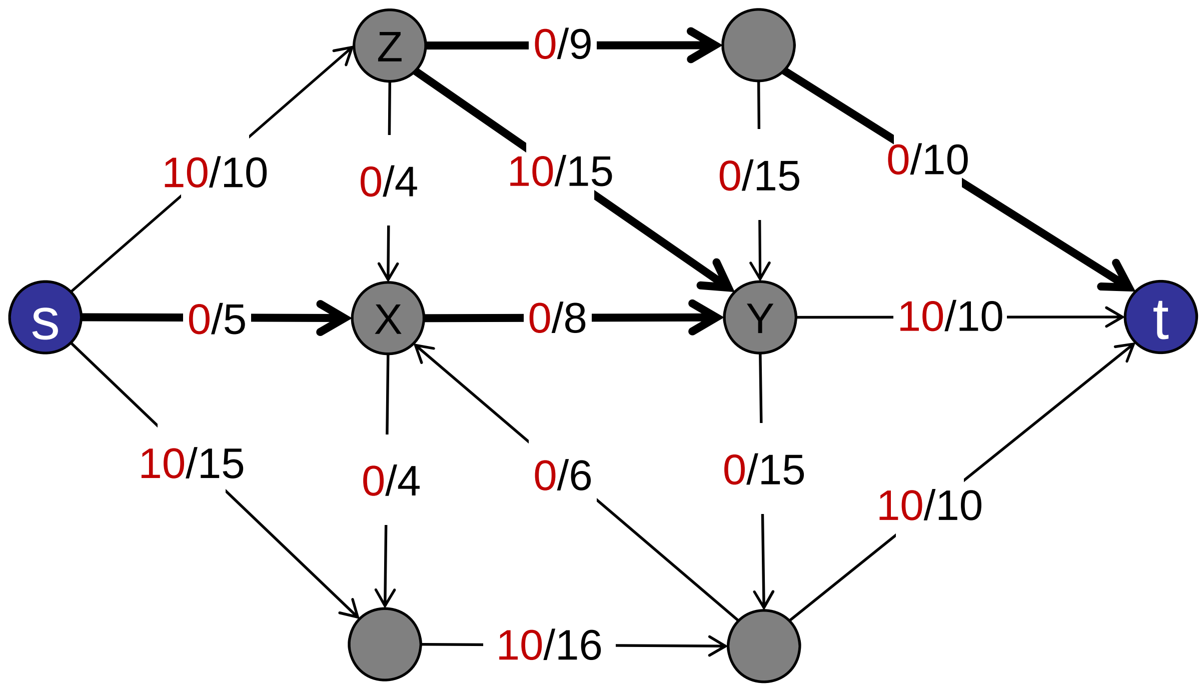


value = 0 + 10 + 10

Ford-Fulkerson

Augmenting path: directed path from s → t

- Can increase flow on all forward edges.

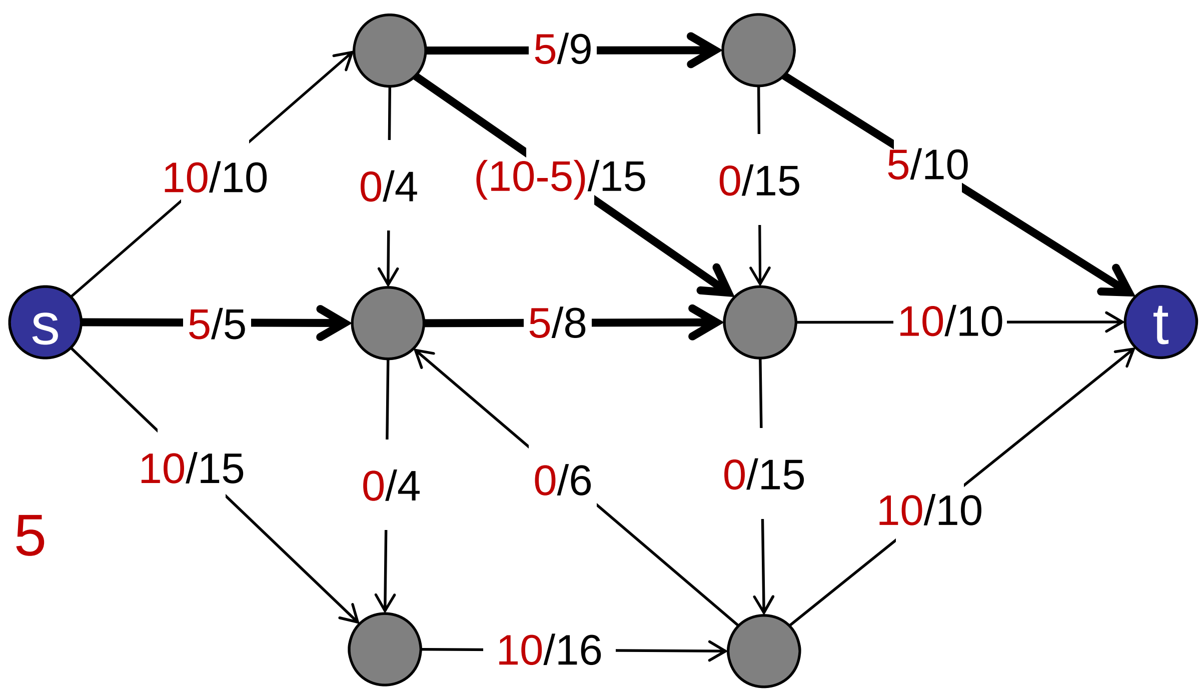


value = 0 + 10 + 10

Ford-Fulkerson

Augmenting path: Undirected path from s → t

- Can increase flow on all forward edges OR
- Can decrease flow on backward edges

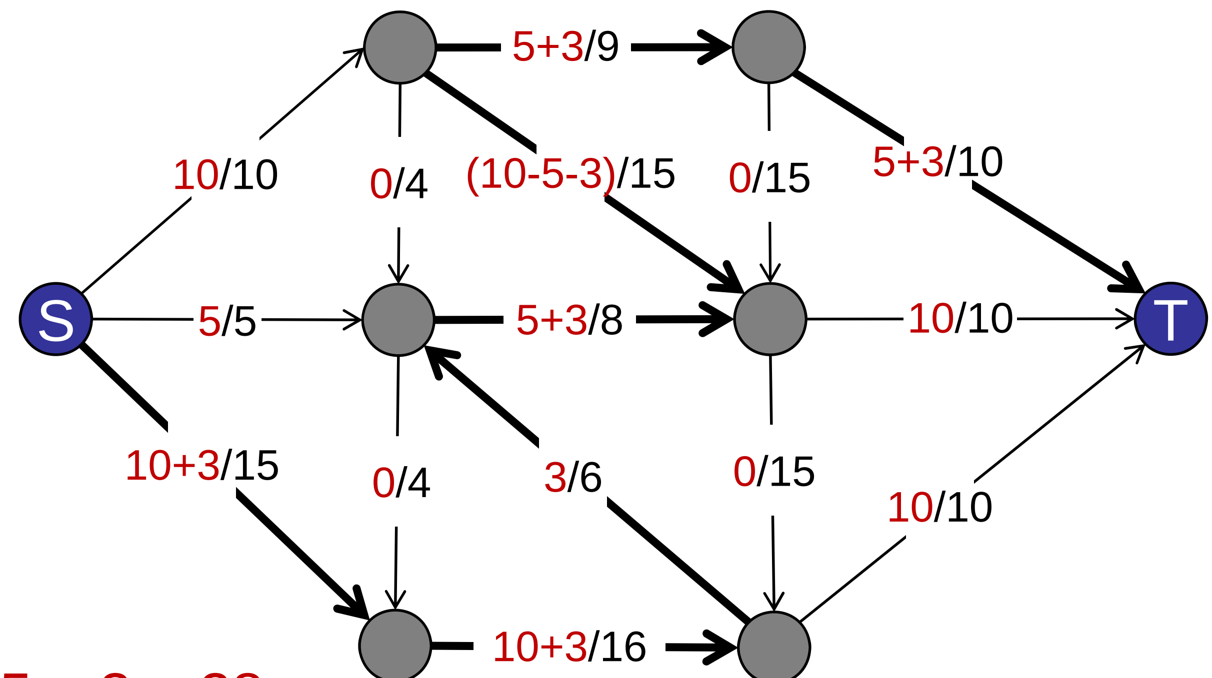


value = 0 + 10 + 10 + 5

Ford-Fulkerson

Augmenting path: Undirected path from s → t

- Can increase flow on all forward edges OR
- Can decrease flow on backward edges

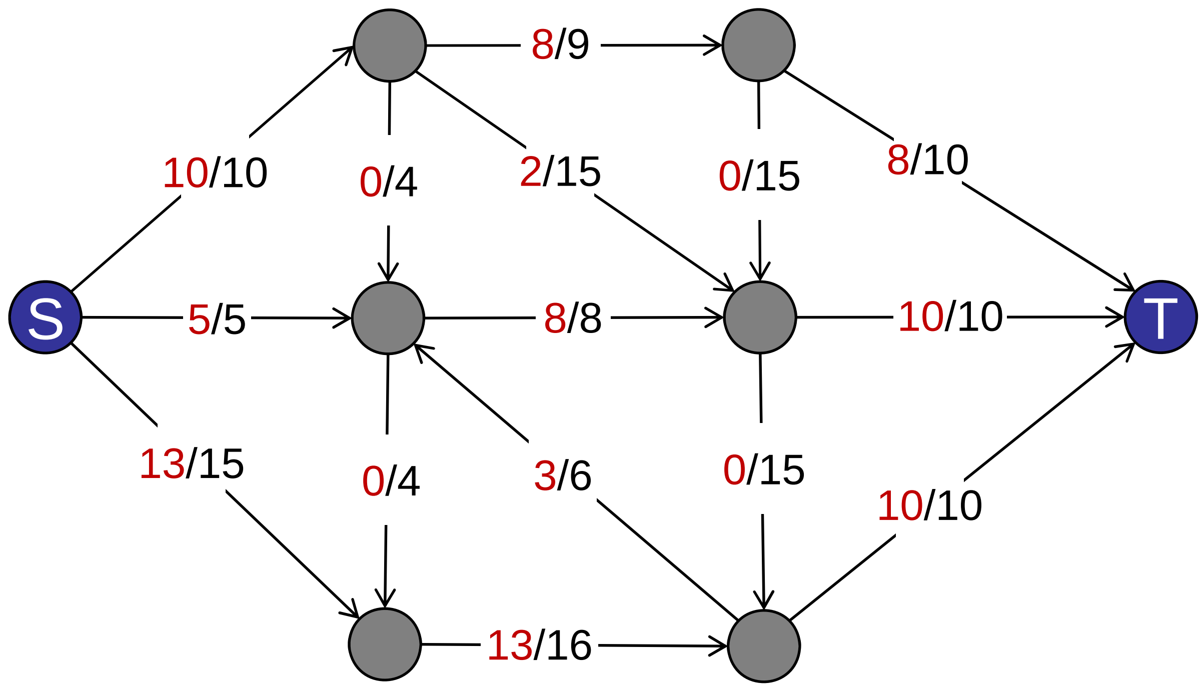


value = 0 + 10 + 10 + 5 + 3 = 28

Ford-Fulkerson

Augmenting path: Undirected path from s → t

- Can increase flow on all forward edges OR
- Can decrease flow on backward edges



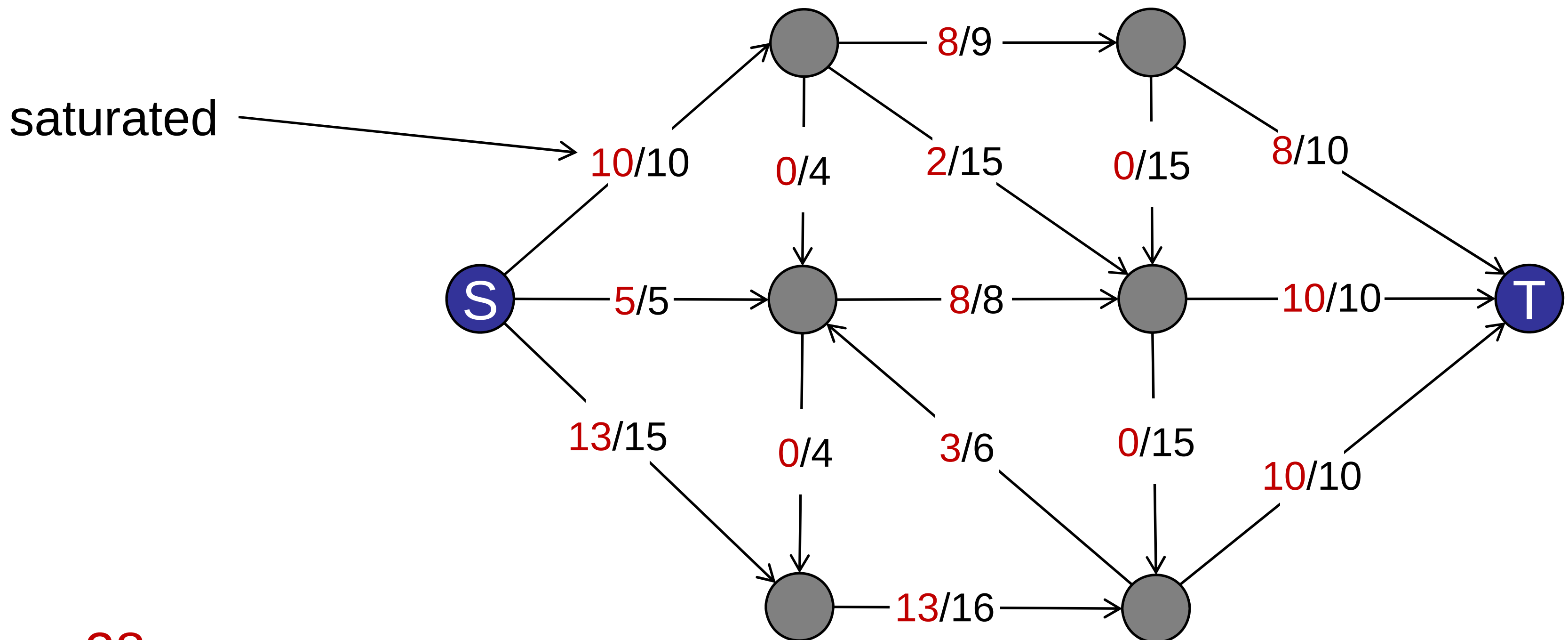
value = 28

No more augmenting paths.

Ford-Fulkerson

Augmenting path: Undirected path from s → t

- Can increase flow on all forward edges OR
- Can decrease flow on backward edges

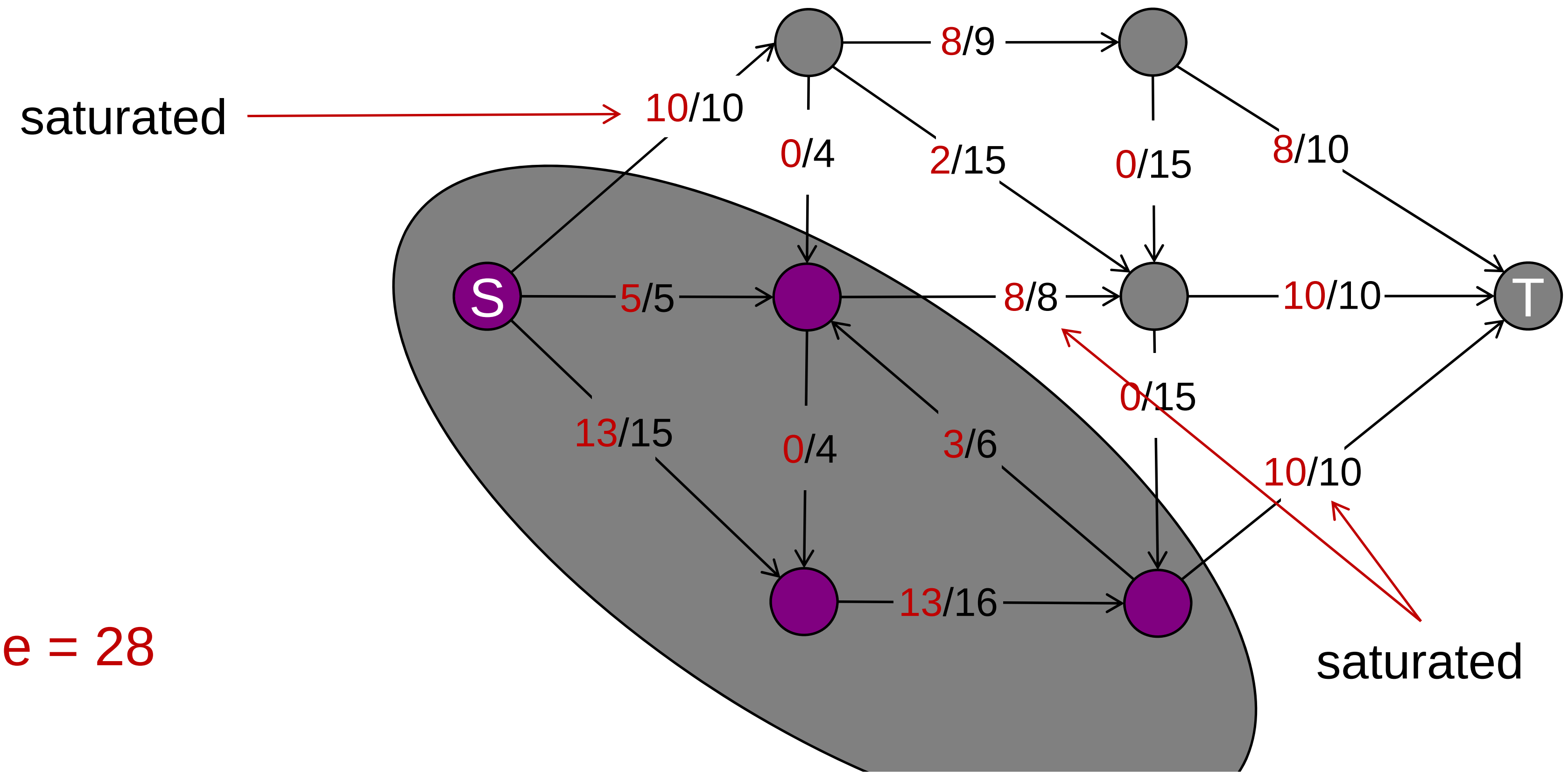


value = 28

Ford-Fulkerson

Augmenting path: Undirected path from s → t

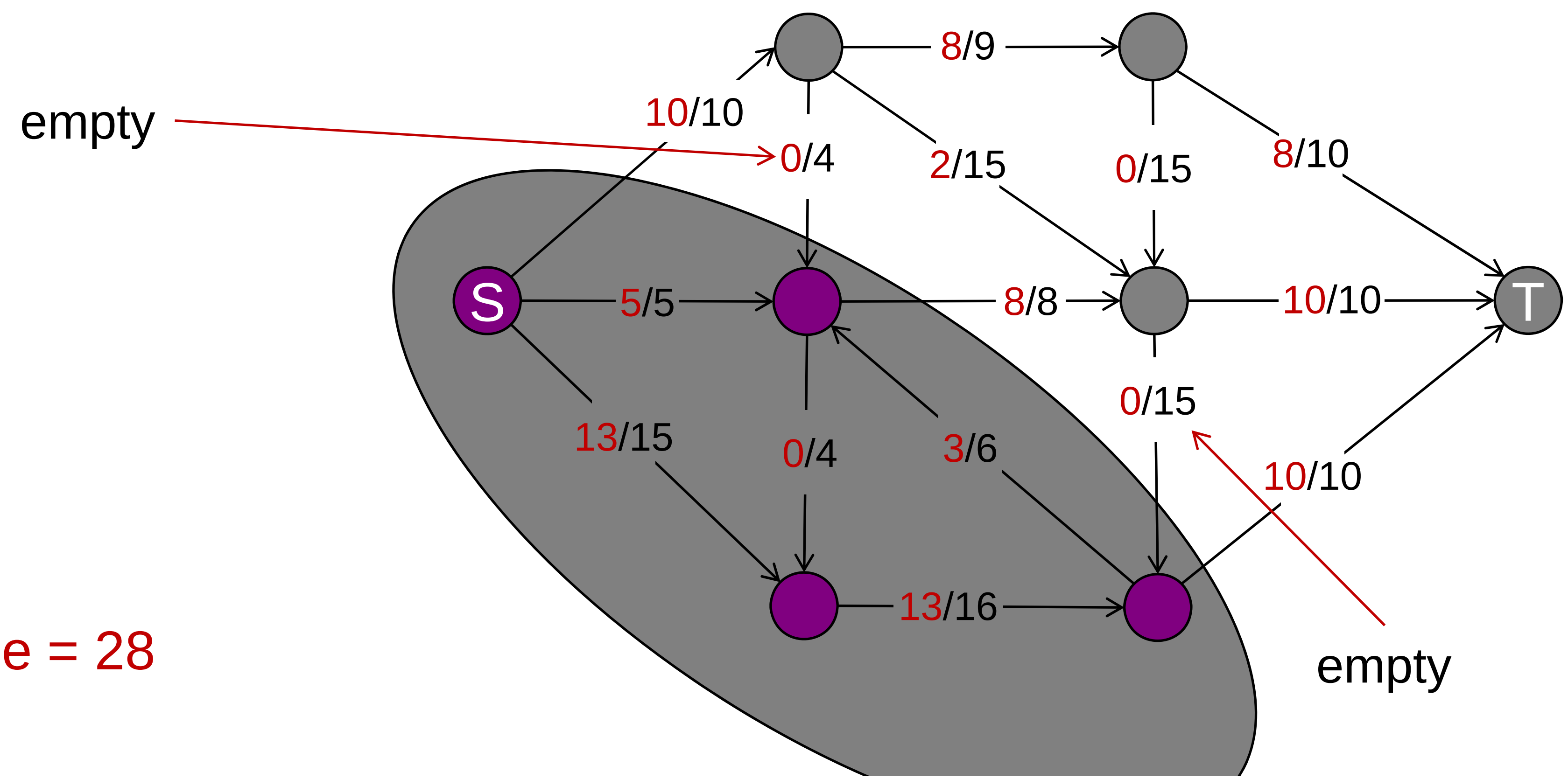
- Can increase flow on all forward edges OR
- Can decrease flow on backward edges



Ford-Fulkerson

Augmenting path: Undirected path from s → t

- Can increase flow on all forward edges OR
- Can decrease flow on backward edges

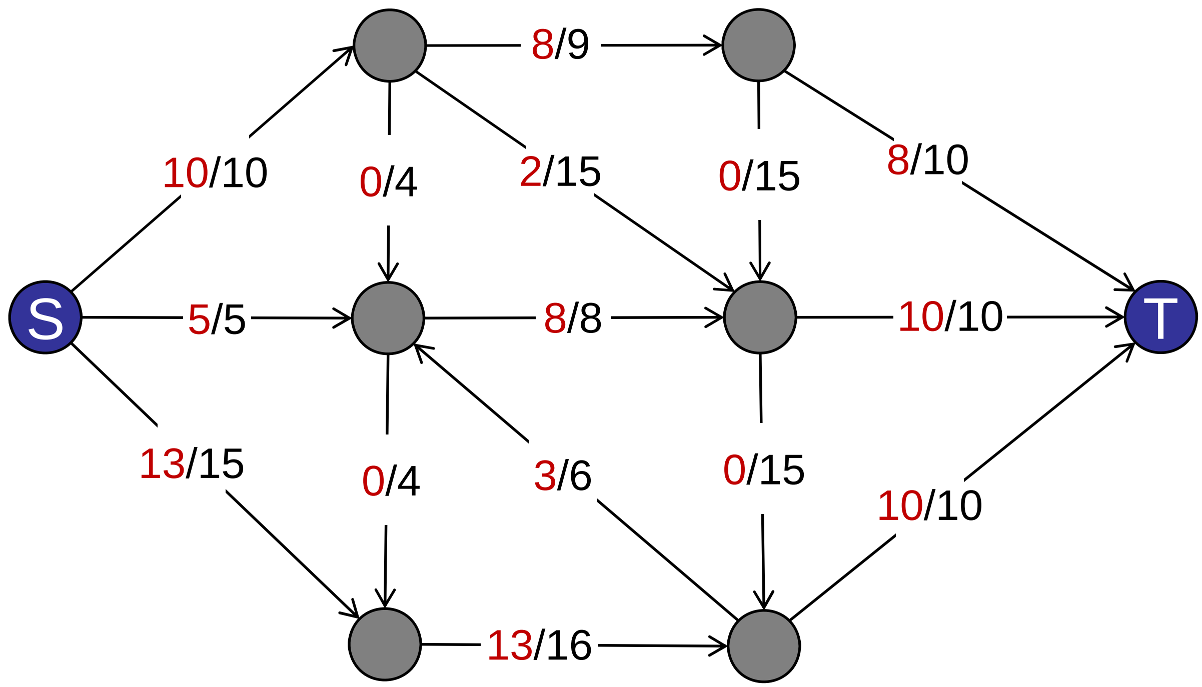


value = 28

Ford-Fulkerson

Augmenting path: Undirected path from s → t

- Can increase flow on all forward edges OR
- Can decrease flow on backward edges



value = 28

No more augmenting paths.

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:

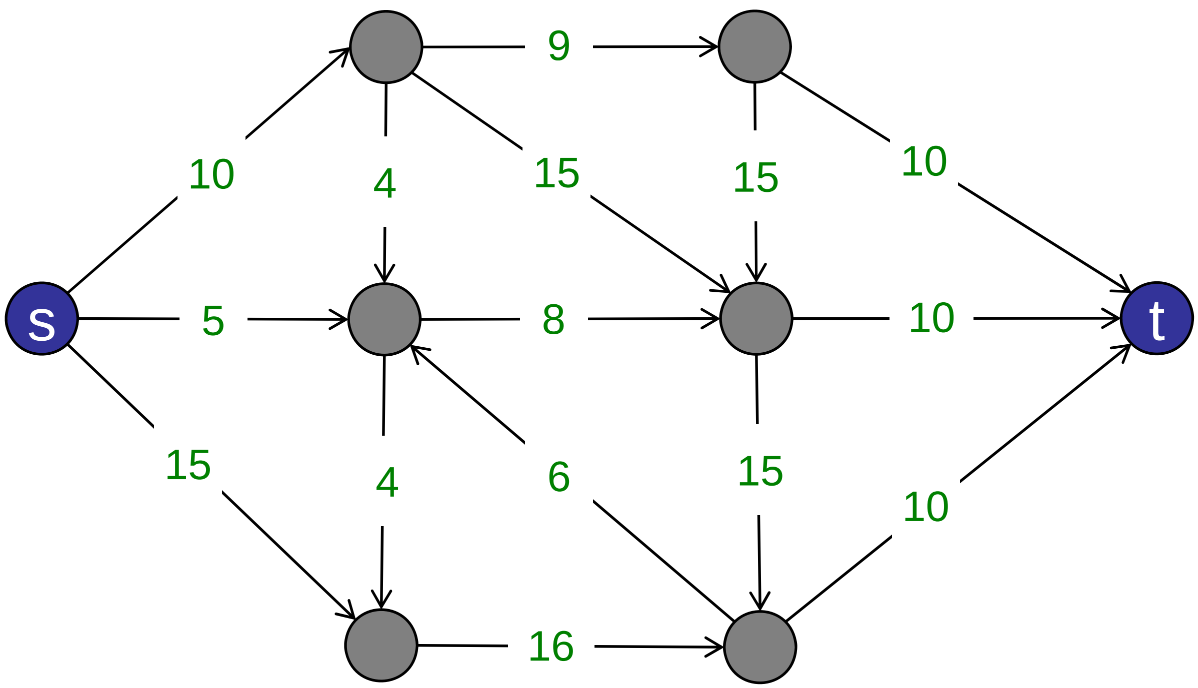
- How to find an augmenting path?
- Does Ford-Fulkerson always terminate? How fast?
- If it terminates, does it always find a max-flow?

Finding an Augmenting Path

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)

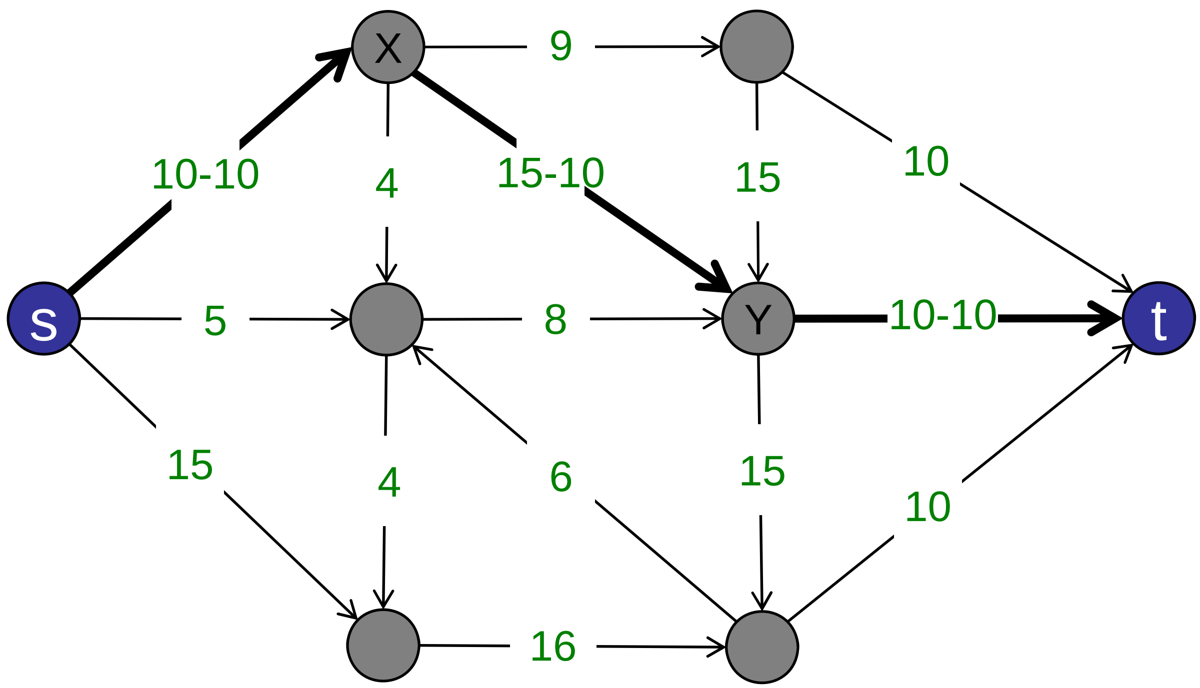


Initial graph: flow = 0 ➔ residual = capacity.

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)

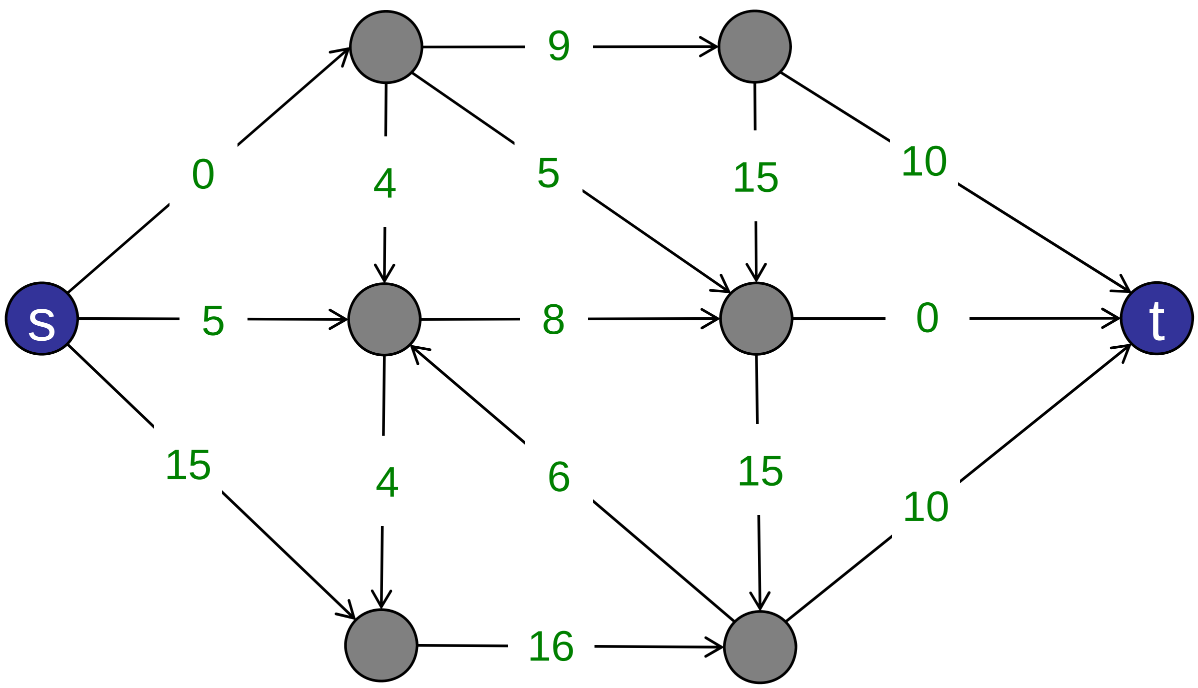


Step 1: augmenting path of flow 10.

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)



After augmenting path of flow 10.

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

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

For now:

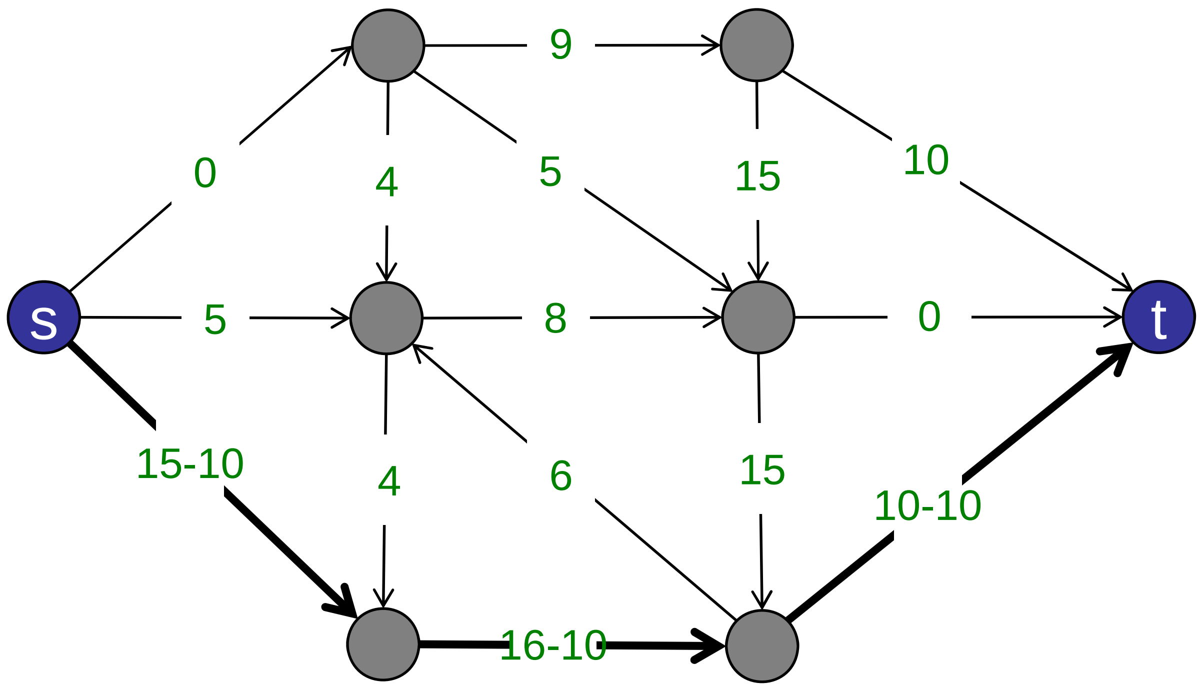
Any graph search will do (BFS, DFS, etc.)

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

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)

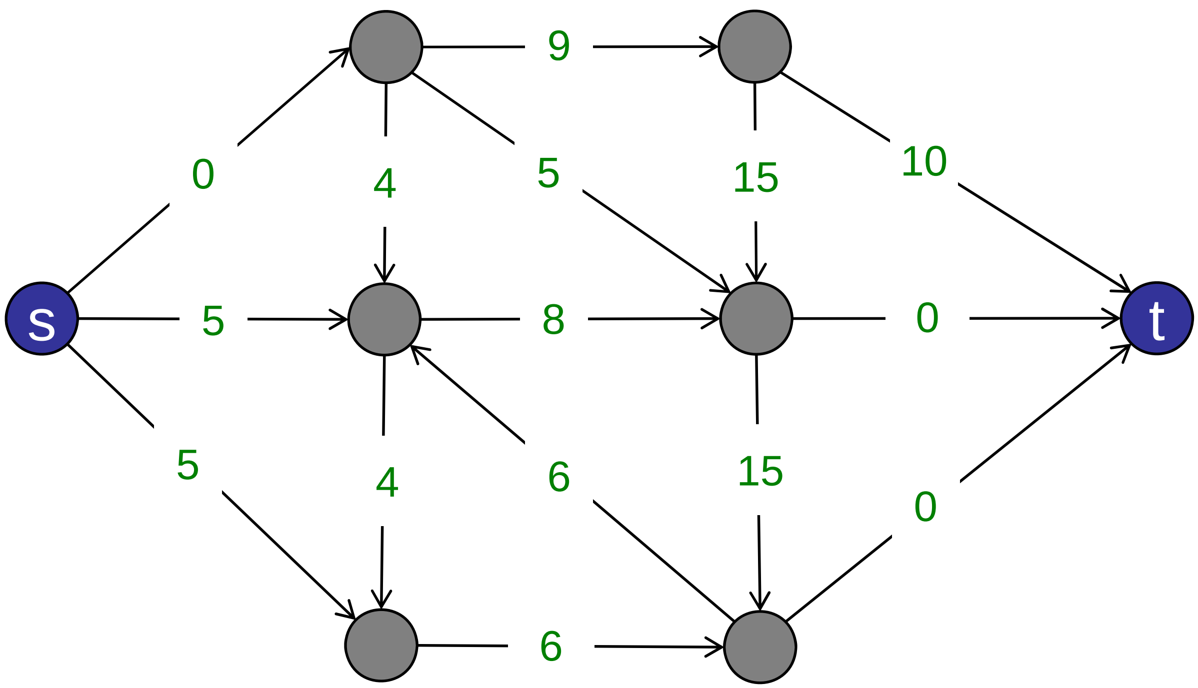


Step 2: augmenting path of flow 10.

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)

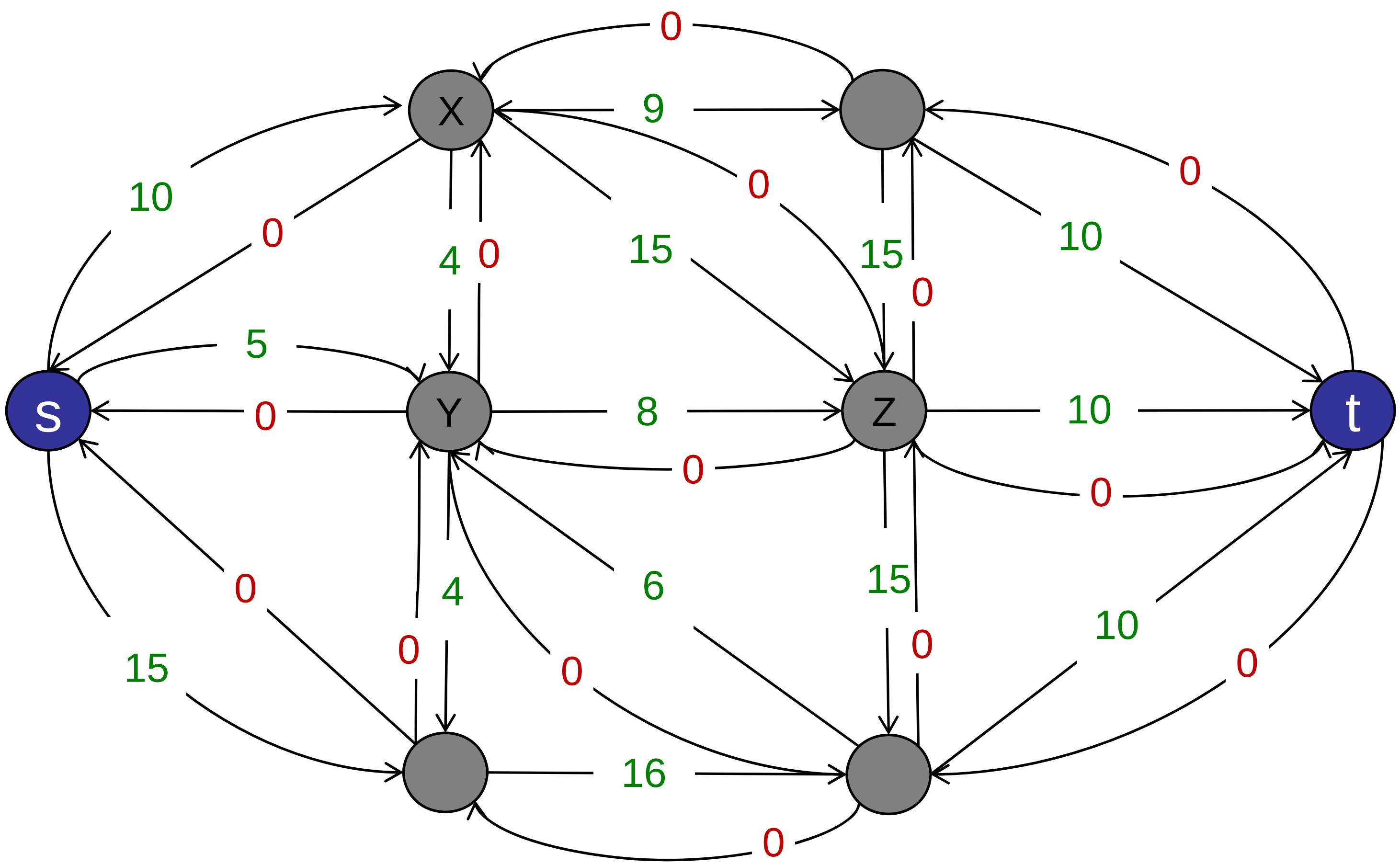


After step 2: augmenting path of flow 10.

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)

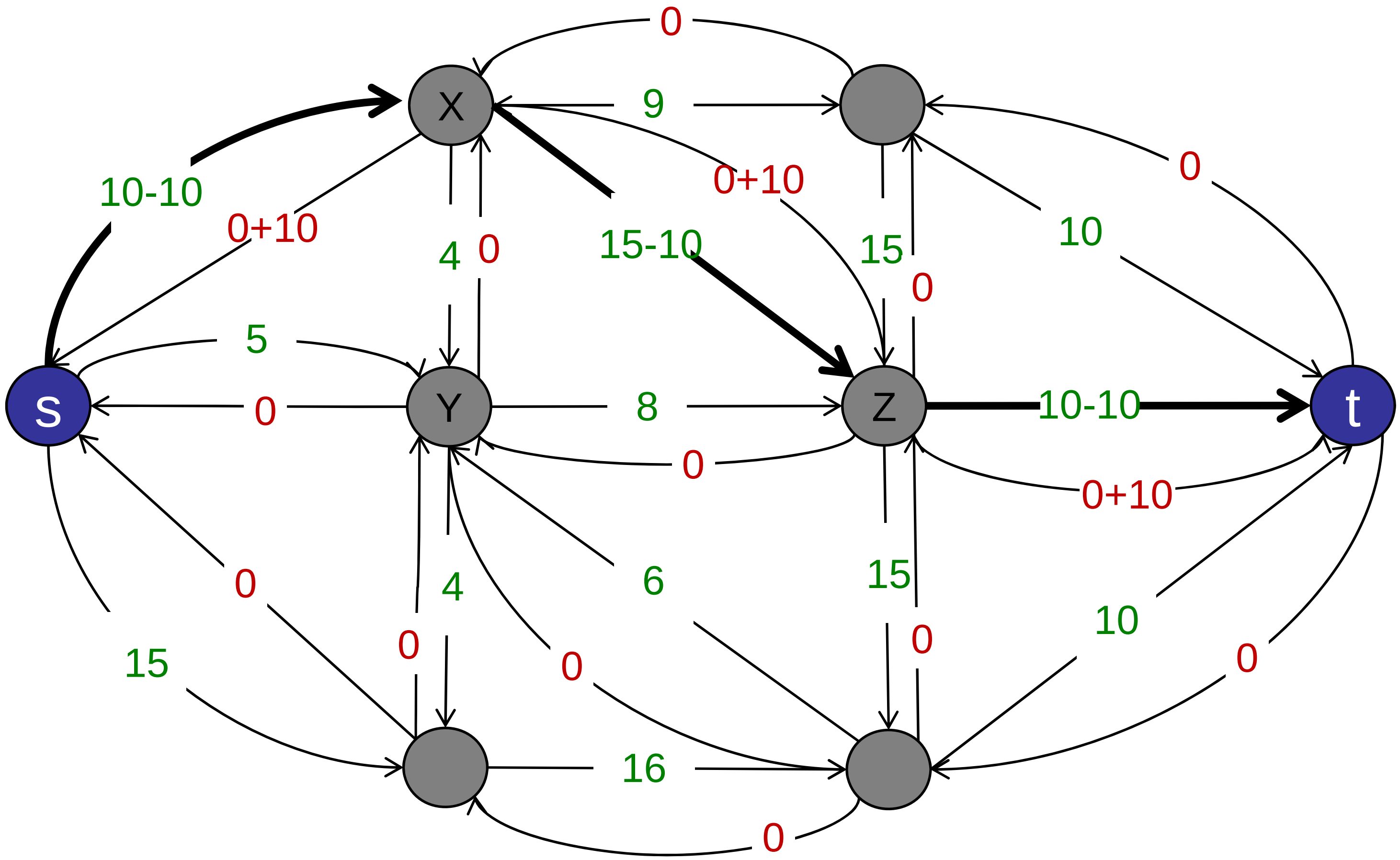


Add edges in both direction.

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)

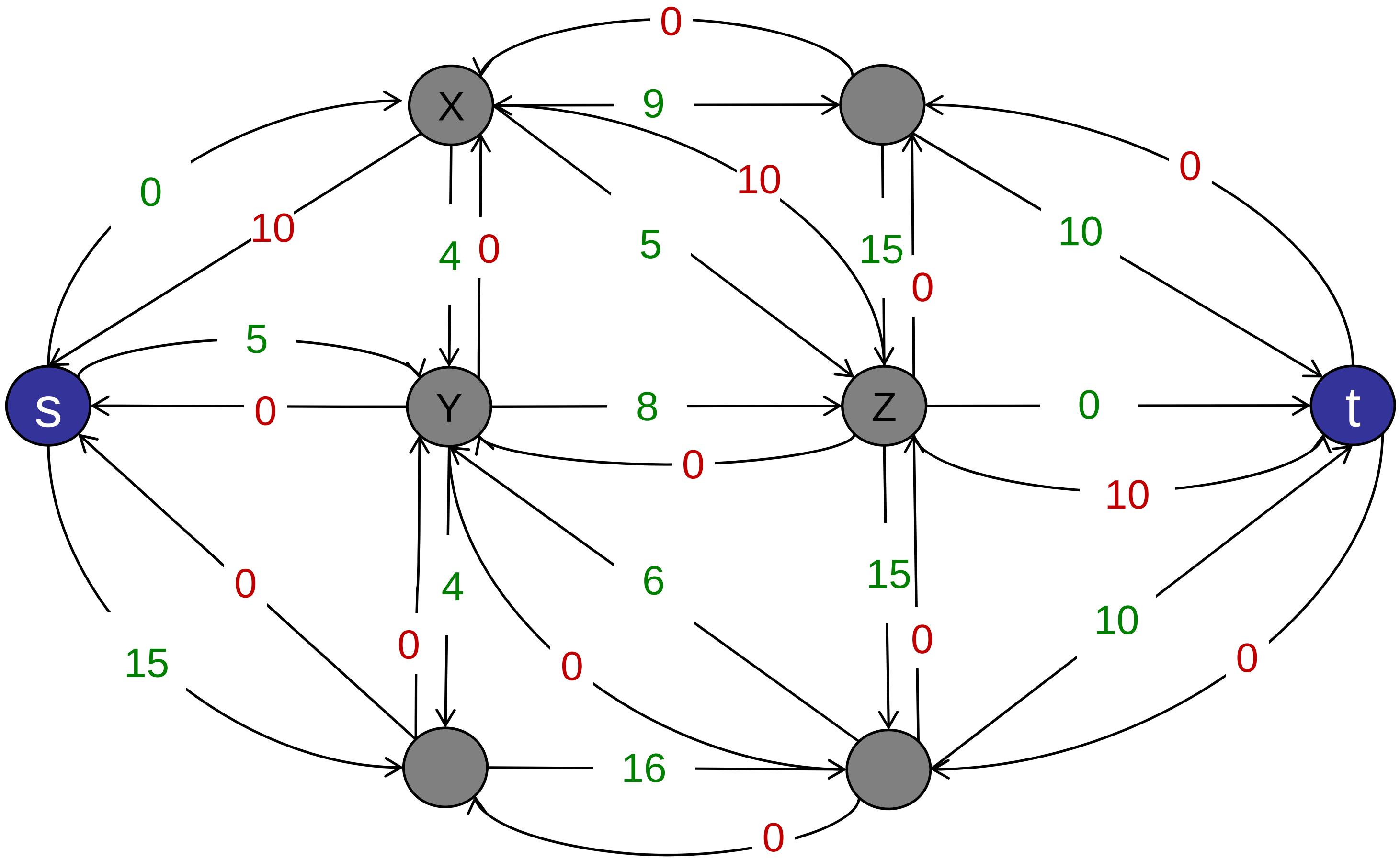


Augmenting path: residual flow in the other direction.

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)

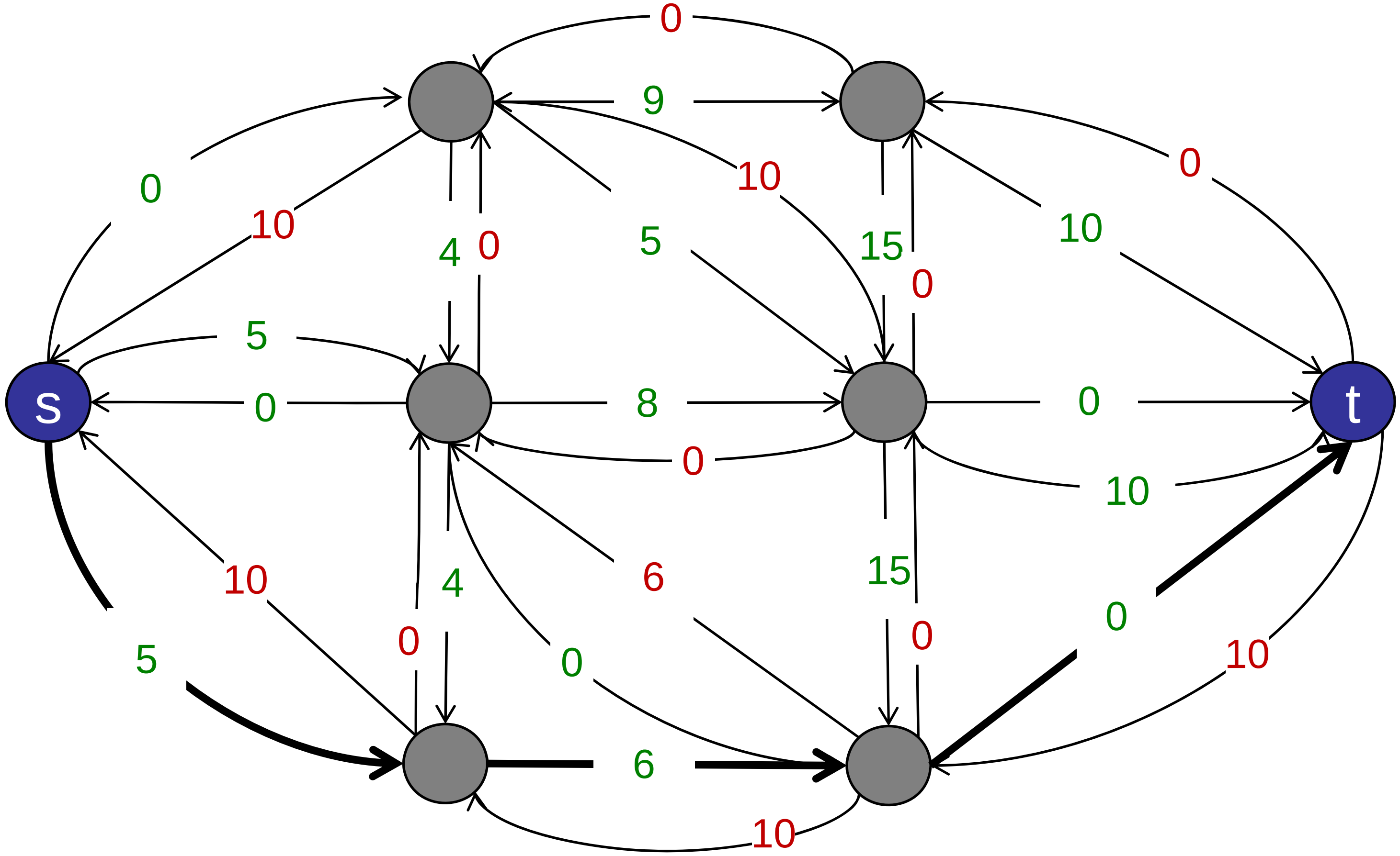


Augmenting path: residual flow in the other direction.

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)

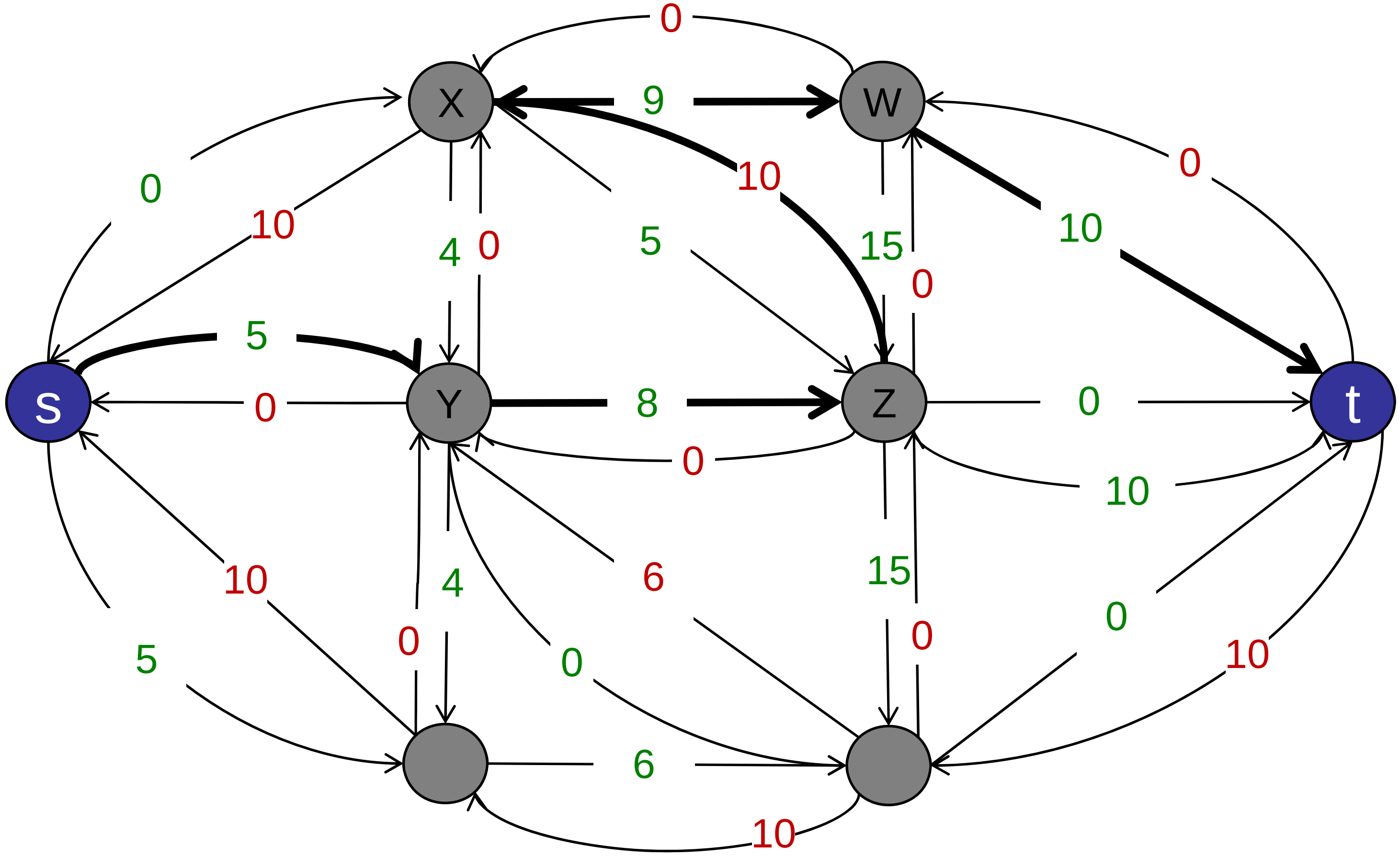


Augmenting path: residual flow in the other direction.

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)

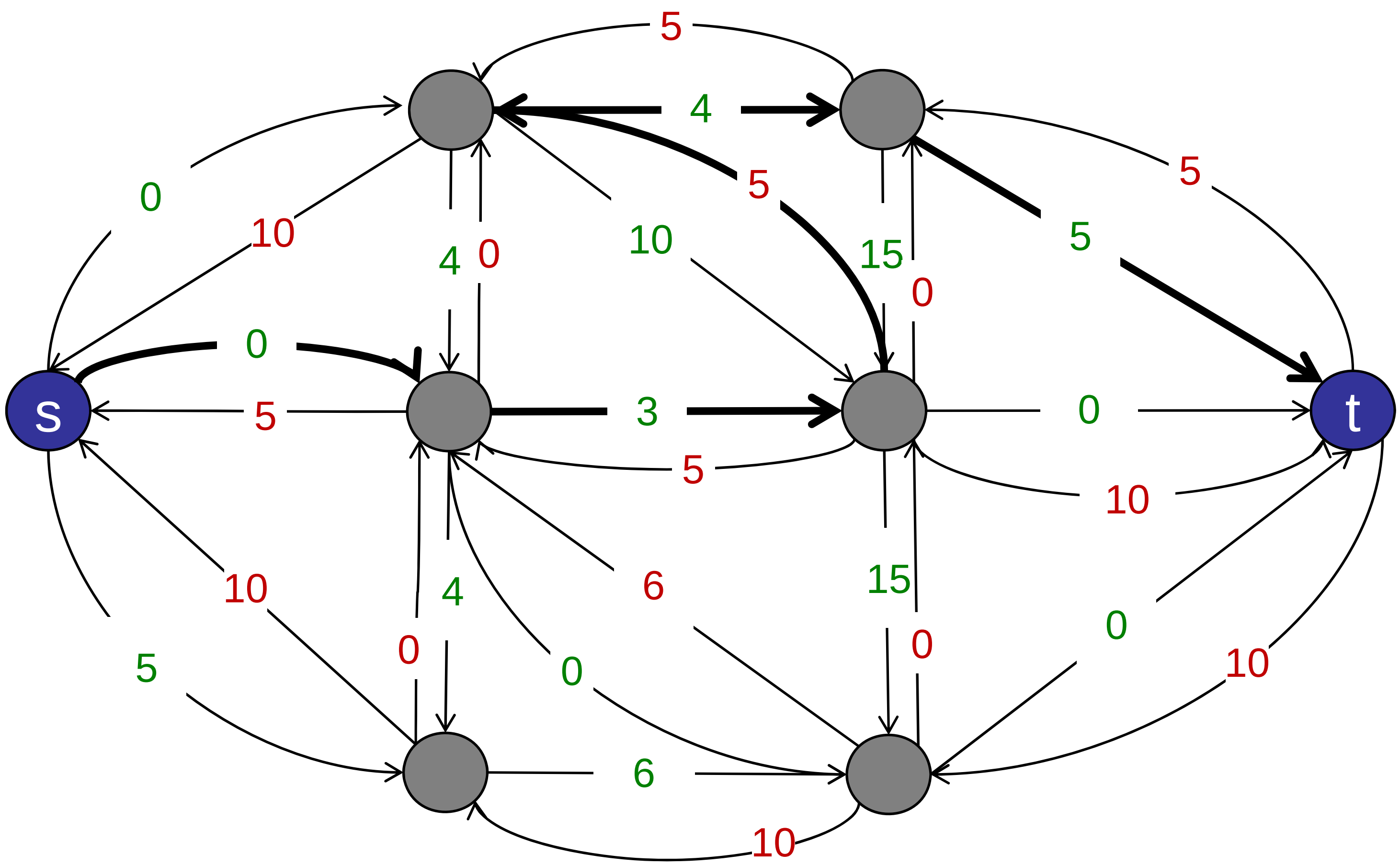


Augmenting path: residual flow in the other direction.

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)

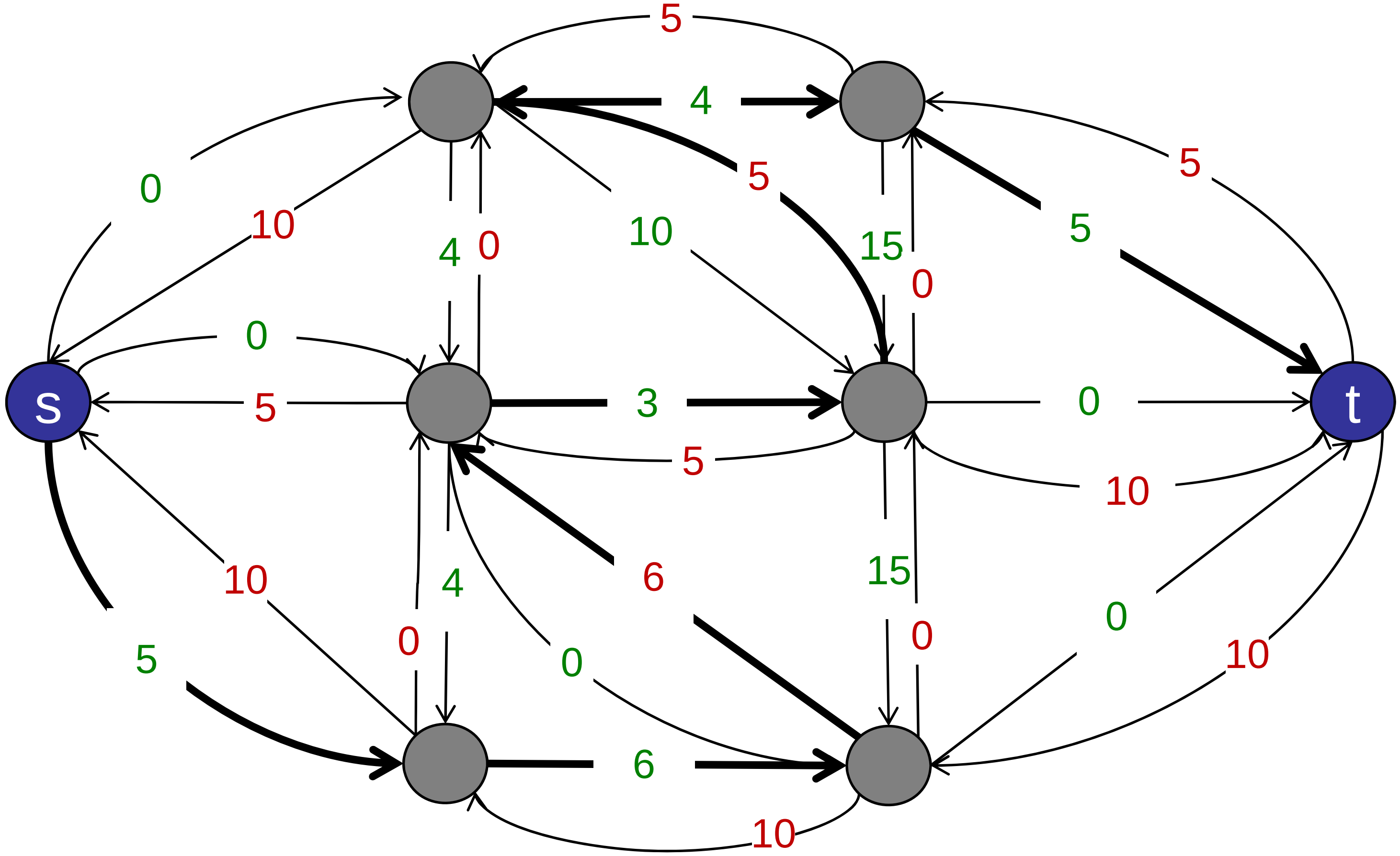


Augmenting path: residual flow in the other direction.

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)

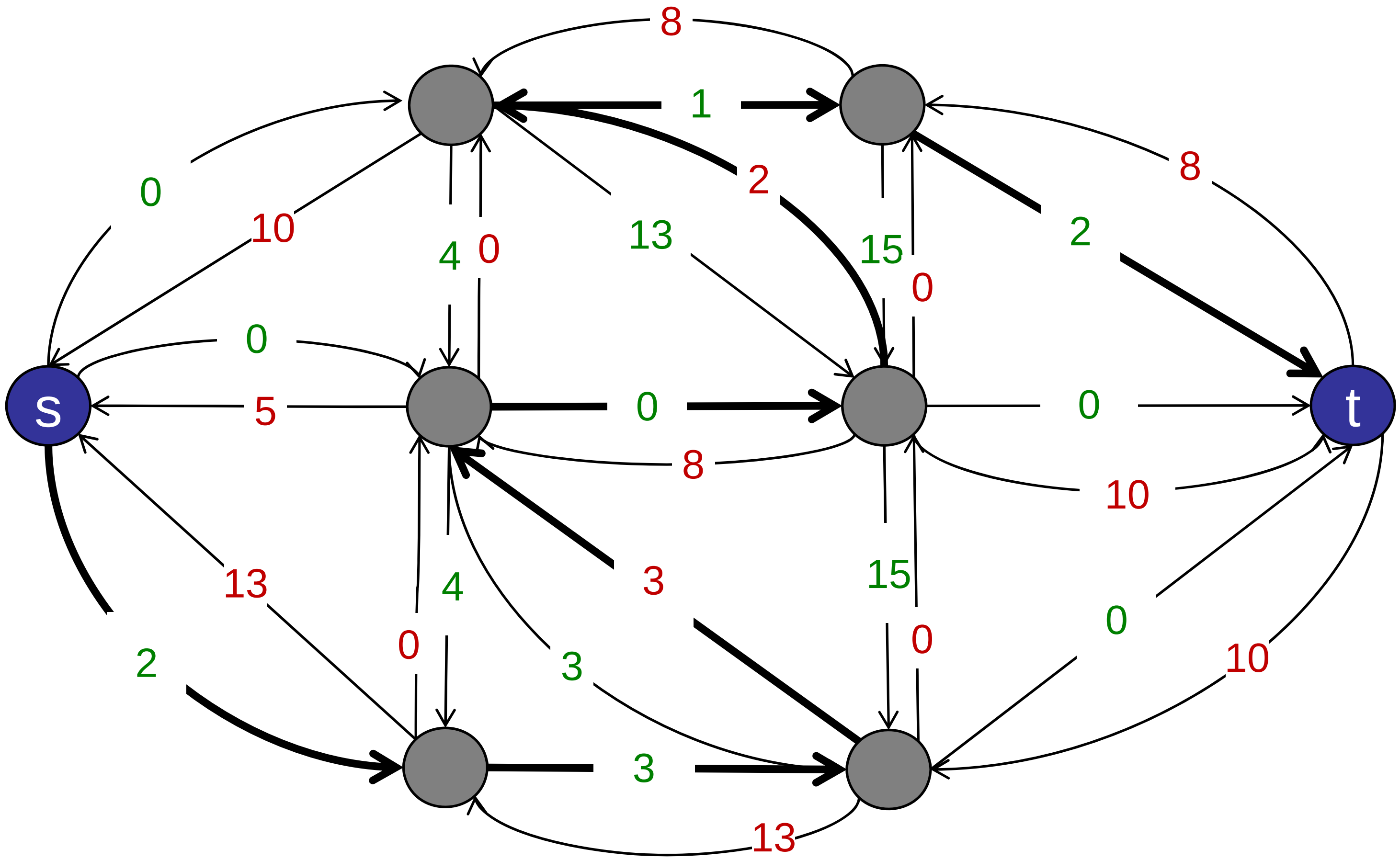


Augmenting path: residual flow in the other direction.

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)

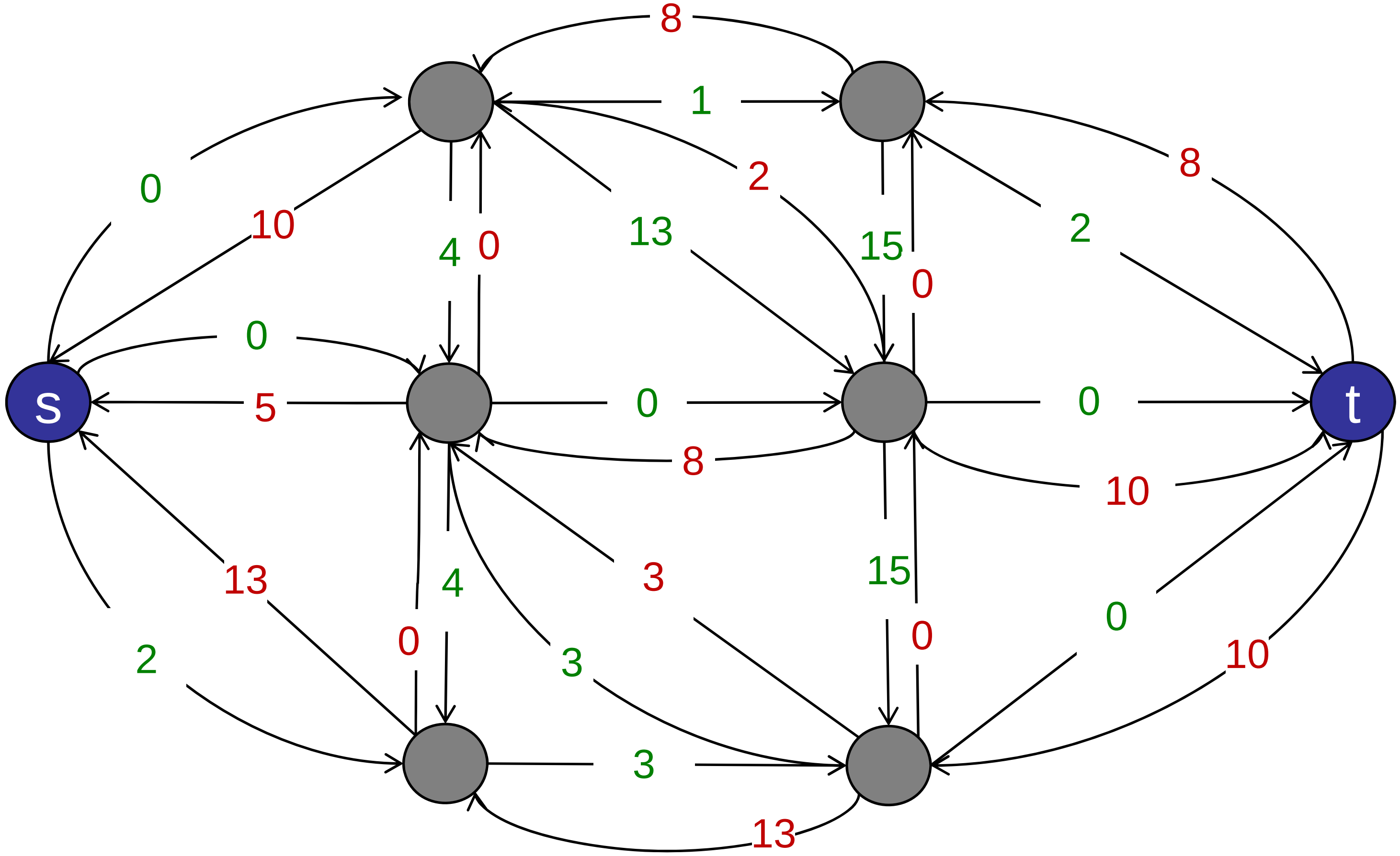


Augmenting path: residual flow in the other direction.

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)

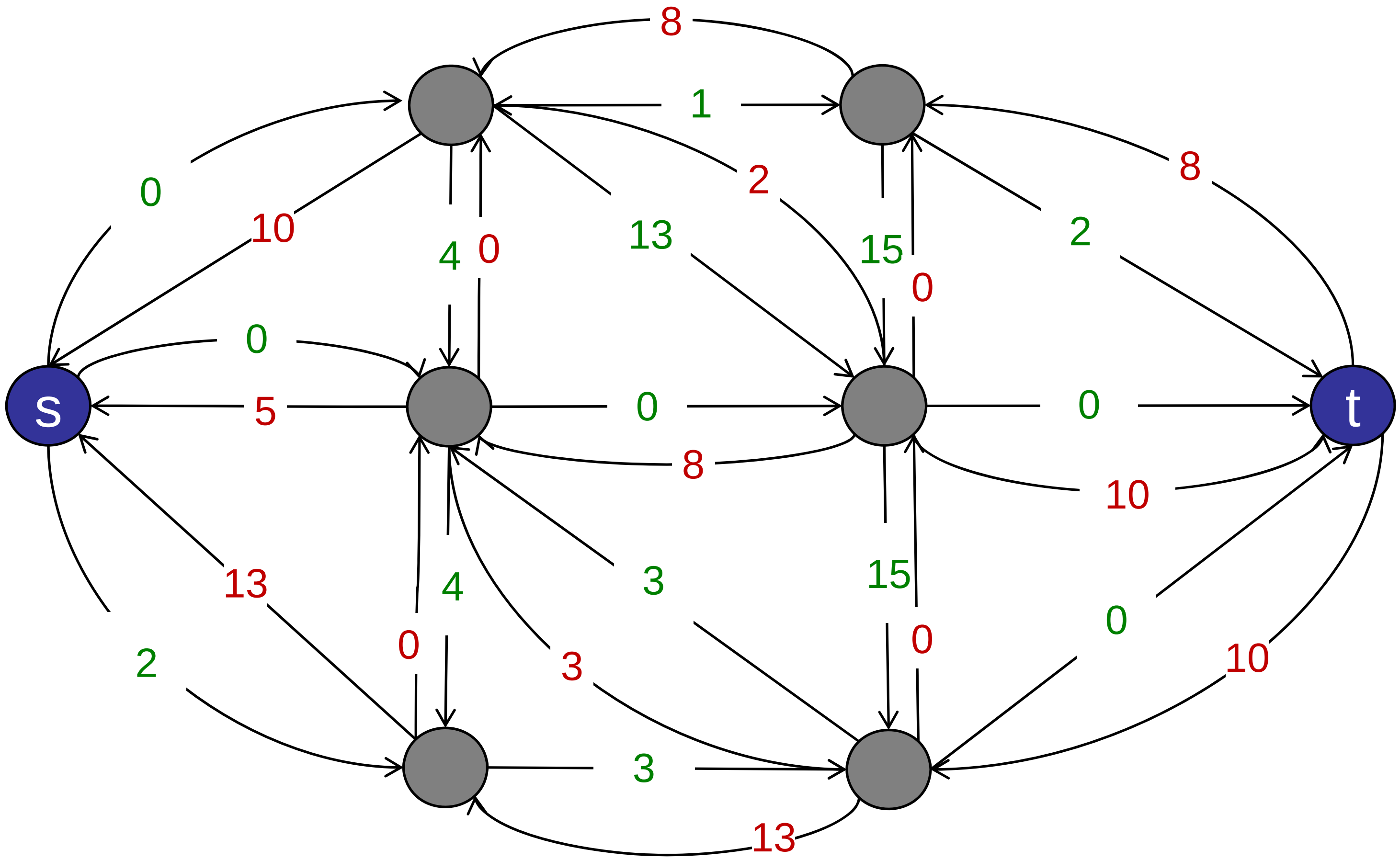


Augmenting path: residual flow in the other direction.

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)



Forward flow = reverse residual flow.

Ford-Fulkerson

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 edge (v,u) with w(v,u) = 0.

While there exists an augmenting path:

- Find an augmenting path via DFS 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.

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?

– Does Ford-Fulkerson always terminate? How fast?

– If it terminates, does it always find a max-flow?

Roadmap

Network Flows

a. Network flows defined

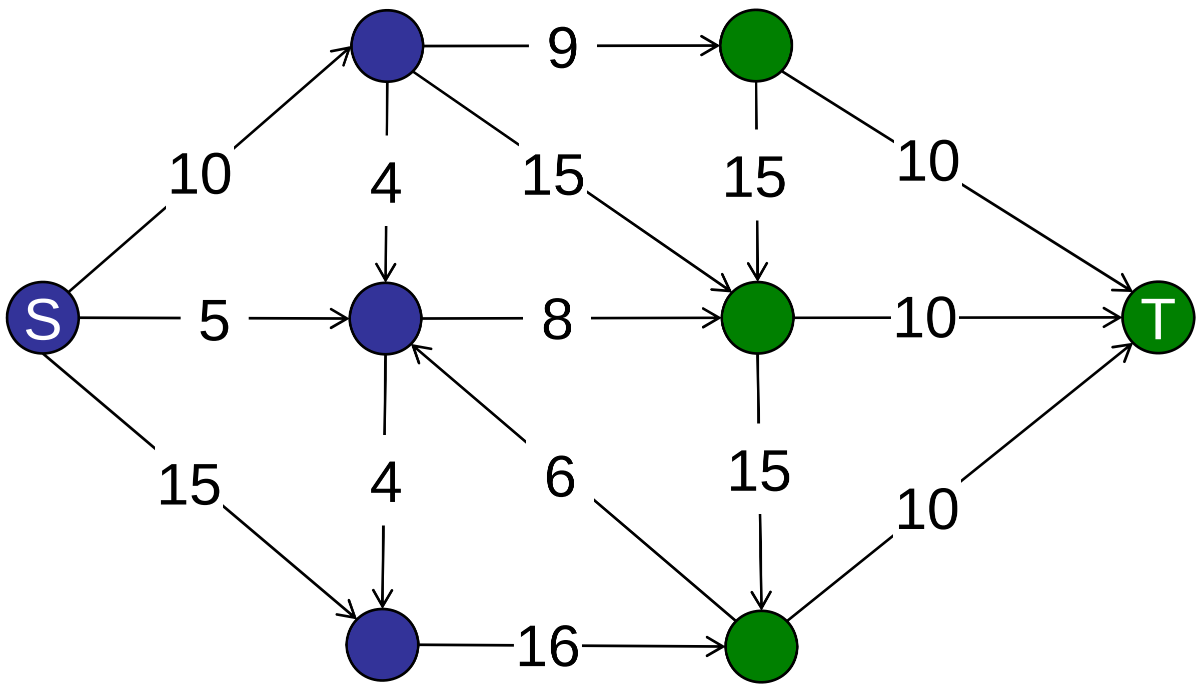
b. Ford-Fulkerson algorithm

c. Max-Flow / Min-Cut Theorem

Cuts and Flows

Definition:

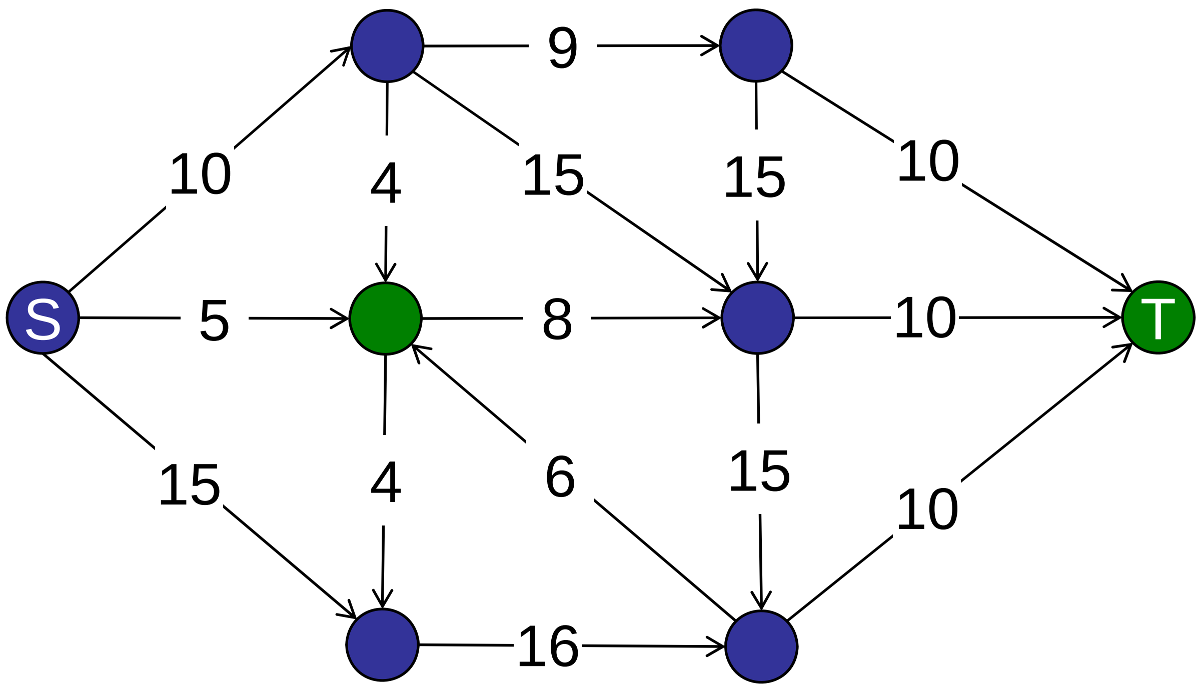
An <u>st-cut</u> partitions the vertices of a graph into two disjoint sets S and T where s $\in$ S and t $\in$ T.



Cuts and Flows

Definition:

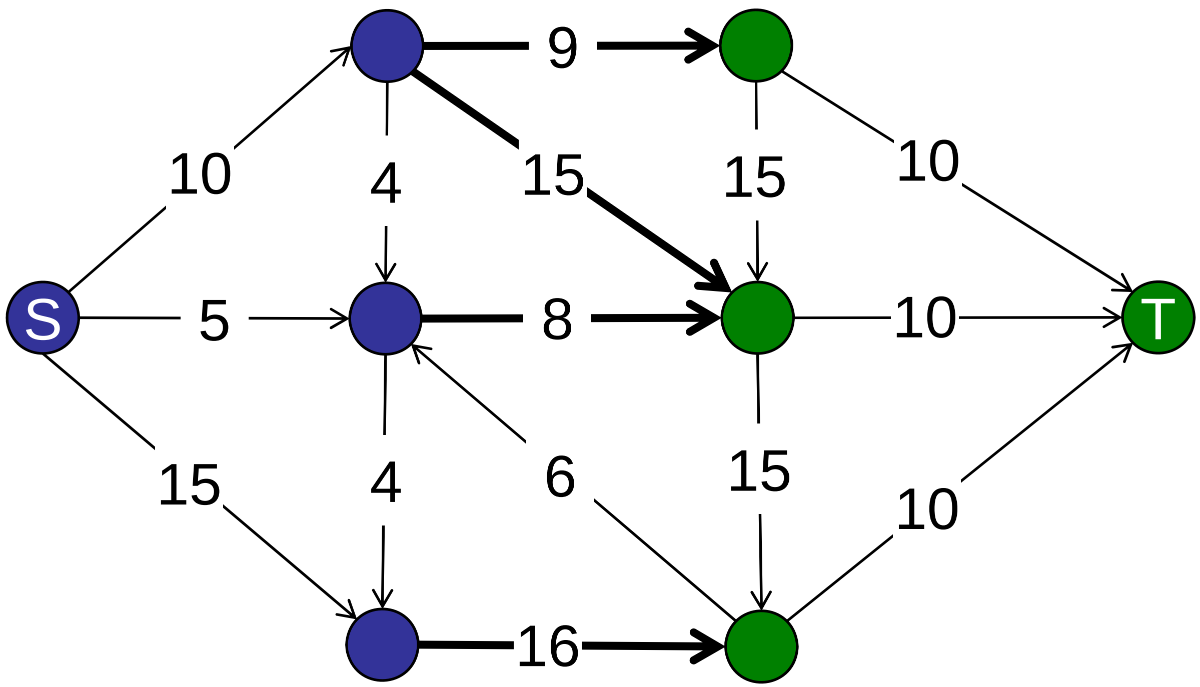
An <u>st-cut</u> partitions the vertices of a graph into two disjoint sets S and T where s ∈ S and t ∈ T.



Cuts and Flows

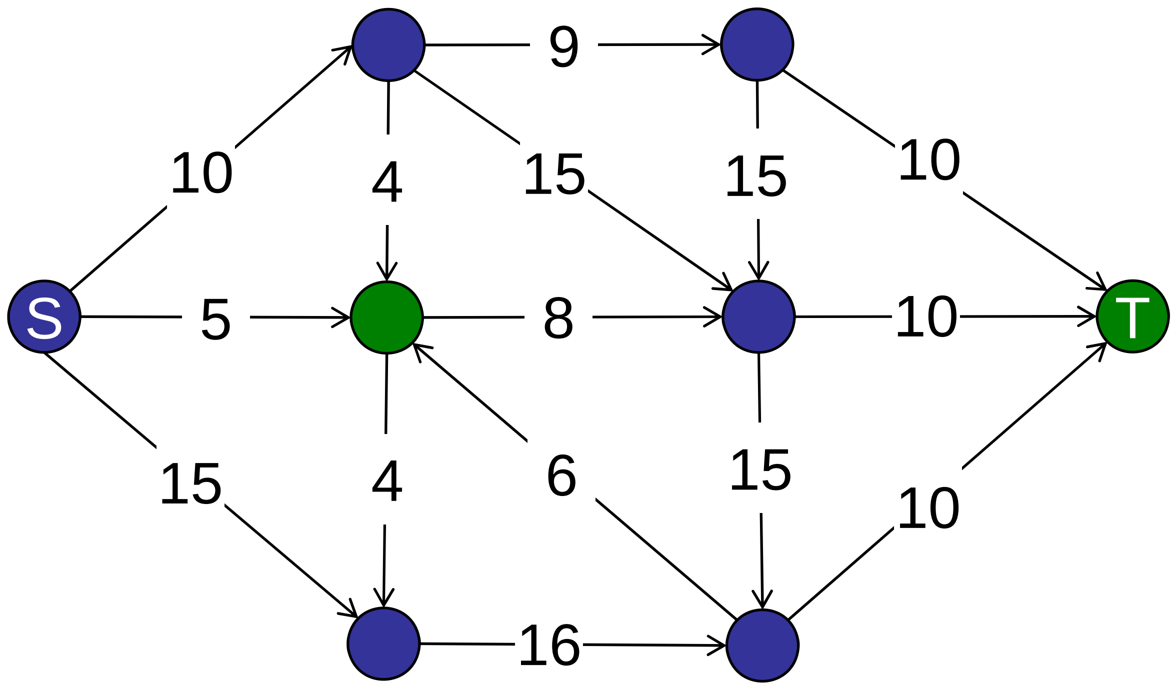
Definition:

The <u>capacity</u> of an st-cut is the sum of the capacities of the edges that cross the cut from S to T.

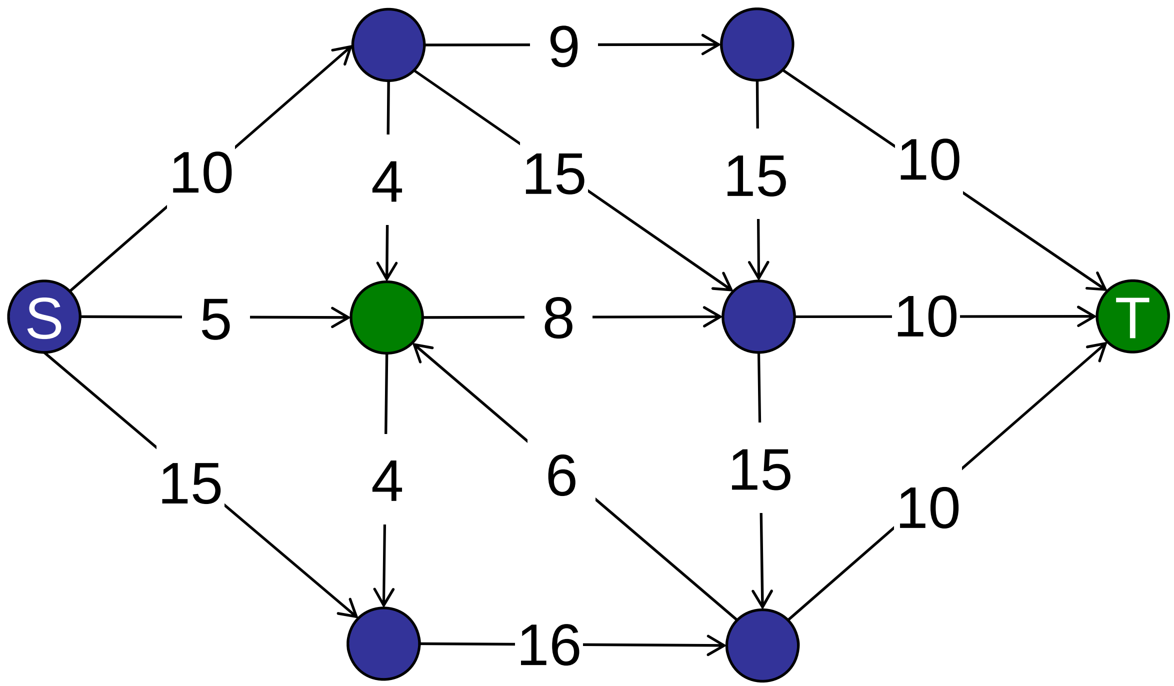


Capacity = 48

What is the capacity of this st-cut?



What is the capacity of this st-cut?

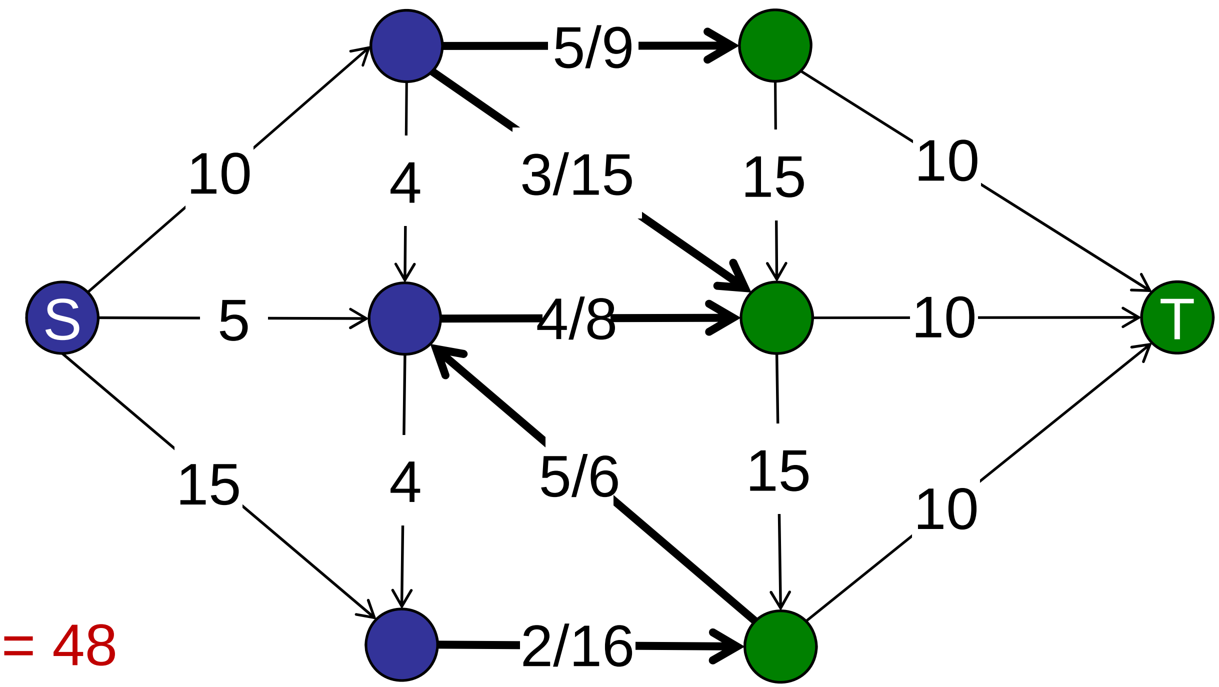


Capacity = 45

Cuts and Flows

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.



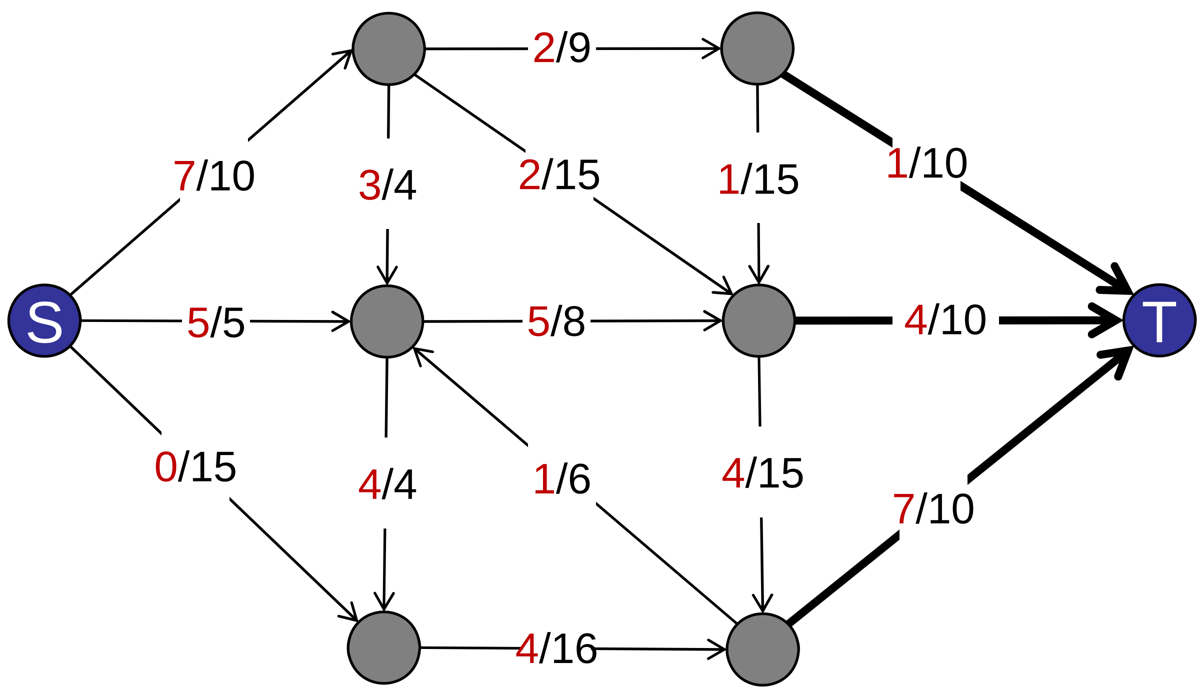
Capacity = 48
Net flow = 9

Cuts and Flows

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.



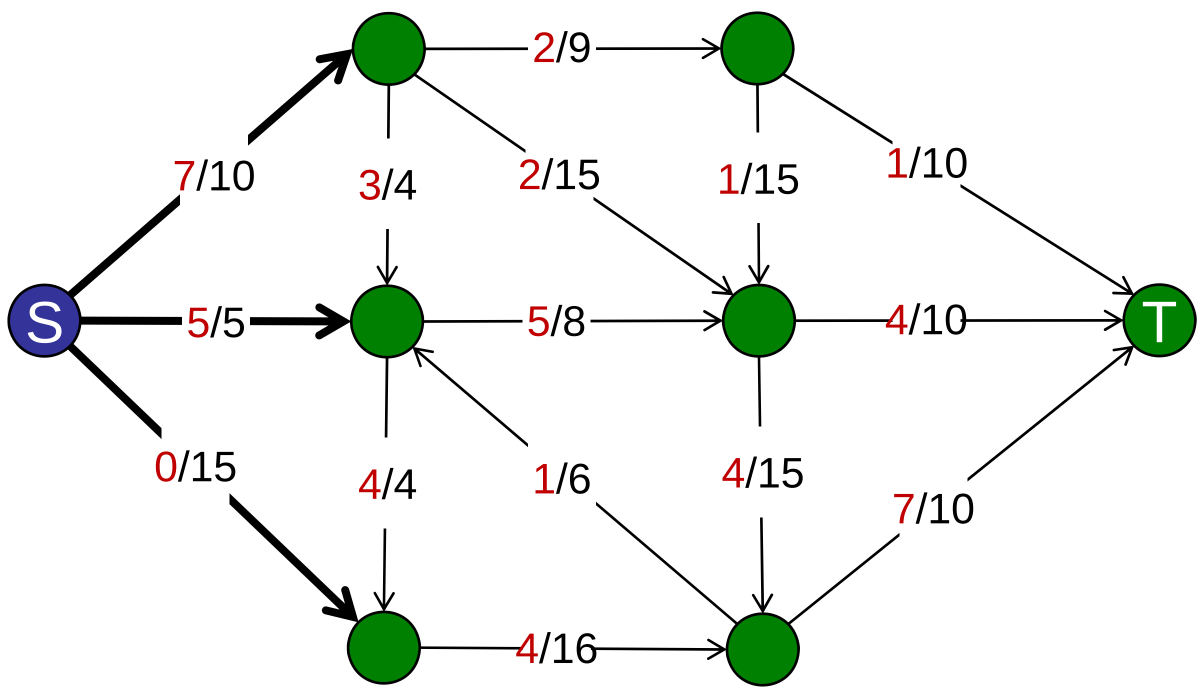
Value of flow = 12

Cuts and Flows

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

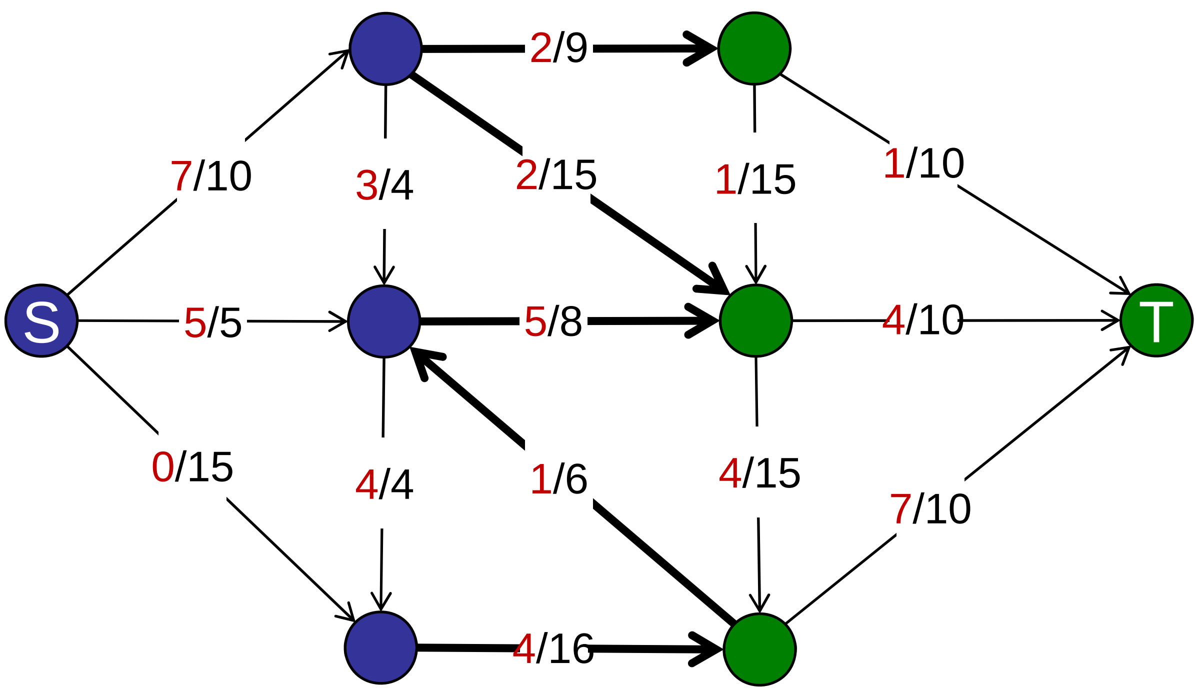
Value of flow = 12

Cuts and Flows

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

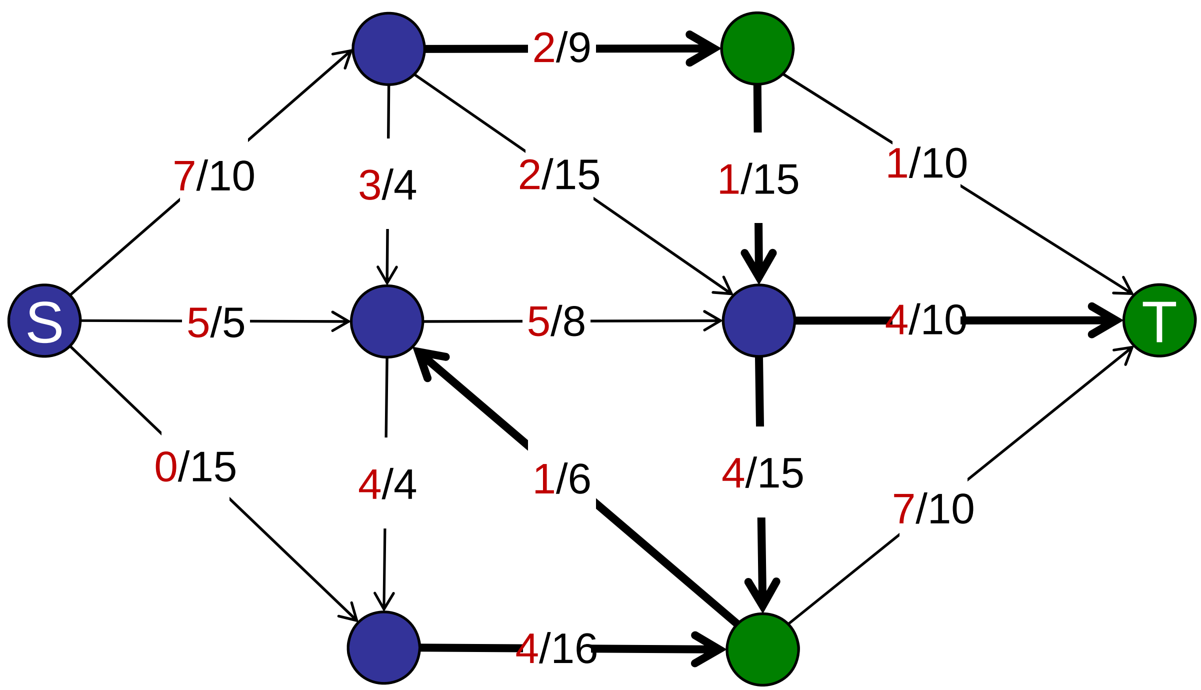
Value of flow = 12

Cuts and Flows

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

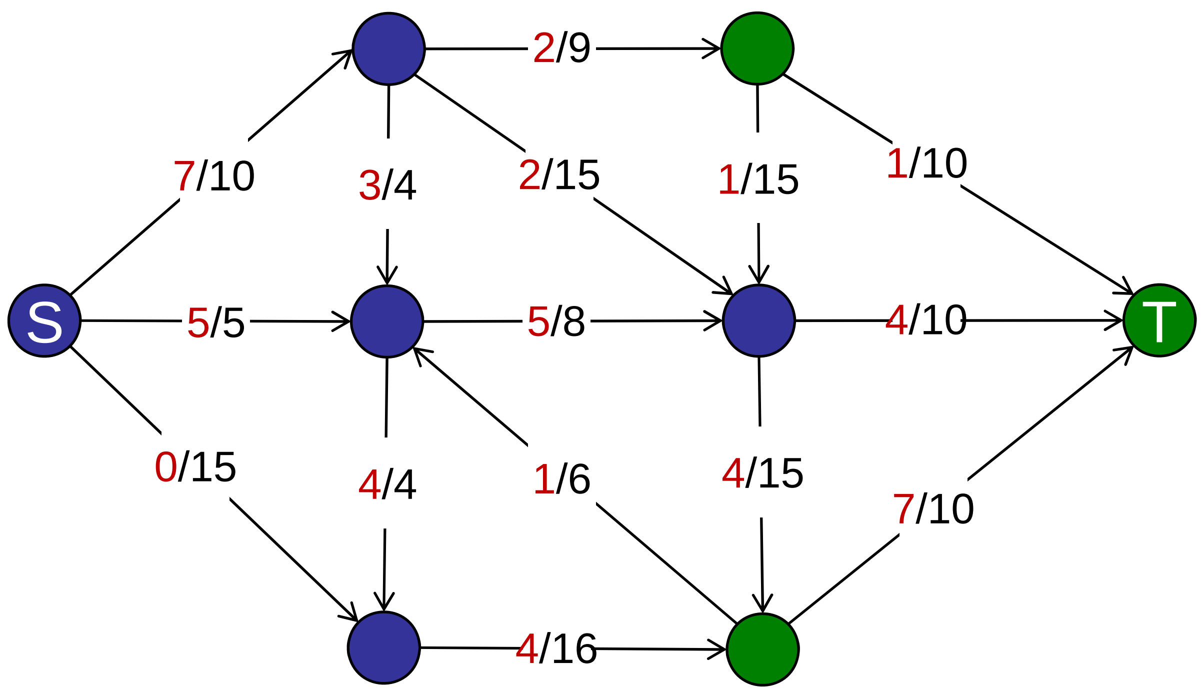
Value of flow = 12

Cuts and Flows

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.



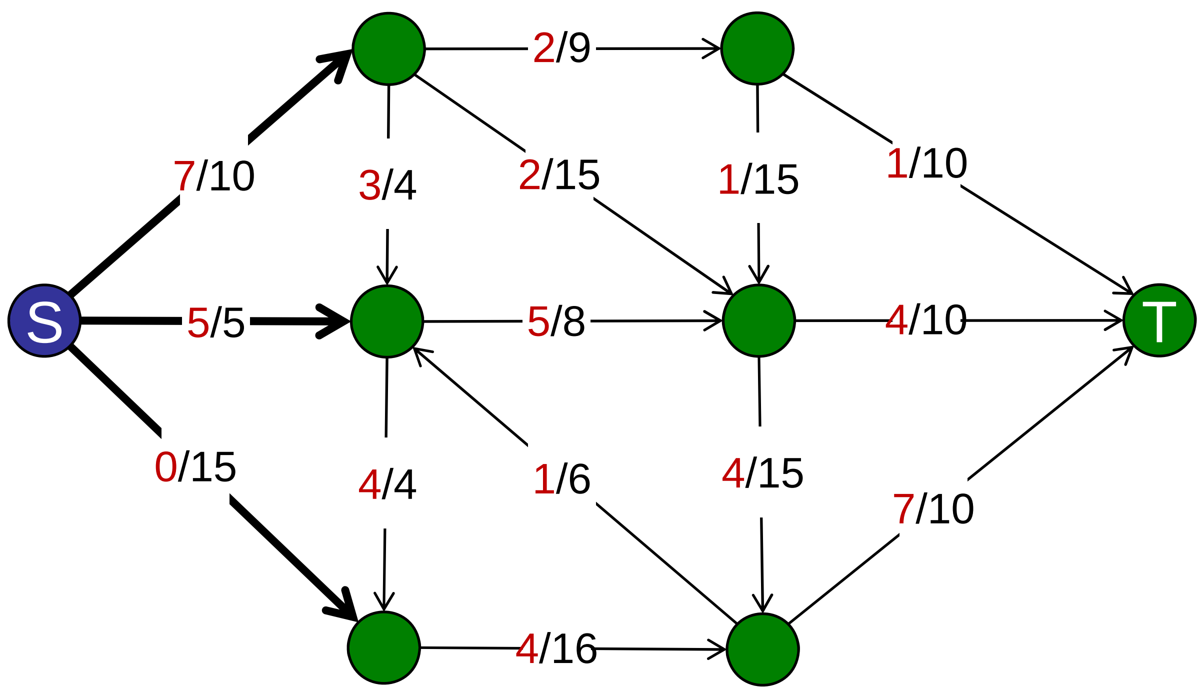
Flow across cut = 12

Value of flow = 12

Cuts and Flows

Proof: (by induction)

- Start with S = {s}, T = V \ S.
- Define F = flow across cut.

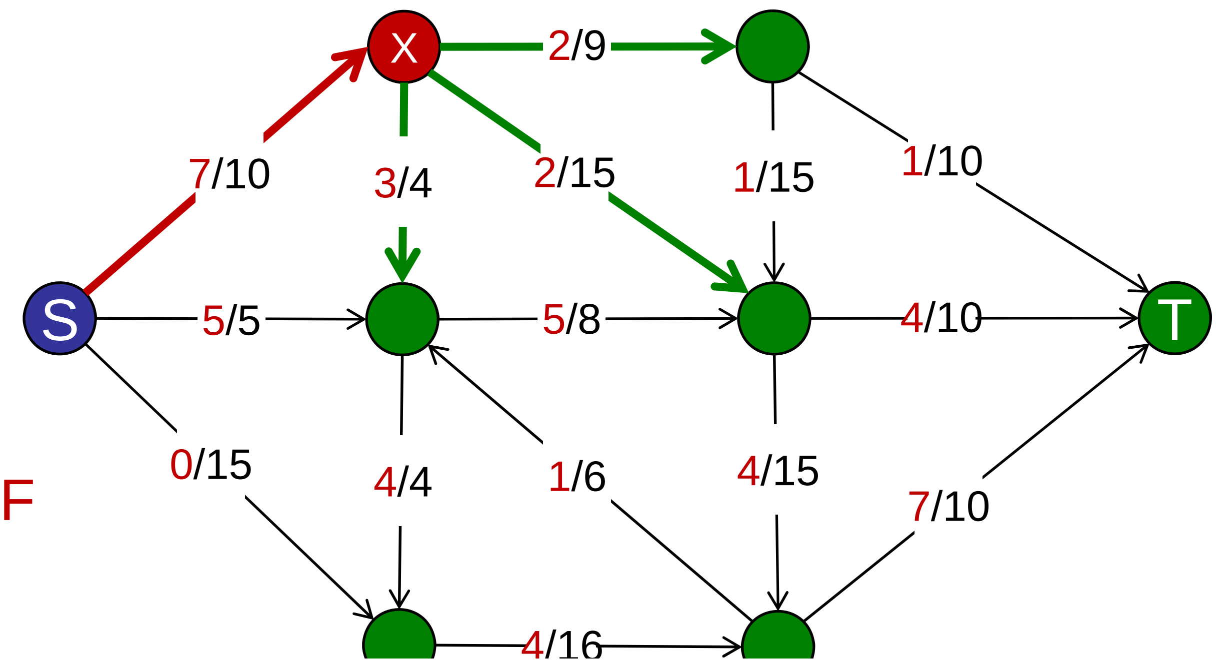


Cuts and Flows

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.



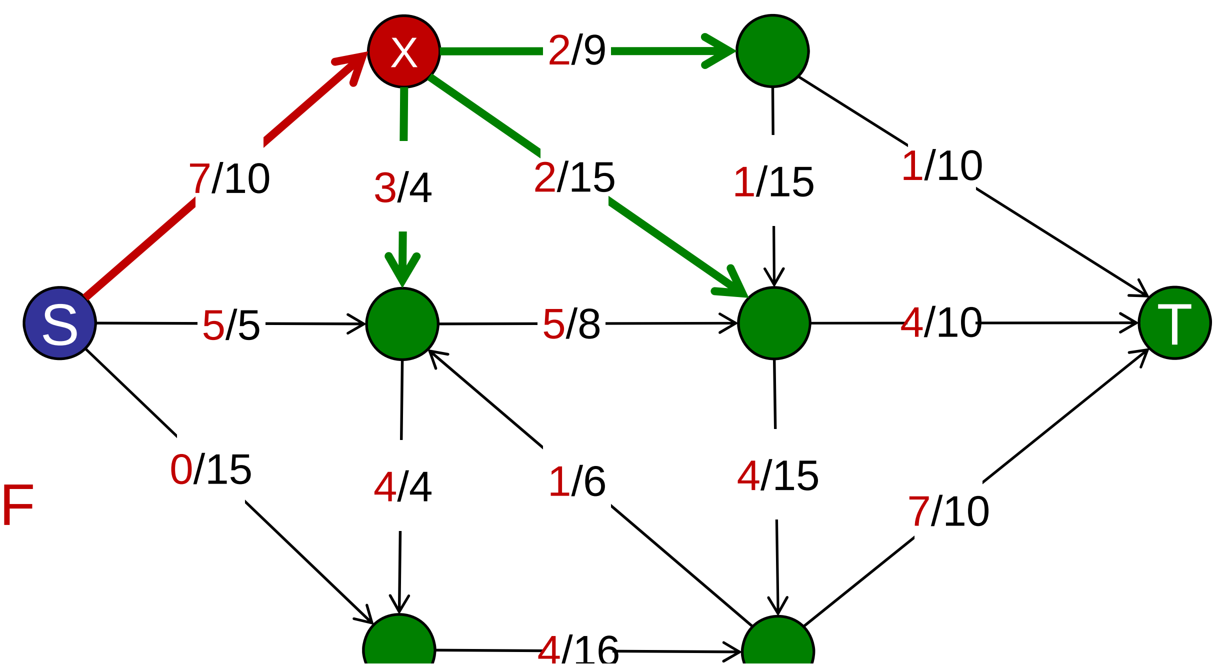
F – 7 + (3 + 2 + 2) = F

Cuts and Flows

Inductive step:

Conservation of flow: (equilibrium constraint)

- Flow into X equals flow out of X.
- Flow that crossed (old S)→X == X→(old T)
- F remains unchanged

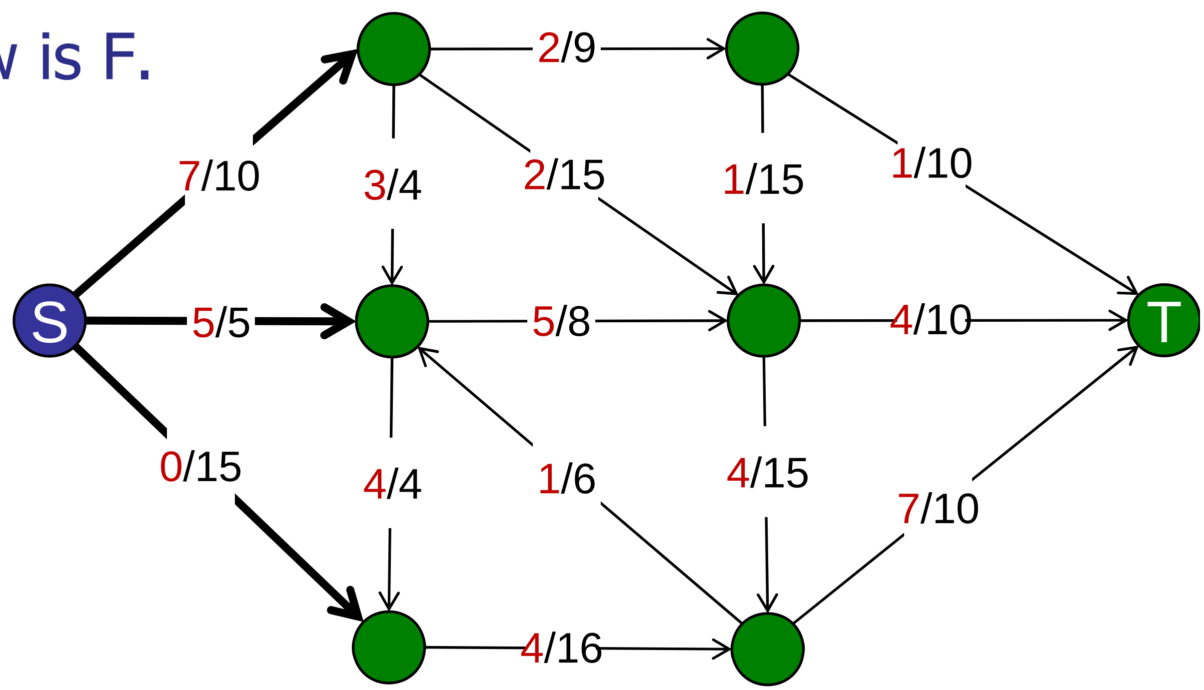


F – 7 + (3 + 2 + 2) = F

Cuts and Flows

Proof: (by induction)

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

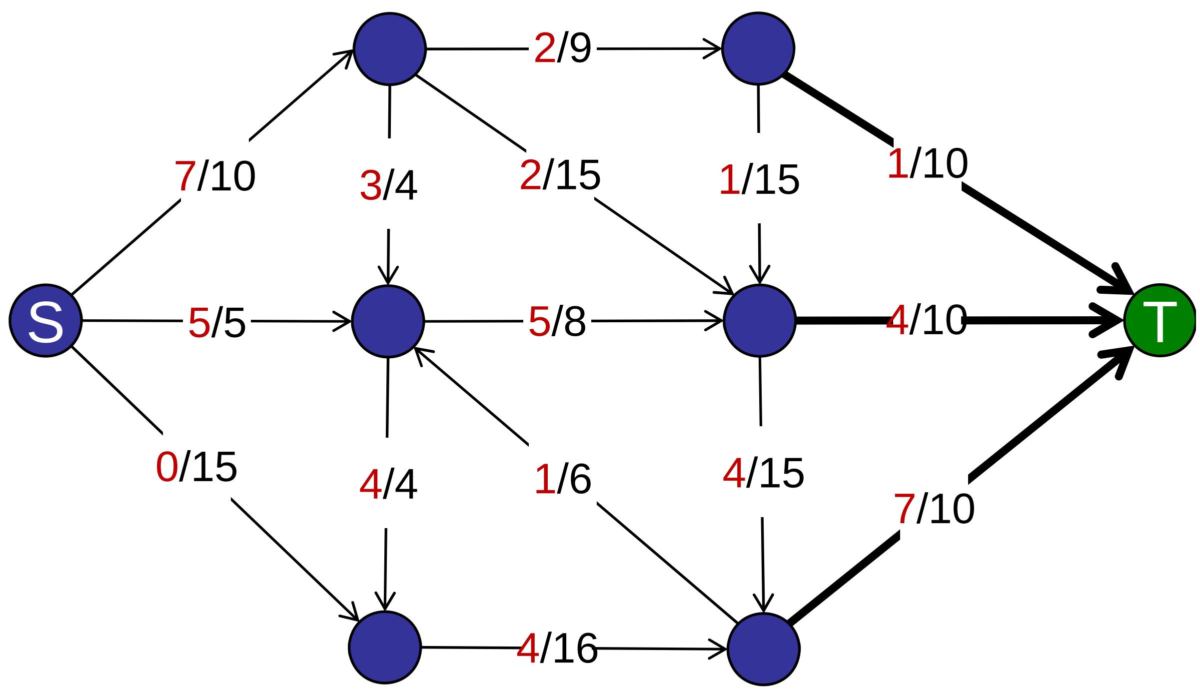


Cuts and Flows

Proof: (by induction)

What is F?

- Consider cut S = V \ {t}, T = {t}.
- All edges crossing cut go to t.
- Value of flow = flow across cut = F.

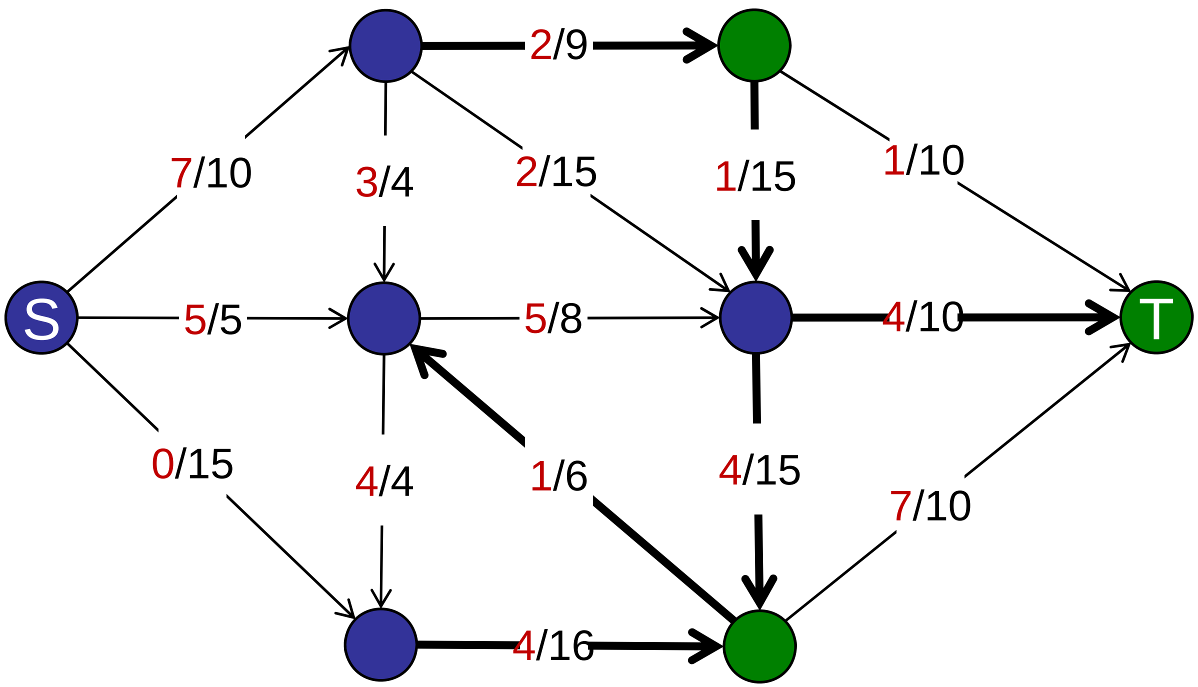


Cuts and Flows

Proposition (Flow Value):

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

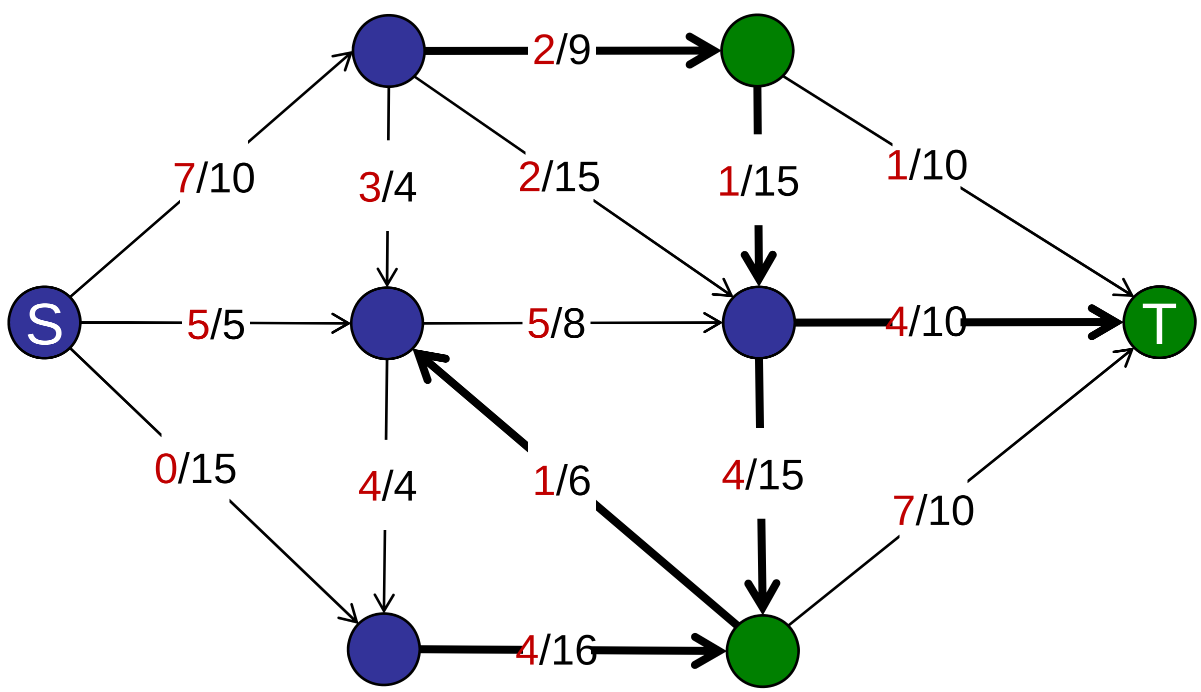
Value of flow = 12

Cuts and Flows

Weak duality:

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

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



Cuts and Flows

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

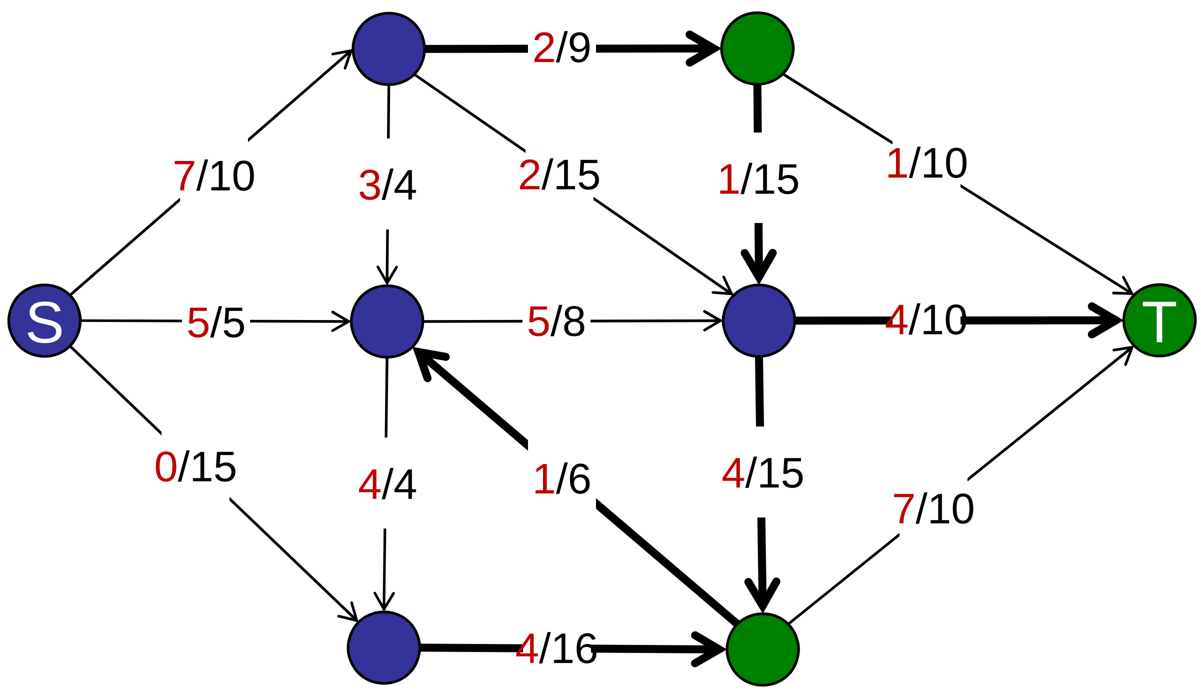
Cuts and Flows

MaxFlow-MinCut Theorem:

Let f be a maximum flow.

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

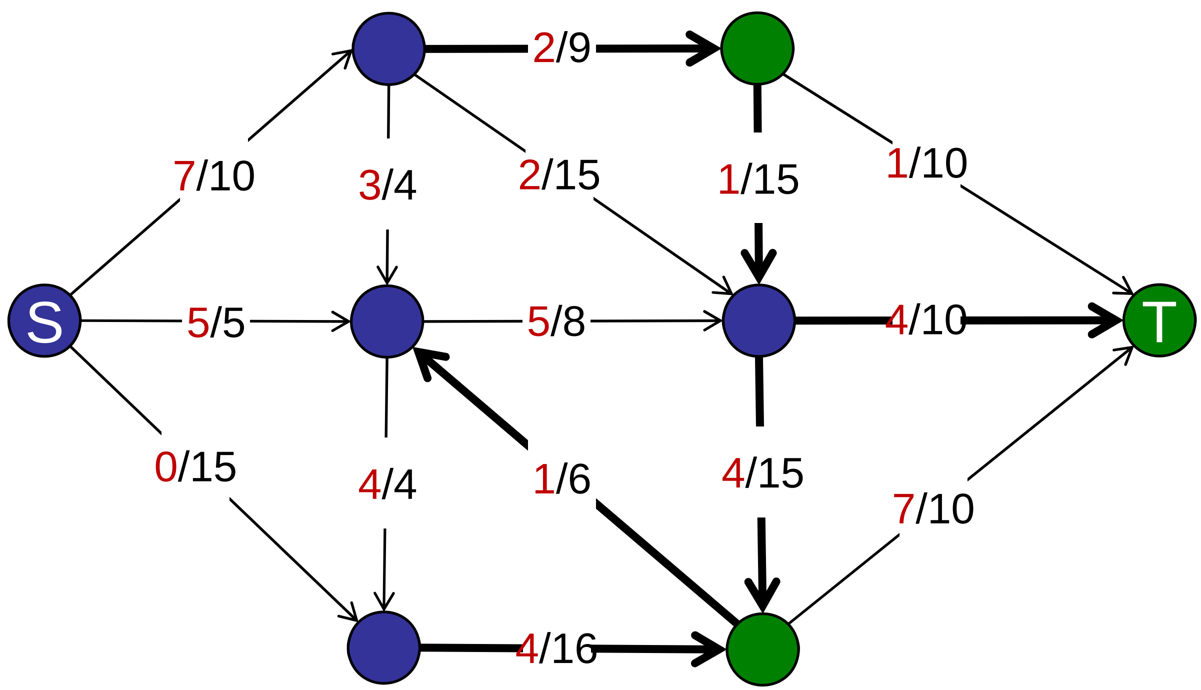
Then value(f) = capacity(S,T).



Cuts and Flows

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

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

1 → 2: There exists an f-capacity cut ➔ f is maximum

Assume (S,T) is a cut with 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

weak duality

Cuts and Flows

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

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

Assume there IS at least 1 augmenting path:

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

Cuts and Flows

Proof:

The following three statements are equivalent for flow f:

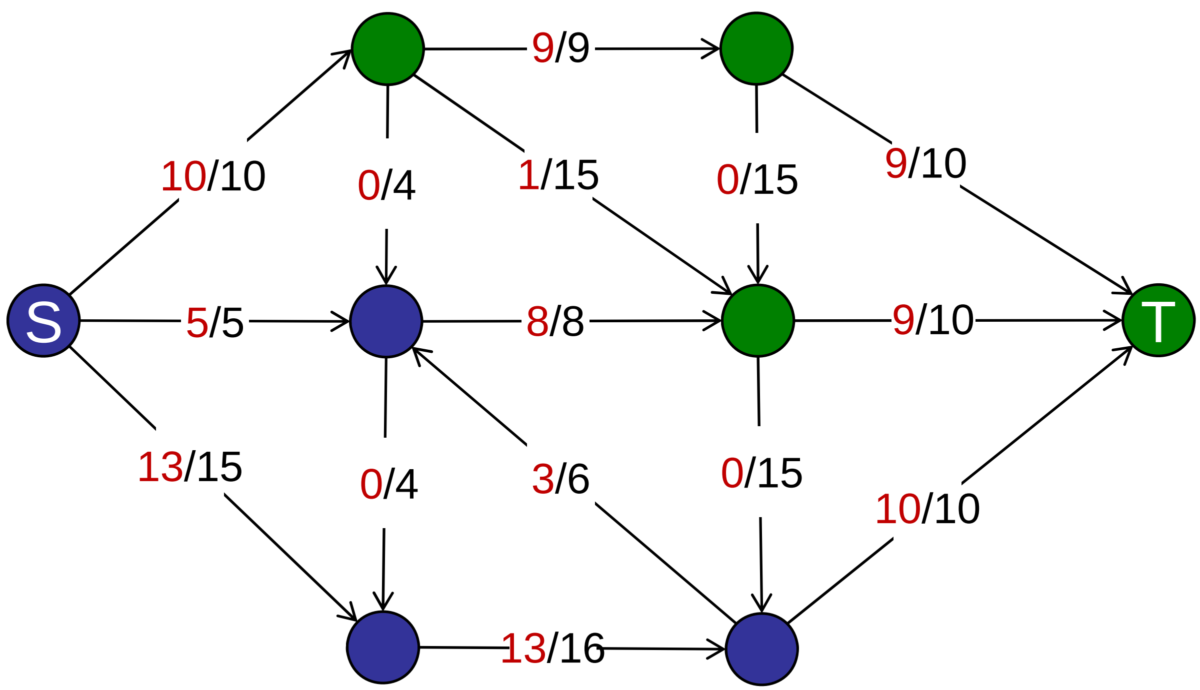
1. There exists a cut whose capacity equals the value of f.
2. f is a maximum flow
3. There is no augmenting path with respect to f.

Cuts and Flows

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

Assume there is no augmenting path:

- Let S be the nodes reachable from the source in the residual graph.
- Let T be the remaining nodes.



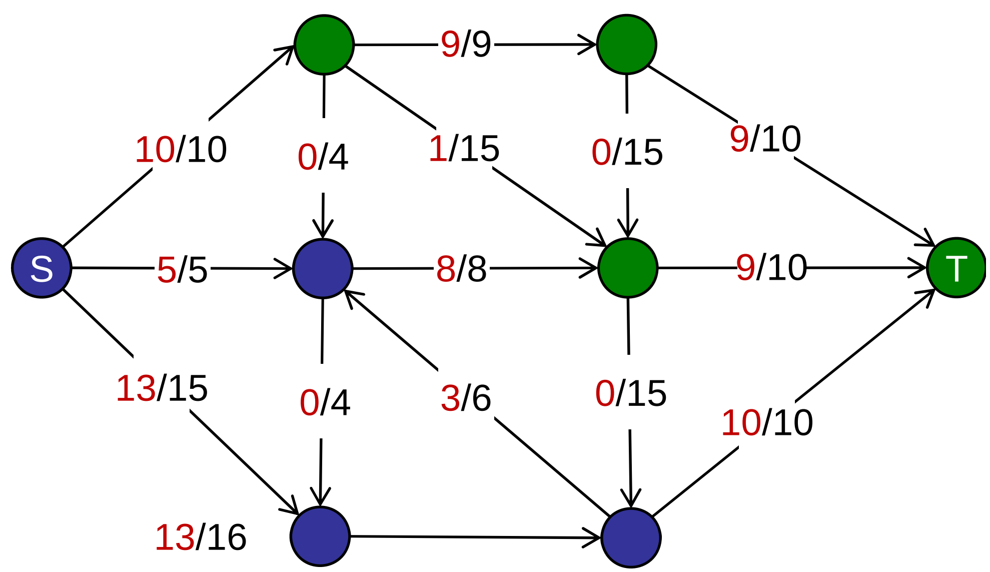
Cuts and Flows

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

Assume there is no augmenting path:

- Let S be the nodes reachable from the source in the residual graph.
- Let T be the remaining nodes.
- S contains the source s.
- T contains the target t.

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



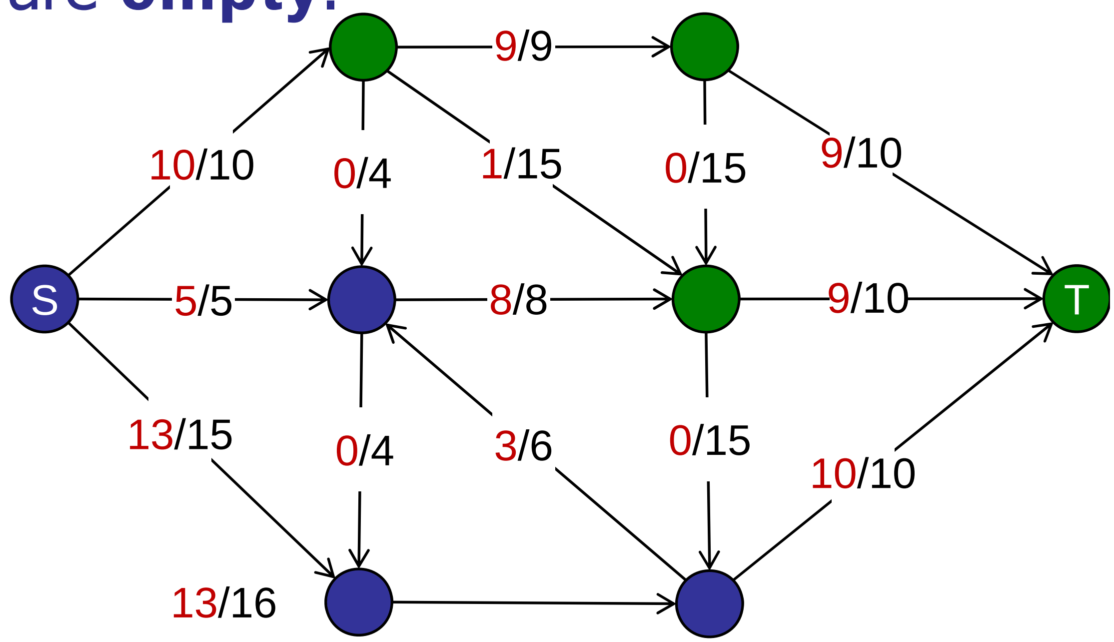
Cuts and Flows

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

Assume there is no augmenting path:

- Let S be the nodes reachable from the source in the residual graph. T = remaining nodes.
- (S,T) is an st-cut.
- All edges from T→S are **empty**.

Otherwise, there would be an outgoing edge in the residual graph.

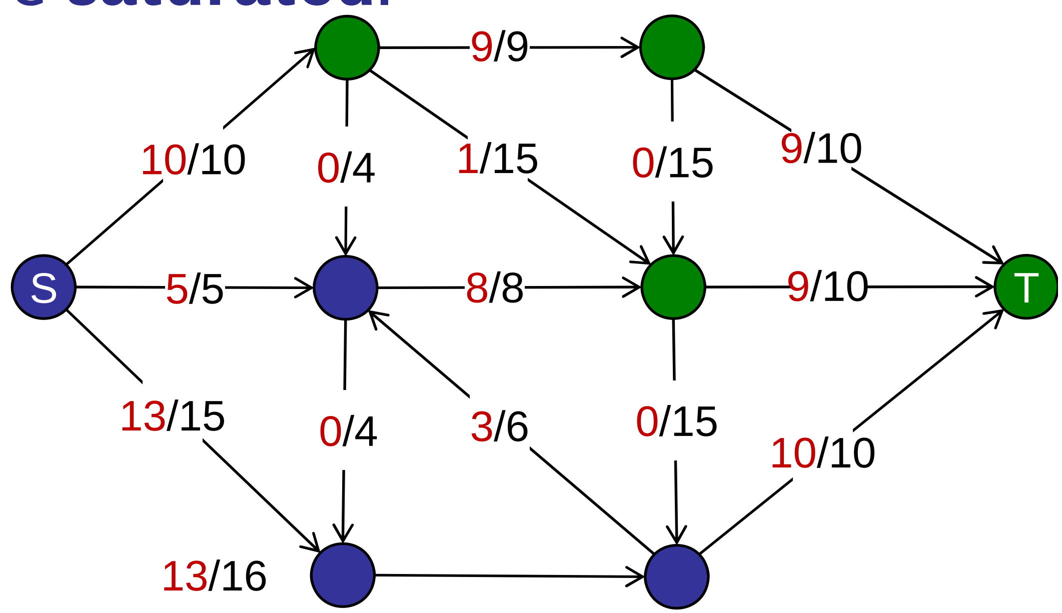


Cuts and Flows

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

Assume there is no augmenting path:

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



Cuts and Flows

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

Assume there is no augmenting path:

- Let S be the nodes reachable from the source in the residual graph. T = remaining nodes.(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

S→T saturation
T→S empty

Cuts and Flows

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

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. Thus, the resulting flow is maximum.

Cuts and Flows

How to find a min-cut:

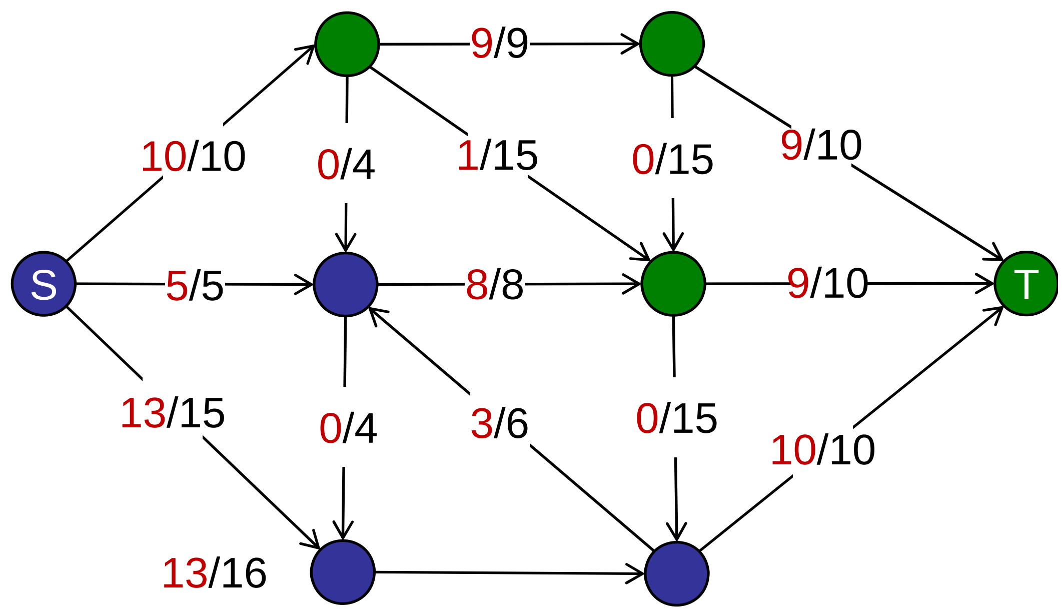
1. Run Ford-Fulkerson until termination.
2. Let S be the set of nodes reachable from the source s:
   - Run DFS in the residual graph.
   - All the nodes reach are in S.
3. For every edge in S, enumerate outgoing edges:
   - If edge exits S, add to min-cut.
   - If both ends of edge are in S, then continue.

Cuts and Flows

Finding a minimum cut:

Assume there is no augmenting path:

- Let S be the nodes reachable from the source in the residual graph.
- T = remaining nodes
- Edges from (S→T) are minimum cut.



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.

Termination: Ford-Fulkserson always terminates.

- ✓ 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

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 terminates if capacities are integers.

- 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

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-Fulkserson 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?