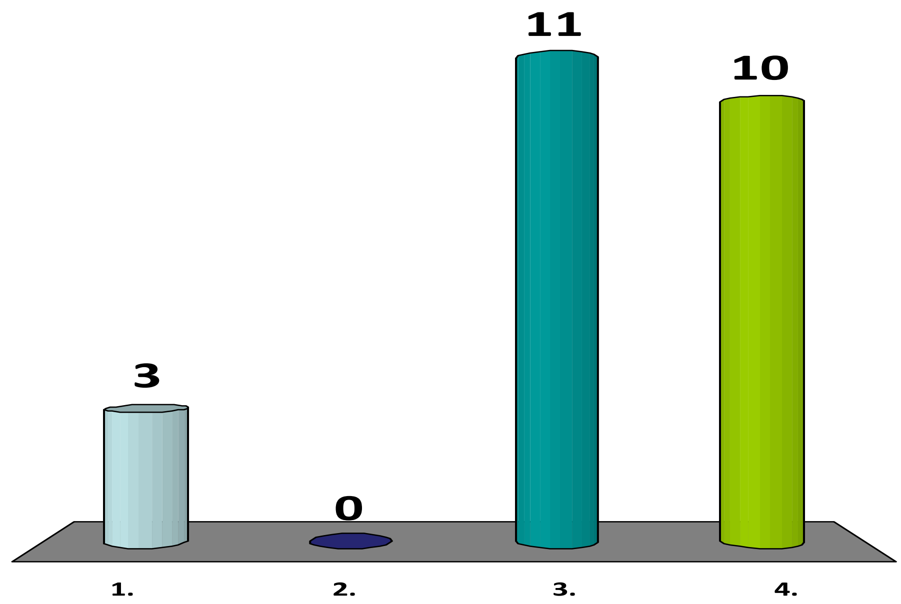
CS2020
Data Structures and Algorithms

Network Flows

On Friday, I'm going to:

1. Bring my clicker to class.
2. Leave my clicker at home.
3. Oops, I already lost my clicker.
4. What is a clicker?



Clickers

Friday

- Return clickers at the end of class!
- Don't forget!
- Missing / unreturned clickers cost $105!!

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. Sample problems
c. Ford-Fulkerson algorithm
d. Max-Flow / Min-Cut Theorem

Network Flow Problems

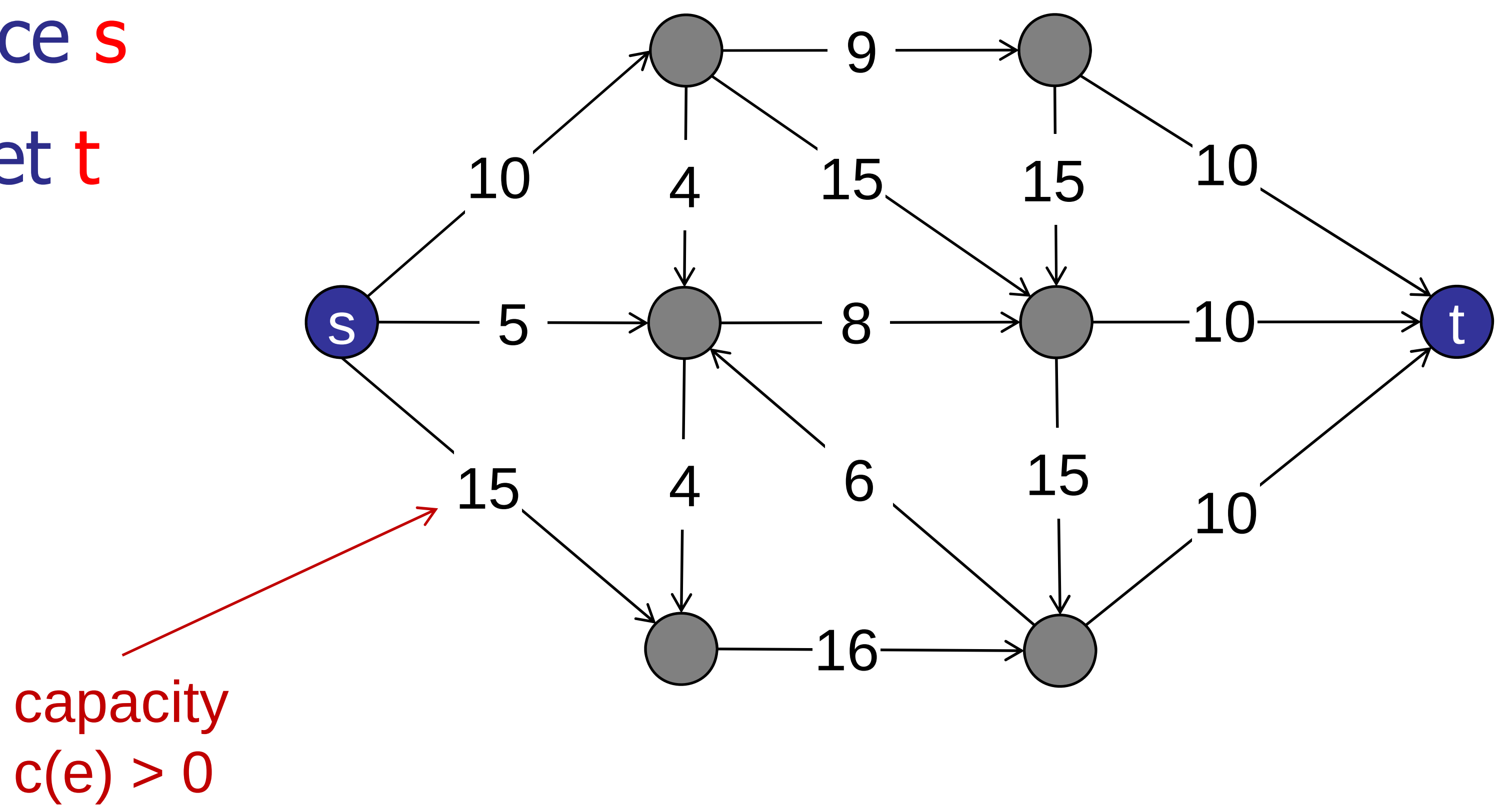
Examples:

- 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

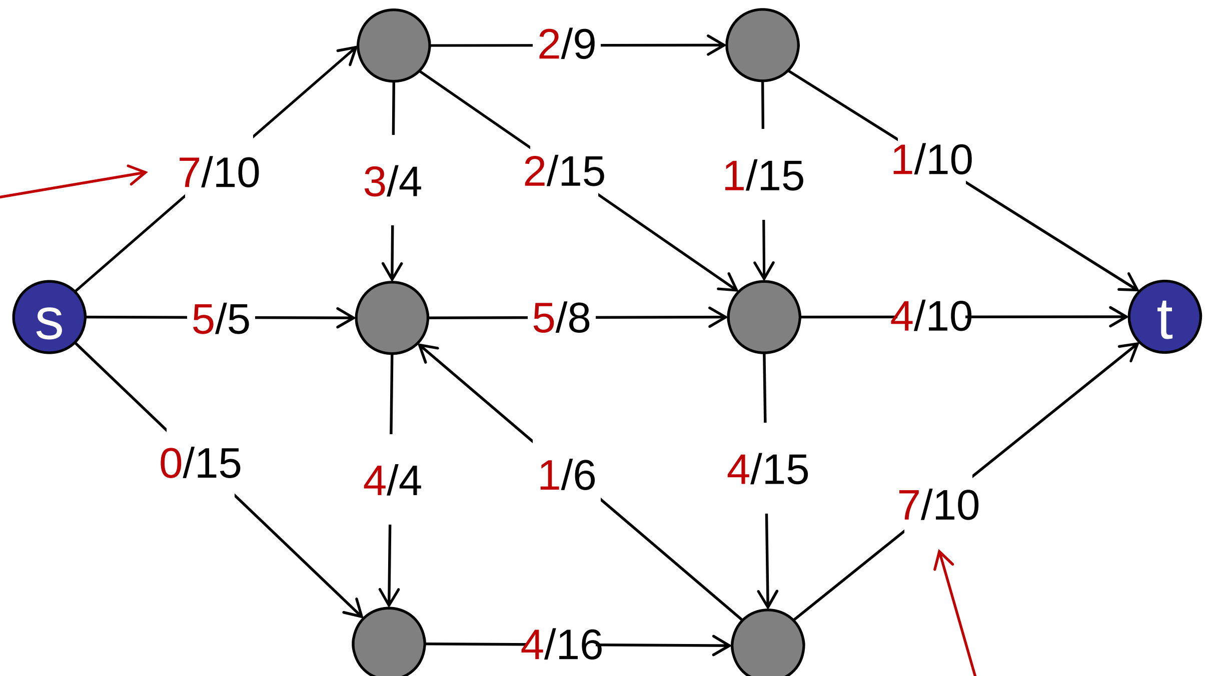
Output: Flow

– Assignment of flow f to each edge

Example:

capacity is 10:
“c(e) = 10”

flow is 7:
“f(e) = 7”

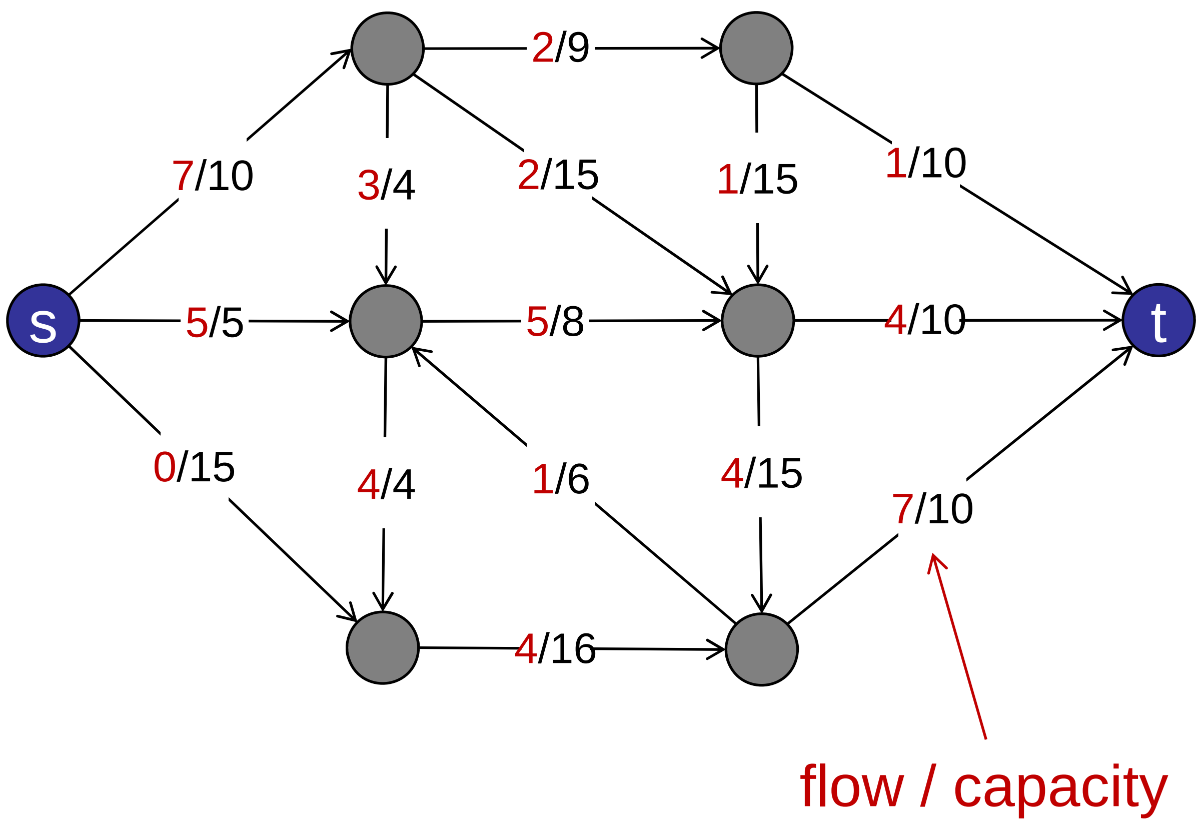


flow / capacity

Network Flow

Output: Flow

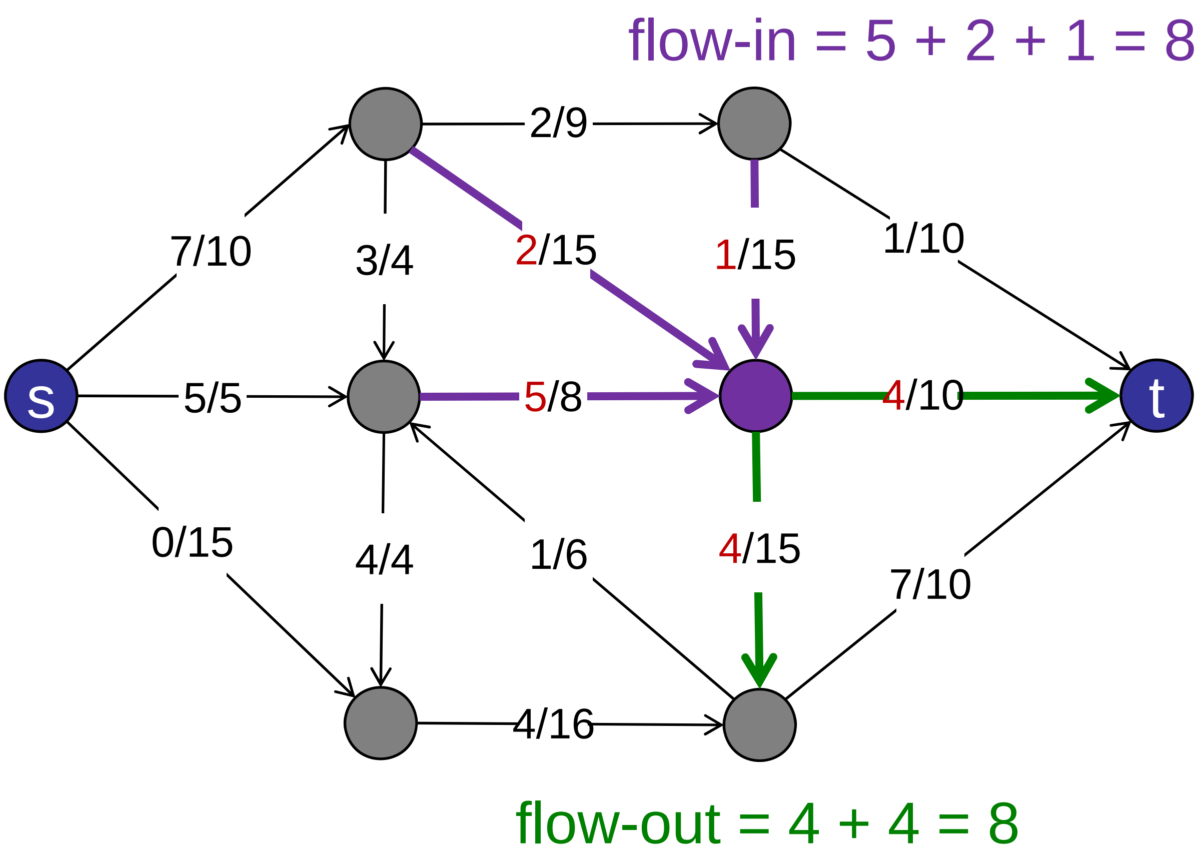
- Flow is not negative: for every edge e, $0 \leq f(e)$
- Flow ≤ capacity: for every edge e, $f(e) \leq c(e)$



Network Flow

Equilibrium constraint:

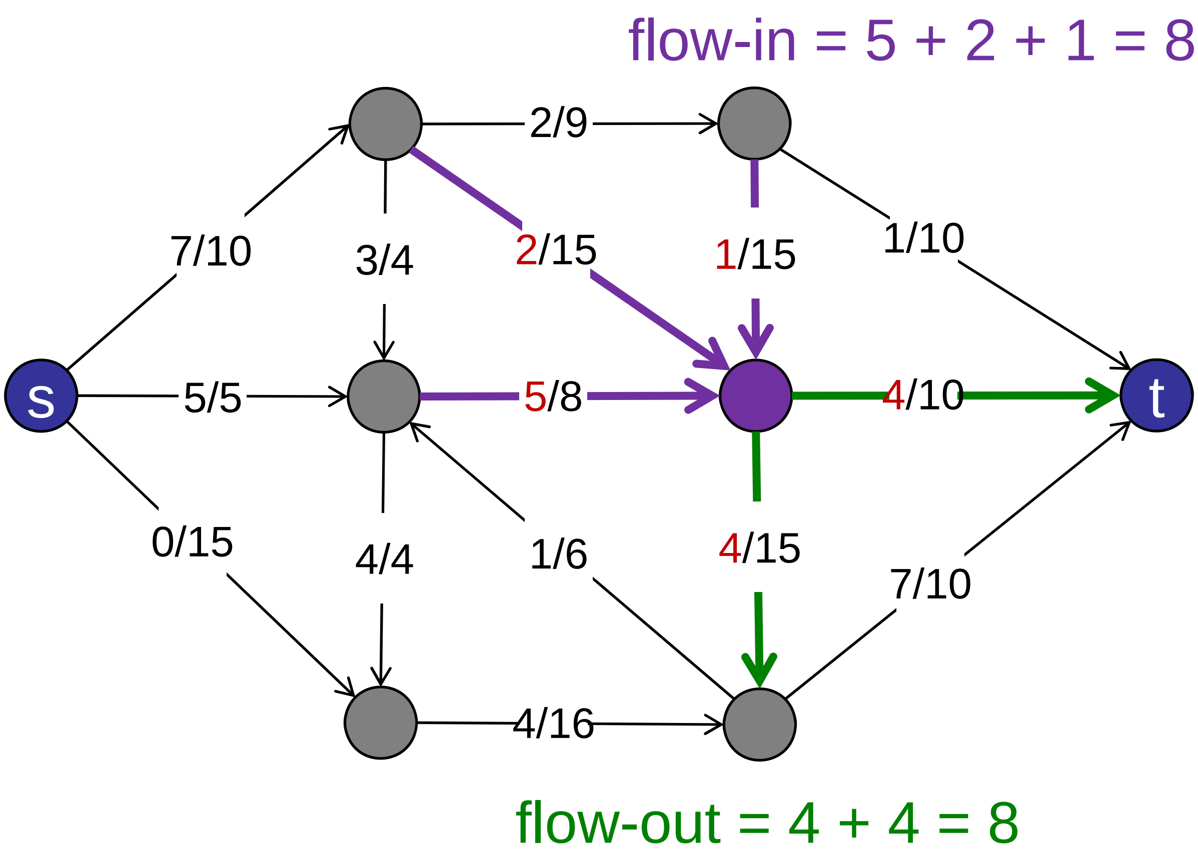
– For every node: flow-in = flow-out



Network Flow

Equilibrium constraint:

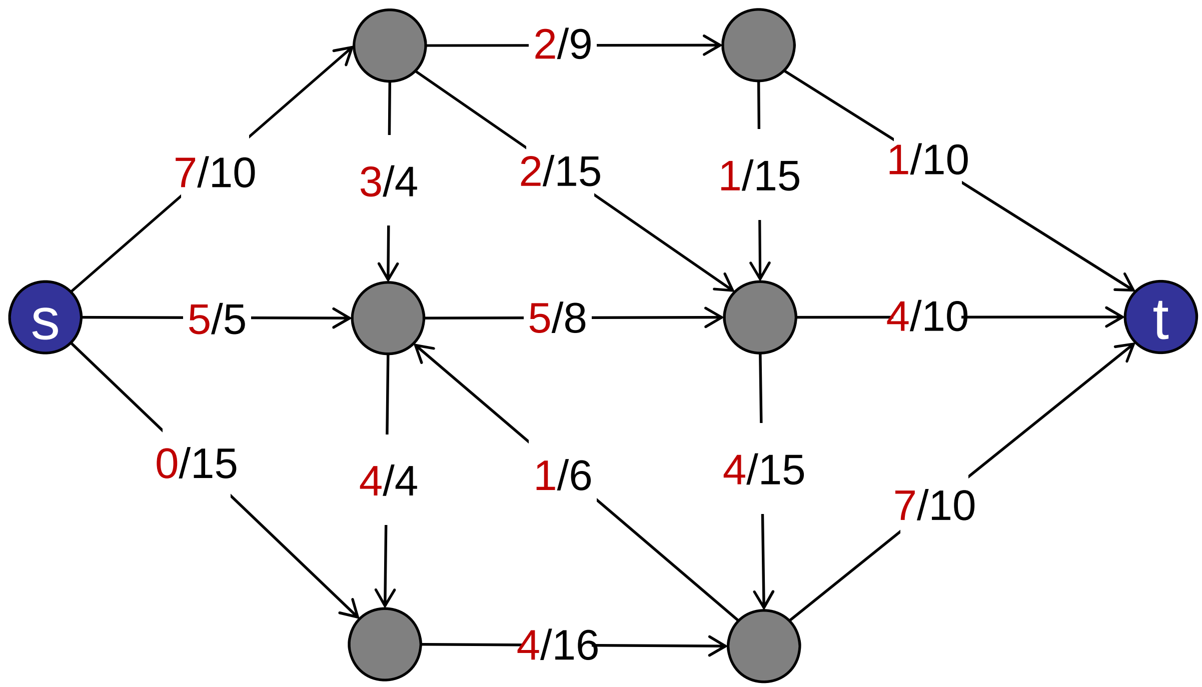
- For every node: flow-in = flow-out
- Except s and t



Network Flow

Output: st-flow

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

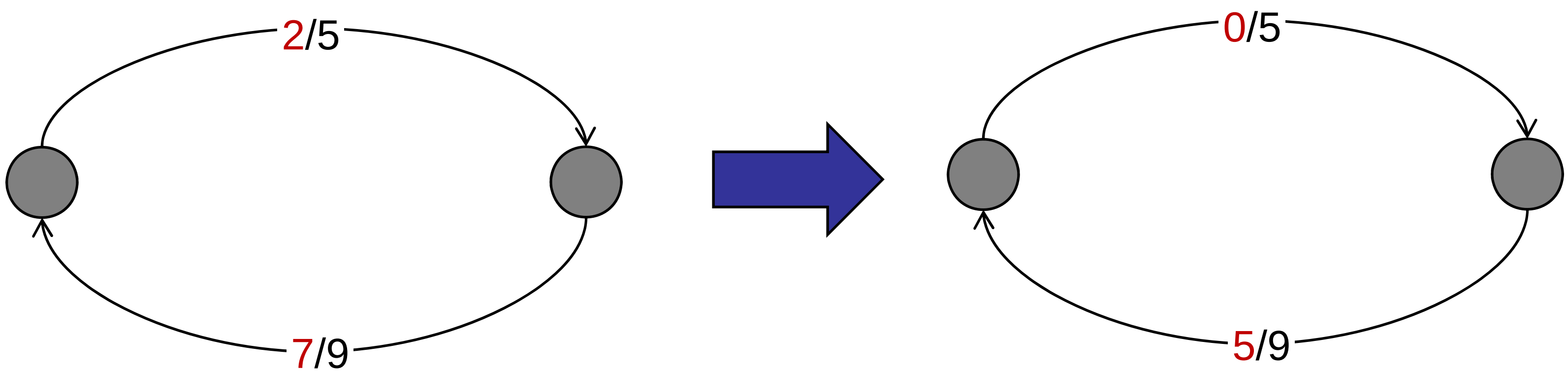


Network Flow

Uni-directional flow: st-flow

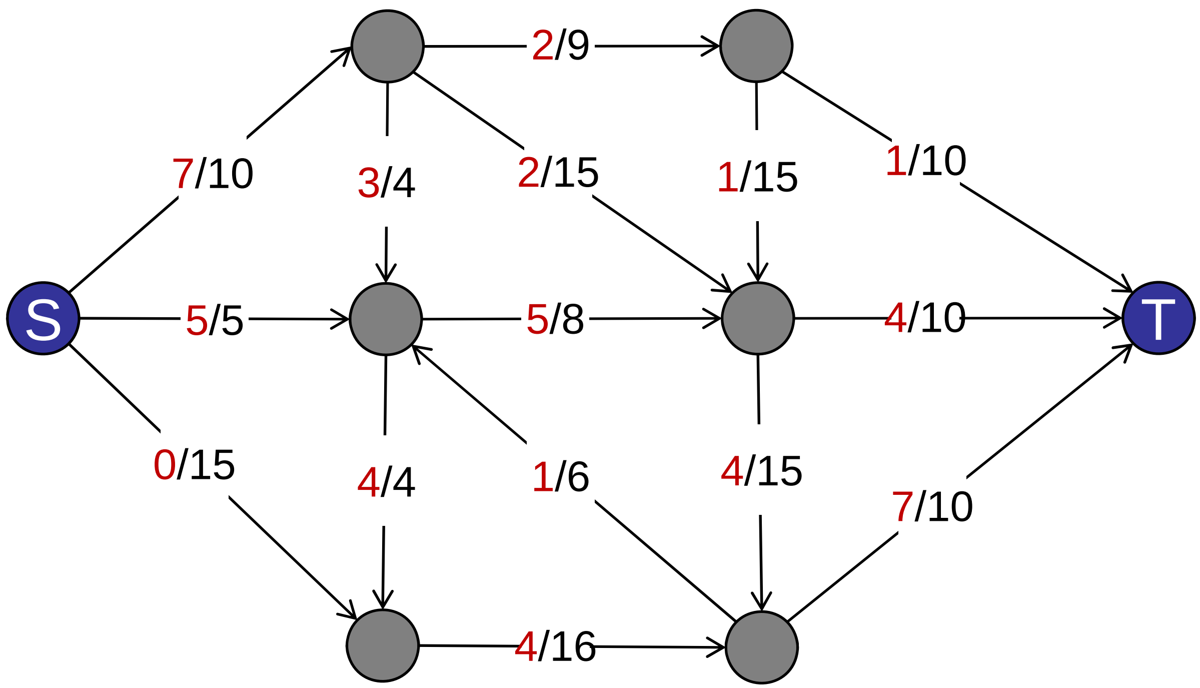
If f(u,v)>0 and f(v,u)>0, then they cancel out.

Flows only go in one direction.



Value of a Flow

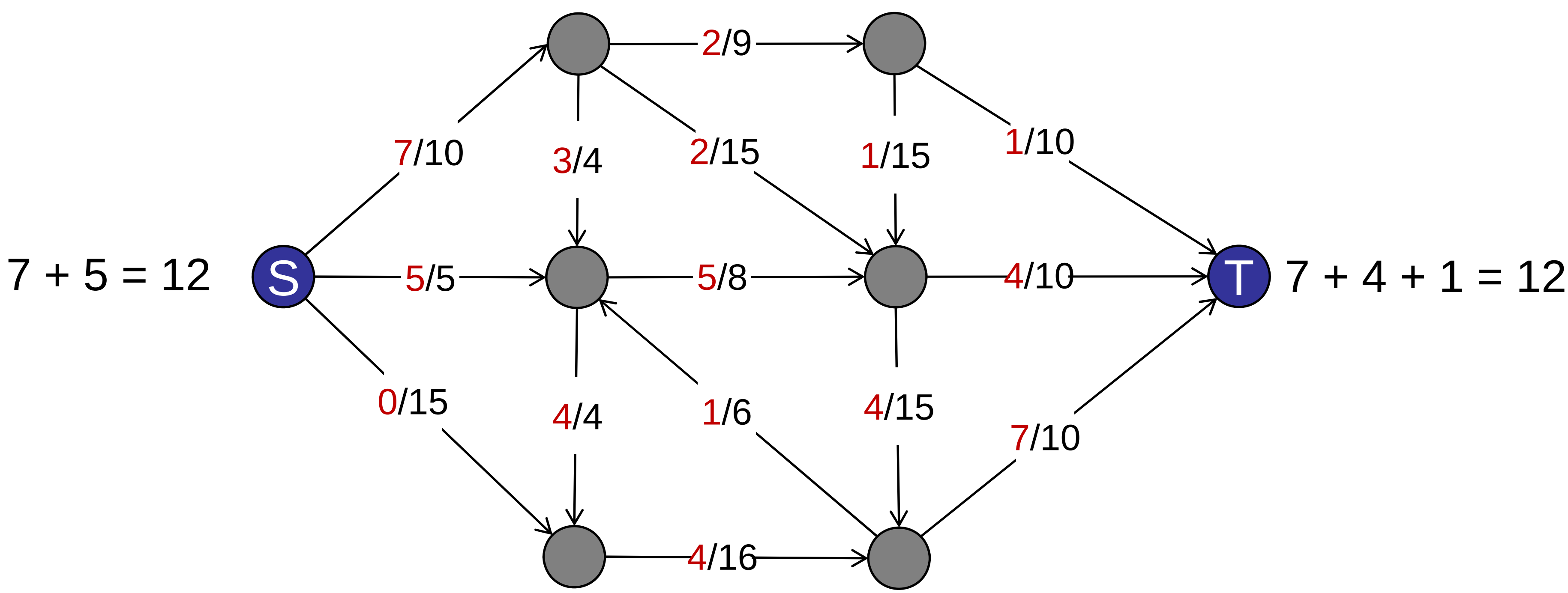
How much stuff gets from s to t?



Value of a Flow

How much stuff gets from s to t?

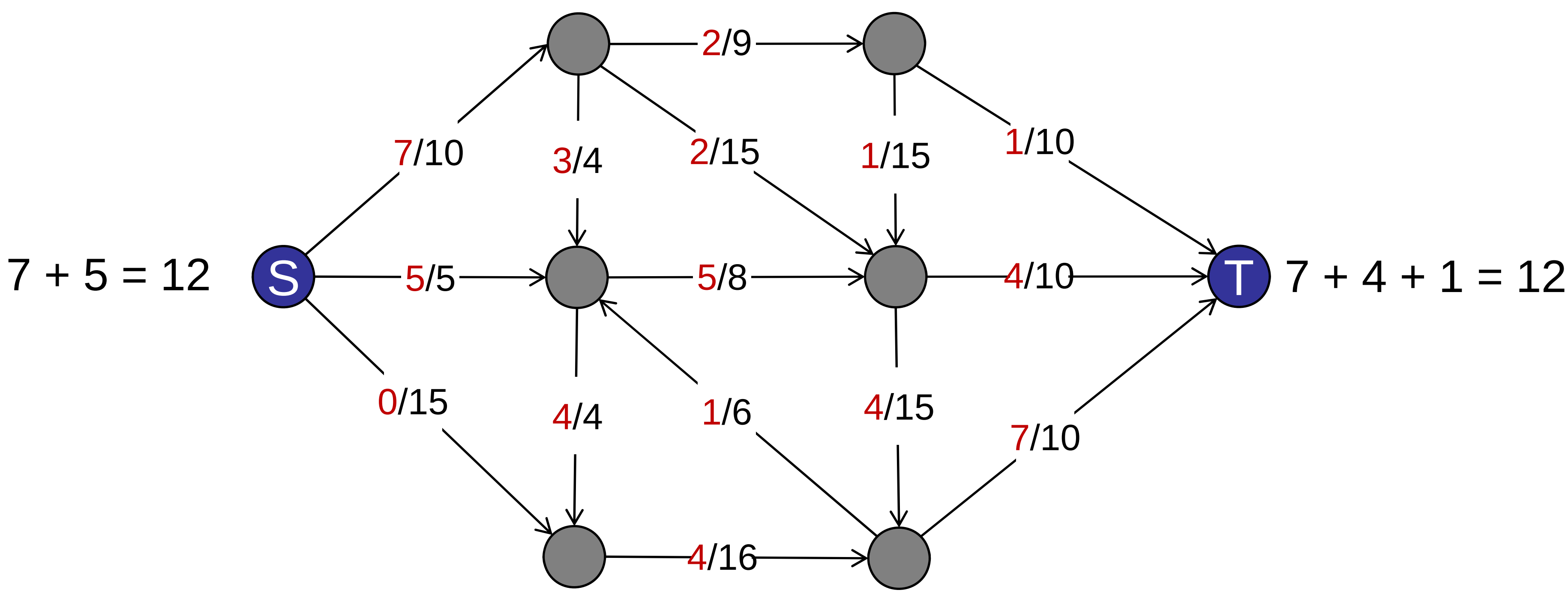
- How much leaves source?
- How much gets to target?



Value of a Flow

Definition:

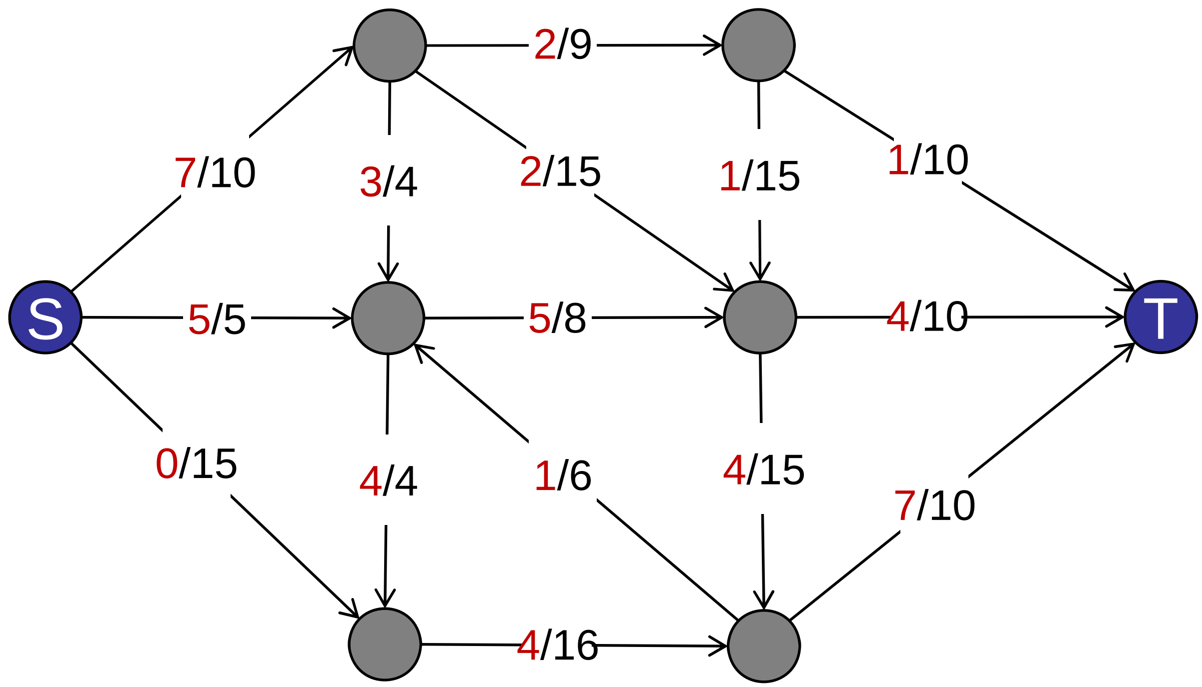
For a flow f: $\text{value}(f) = \sum_{v:(s,v)\in E} f(s,v)$



Max Flow

Goal:

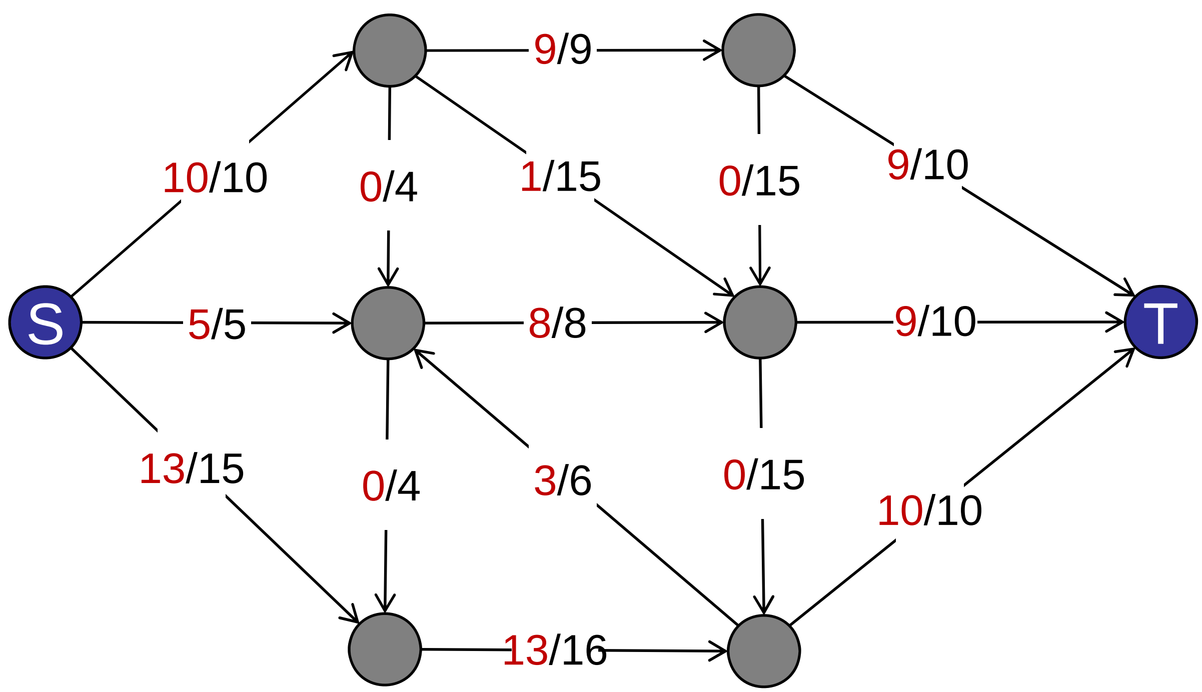
Find an st-flow with maximum value.



Max Flow

Goal:

Find an st-flow with maximum value.



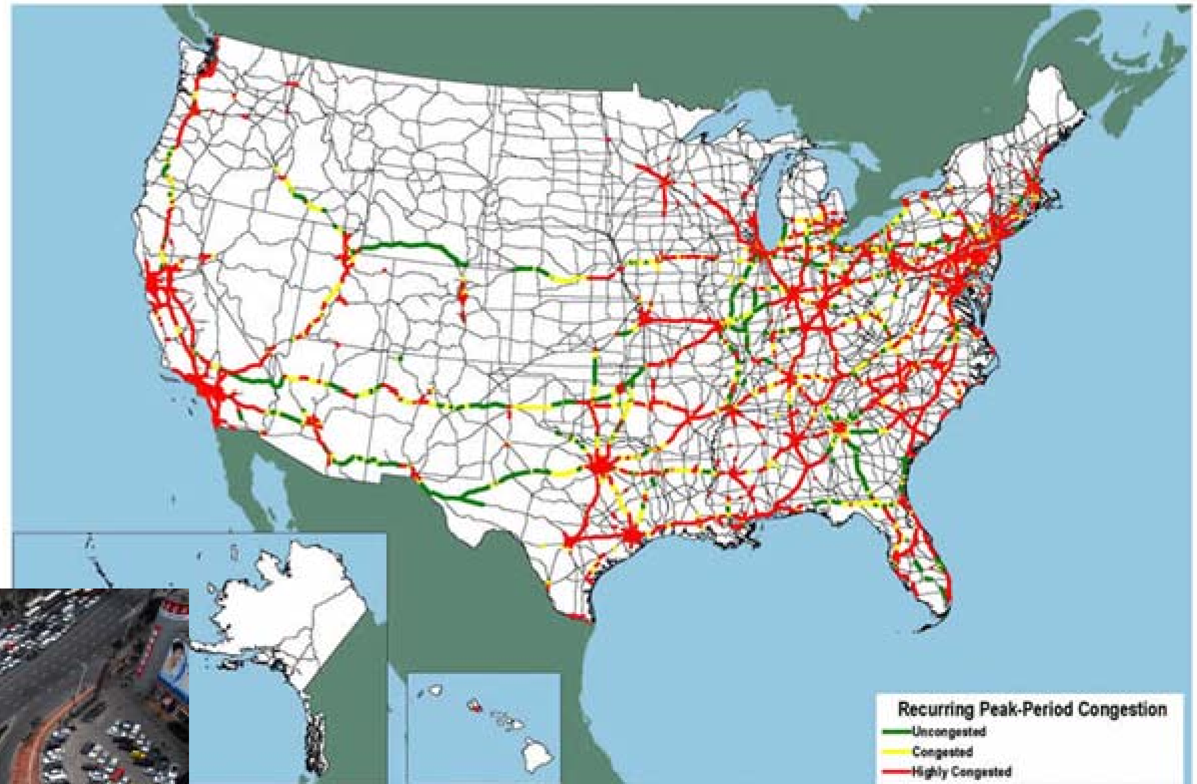
value = 28

Roadmap

Network Flows

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

Sample Problems

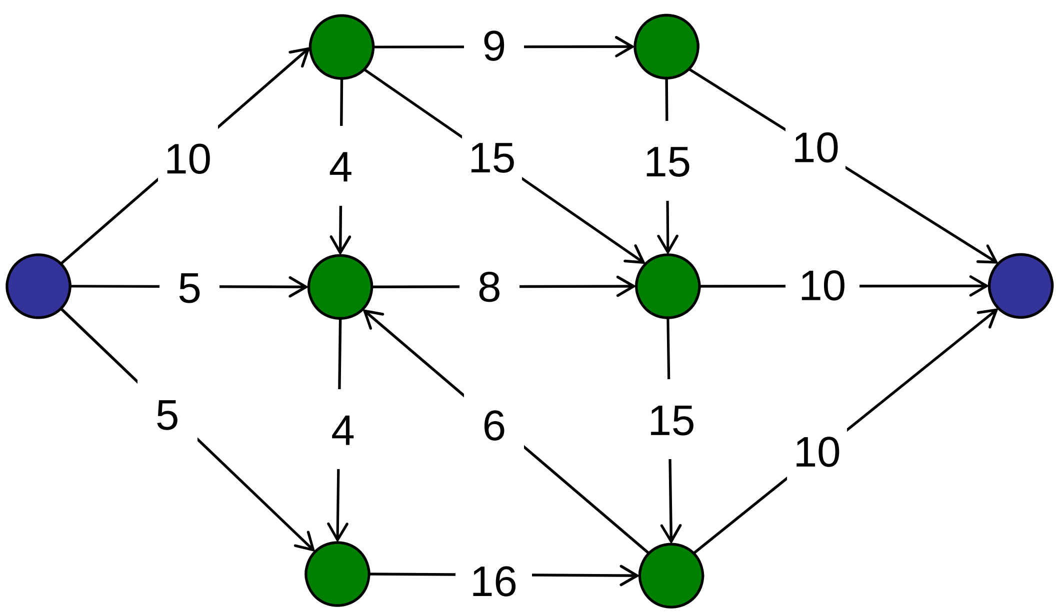


Classic Flow Problems

Moving traffic:

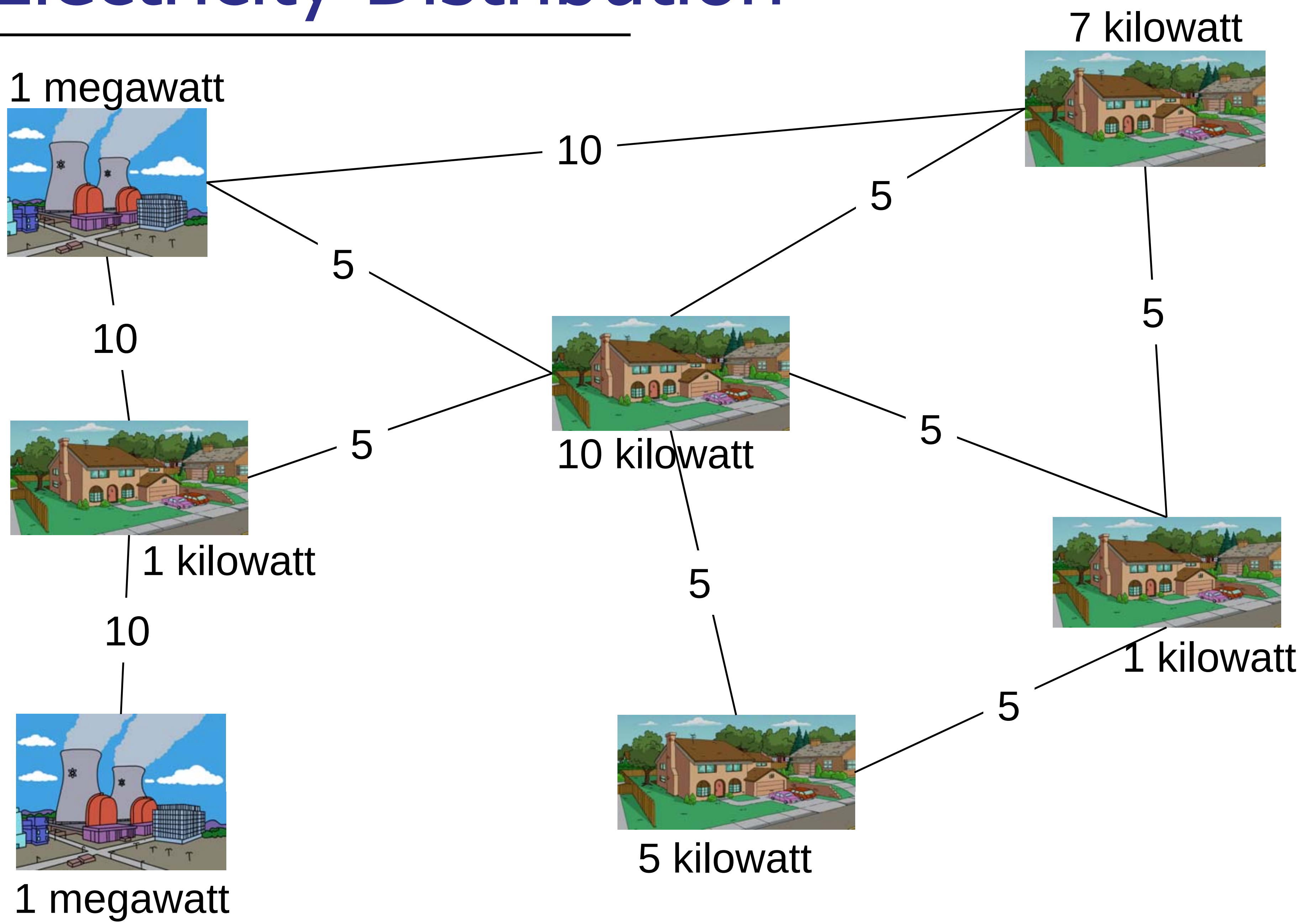
- source: entry point of high-traffic zone
- edges: roads with capacities in cars/hour
- target: exit point of high-traffic zone

What is the max cars/hour that can transit the zone?



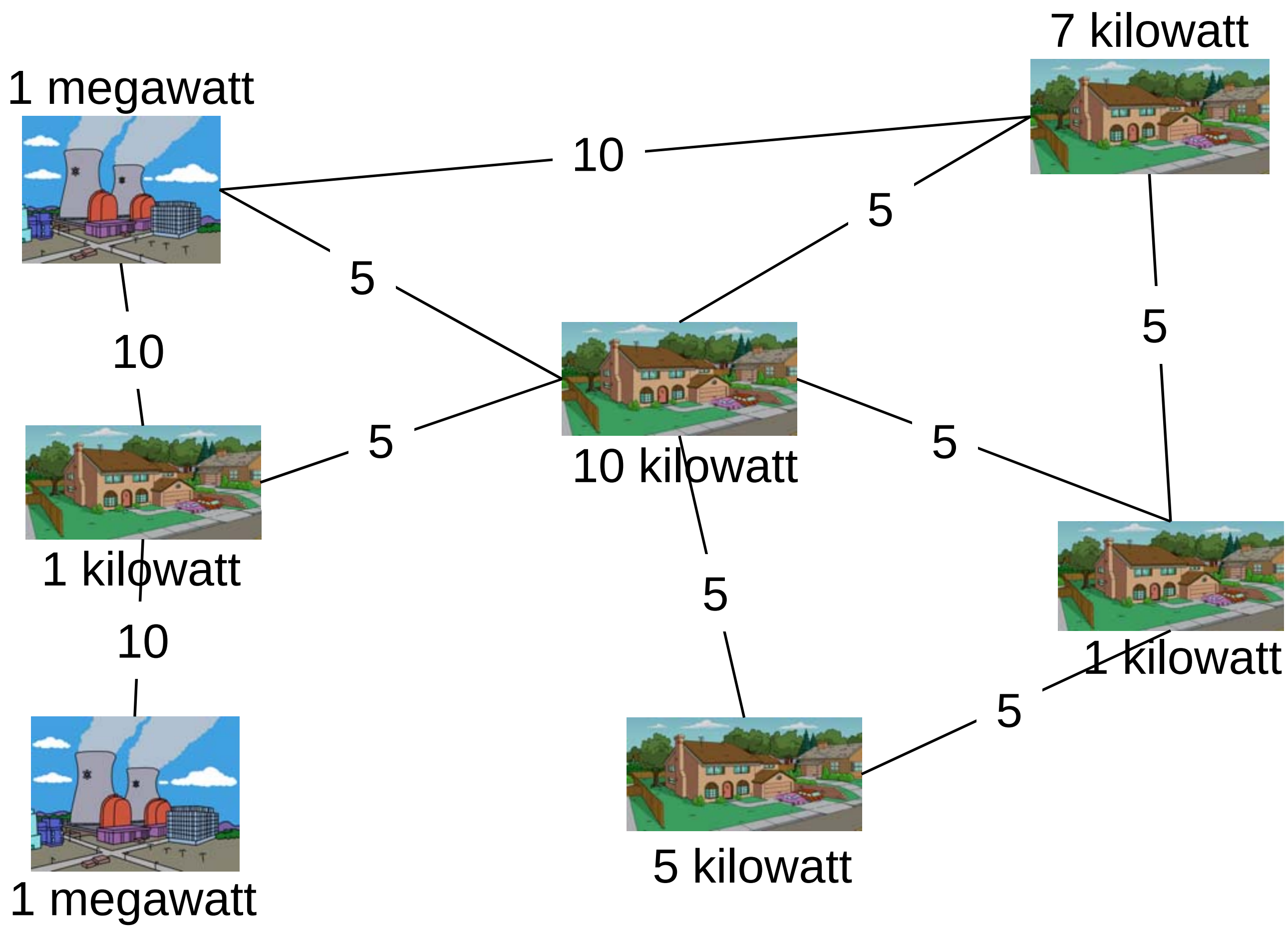
Sample Problems

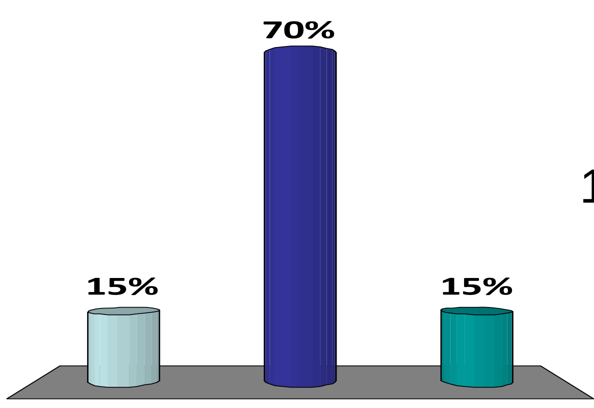
Electricity Distribution



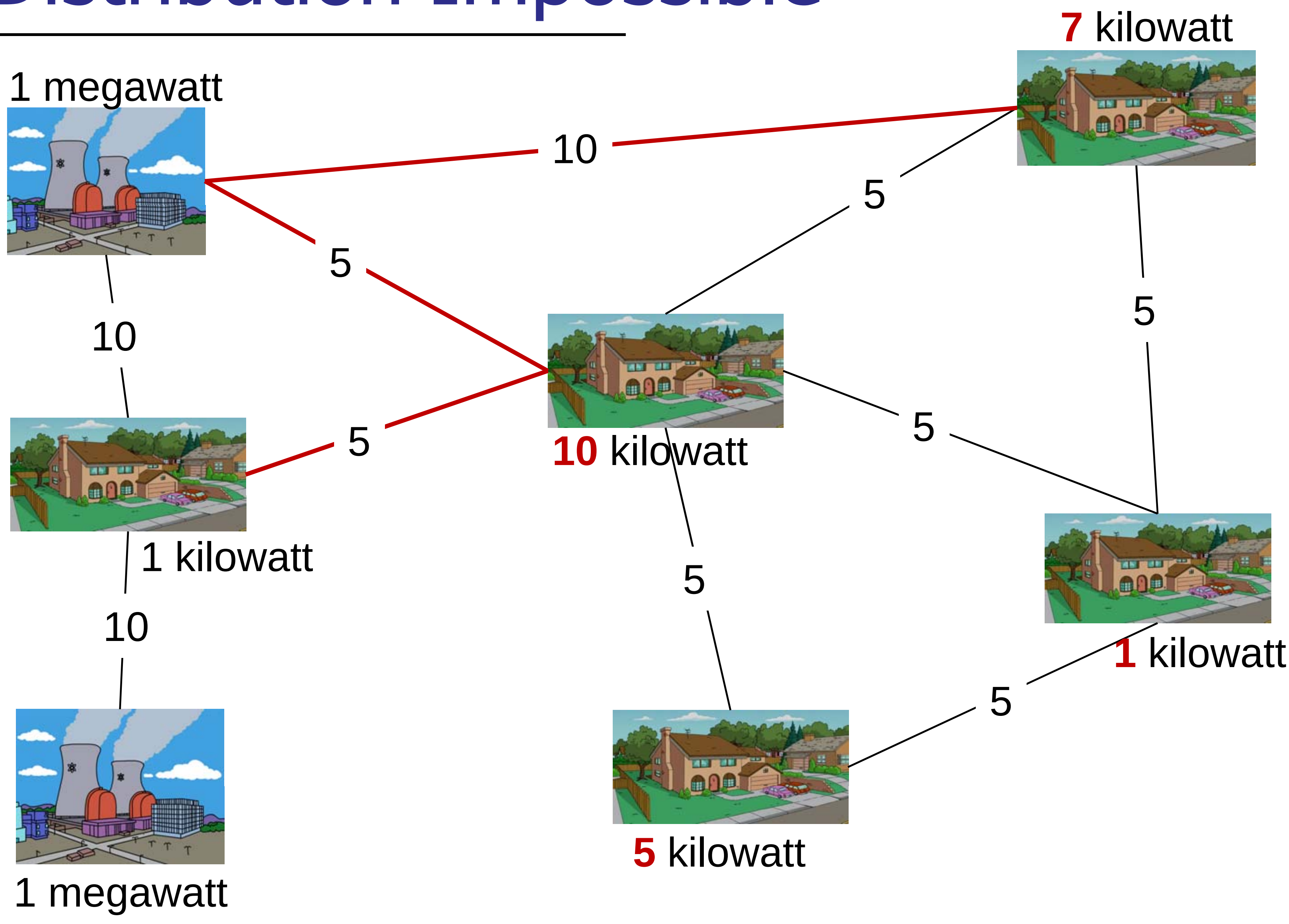
Can every home get power?

1. Yes
2. No
3. I’m lazy.



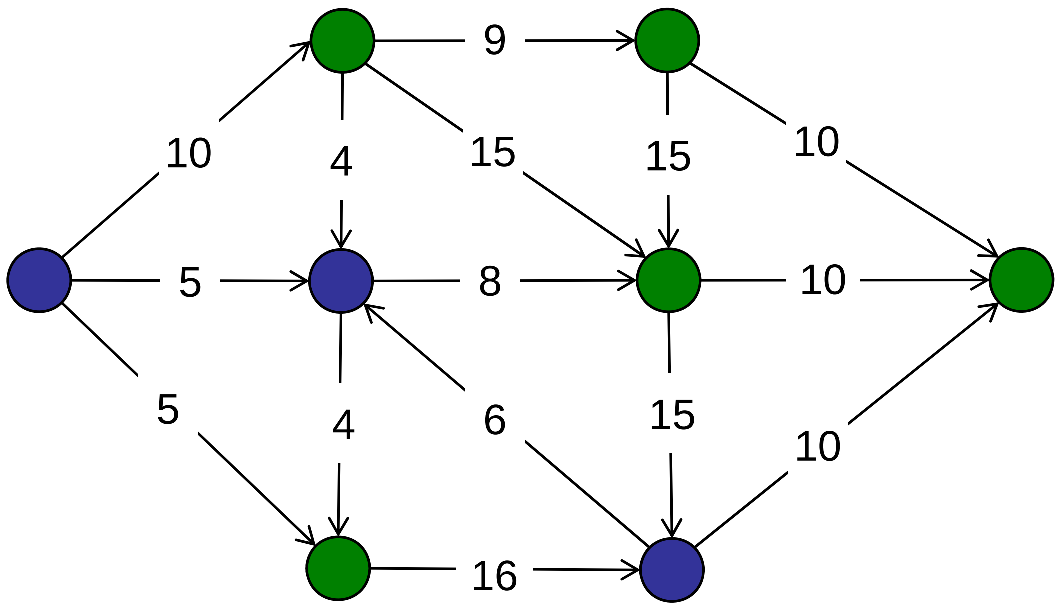


Distribution Impossible



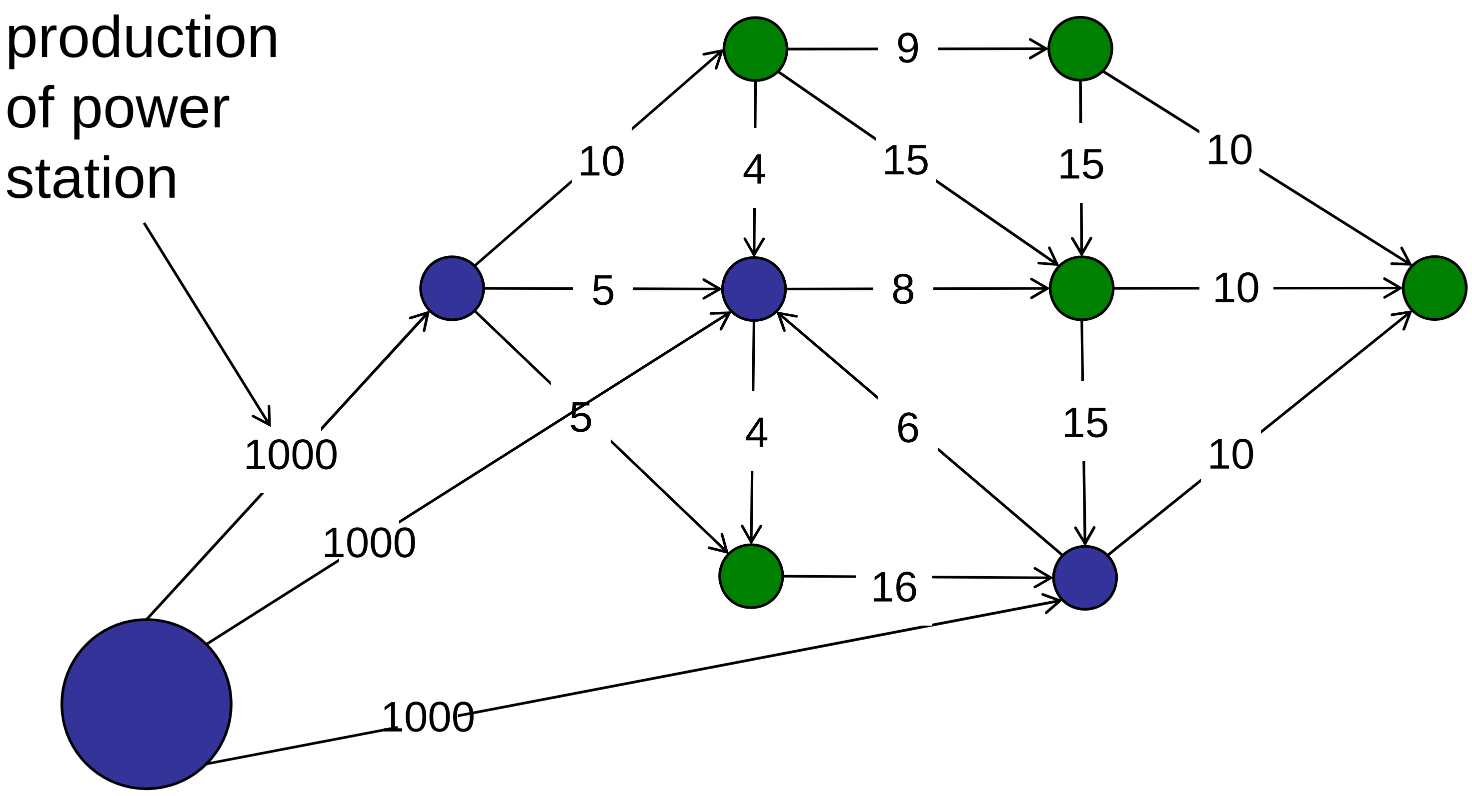
Electricity Distribution

- Blue = power.
- Green = house.



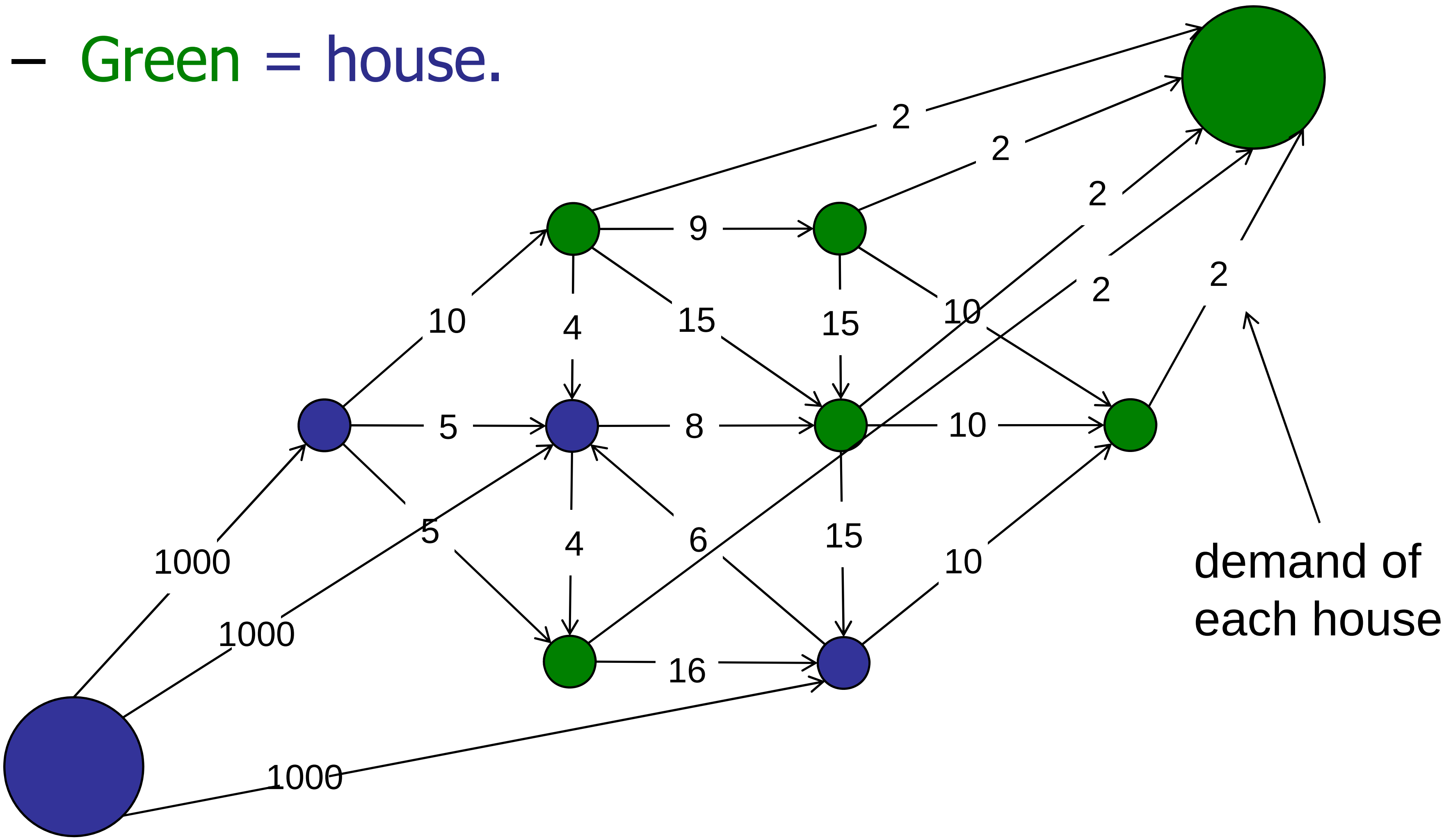
Electricity Distribution

- Blue = power.
- Green = house.



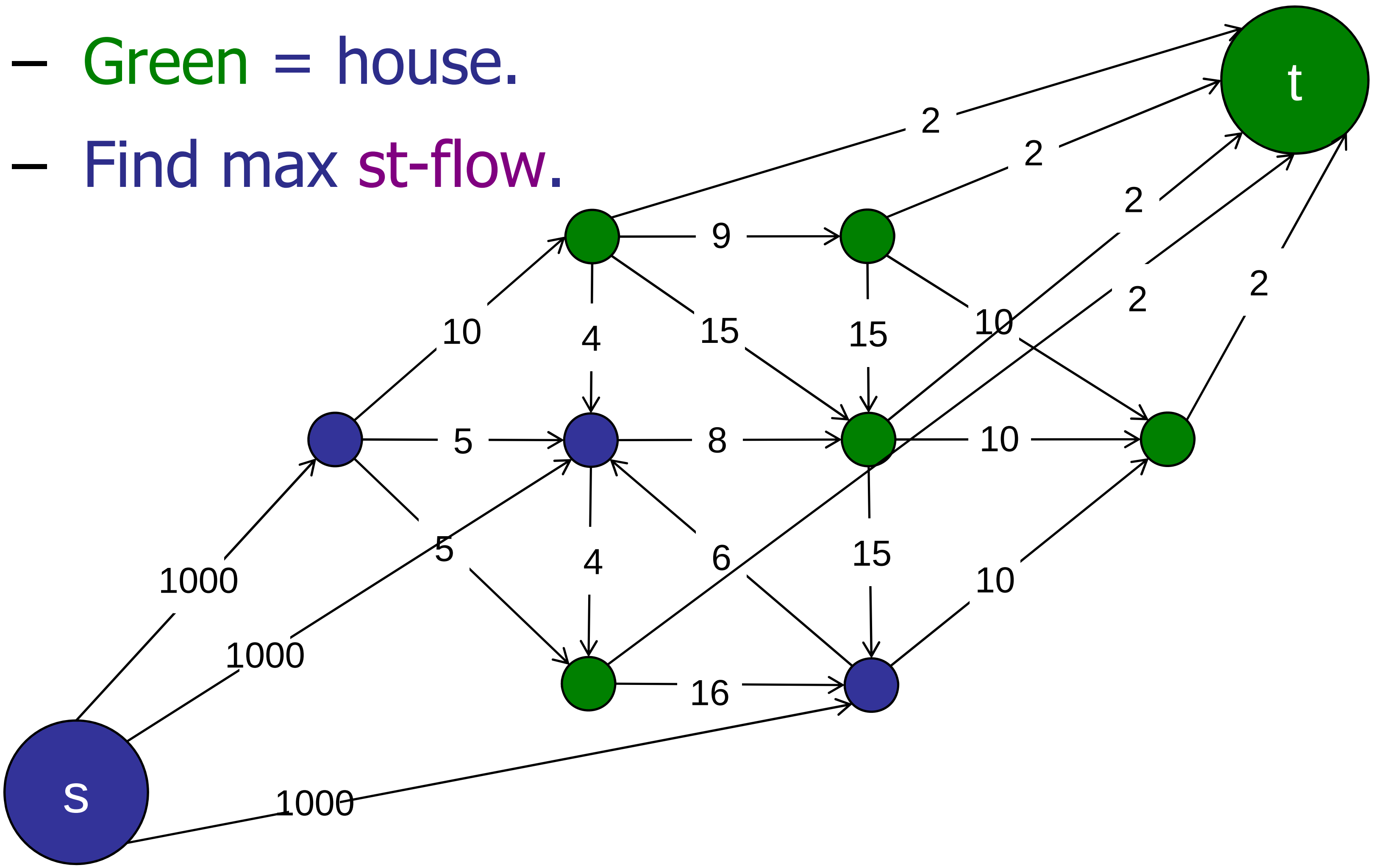
Electricity Distribution

– Blue = power.
– Green = house.



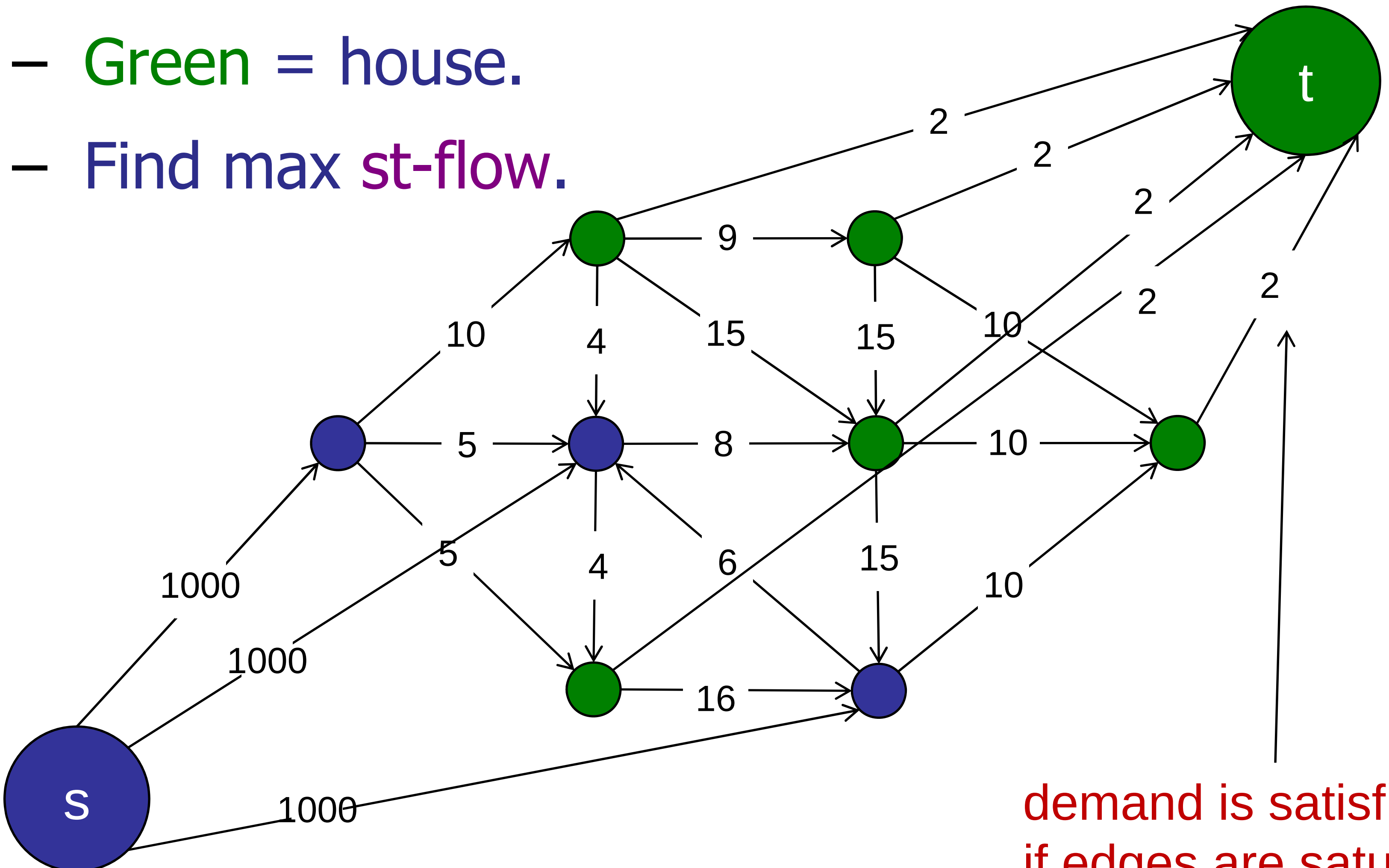
Electricity Distribution

- Blue = power.
- Green = house.
- Find max st-flow.



Electricity Distribution

- Blue = power.
- Green = house.
- Find max st-flow.



demand is satisfied
if edges are saturated

Sample Problems

Christmas Time!

Lots of kids:

Alice, Bob, Carol, David, Ernie, Fran, George, …

Lots of presents:

Apple, Barnie, Candy, Doll, Elmo, Fire engine, Giraffe

Each present is acceptable to some kids:

Apple: Alice, Bob, Carol

Barnie: Carol, Ernie, George

Christmas Time!

Lots of kids:

- Alice, Bob, Carol, David, Ernie, Fran, George, …

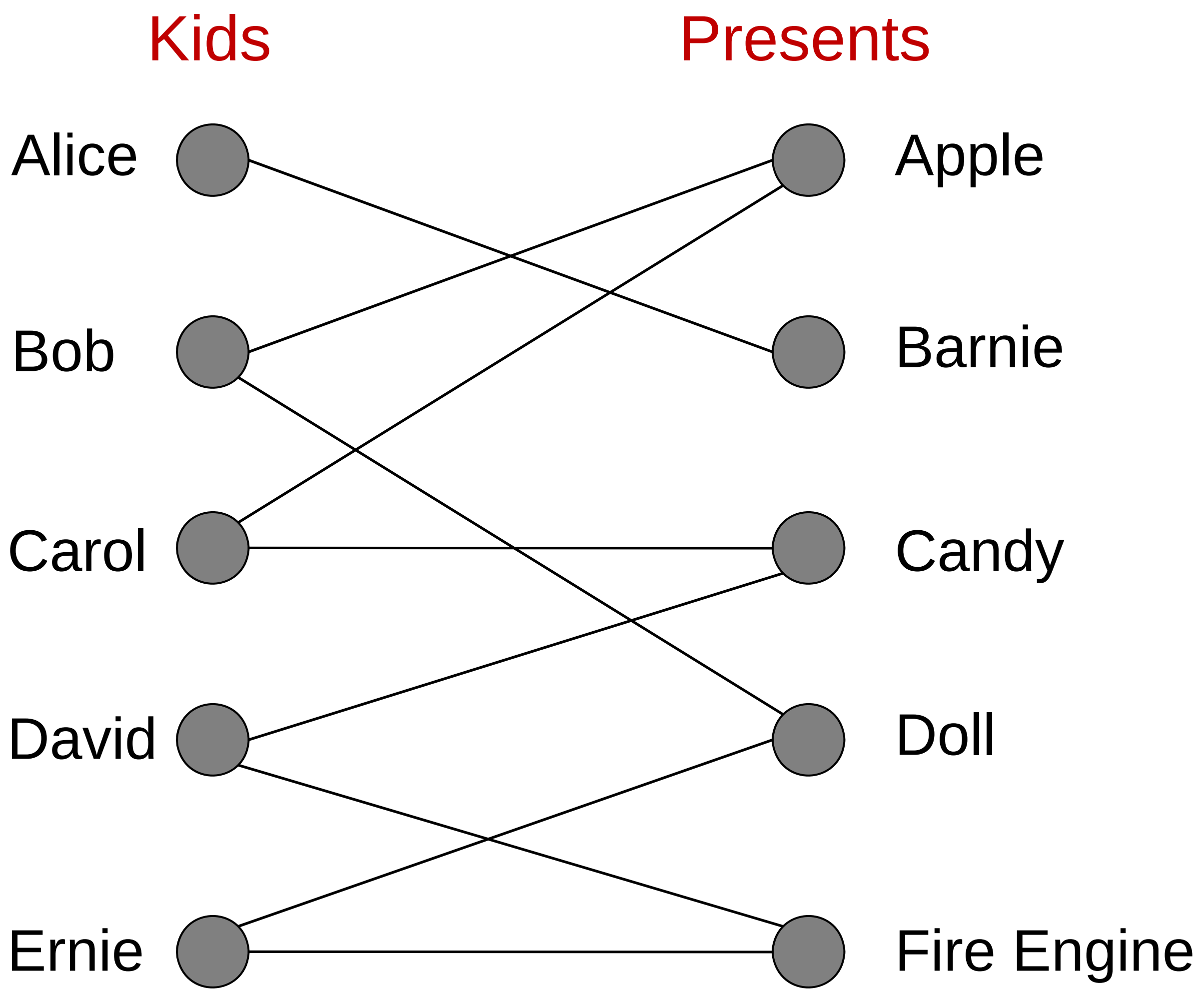
Lots of presents:

- Apple, Barnie, Candy, Doll, Elmo, Fire engine, Giraffe

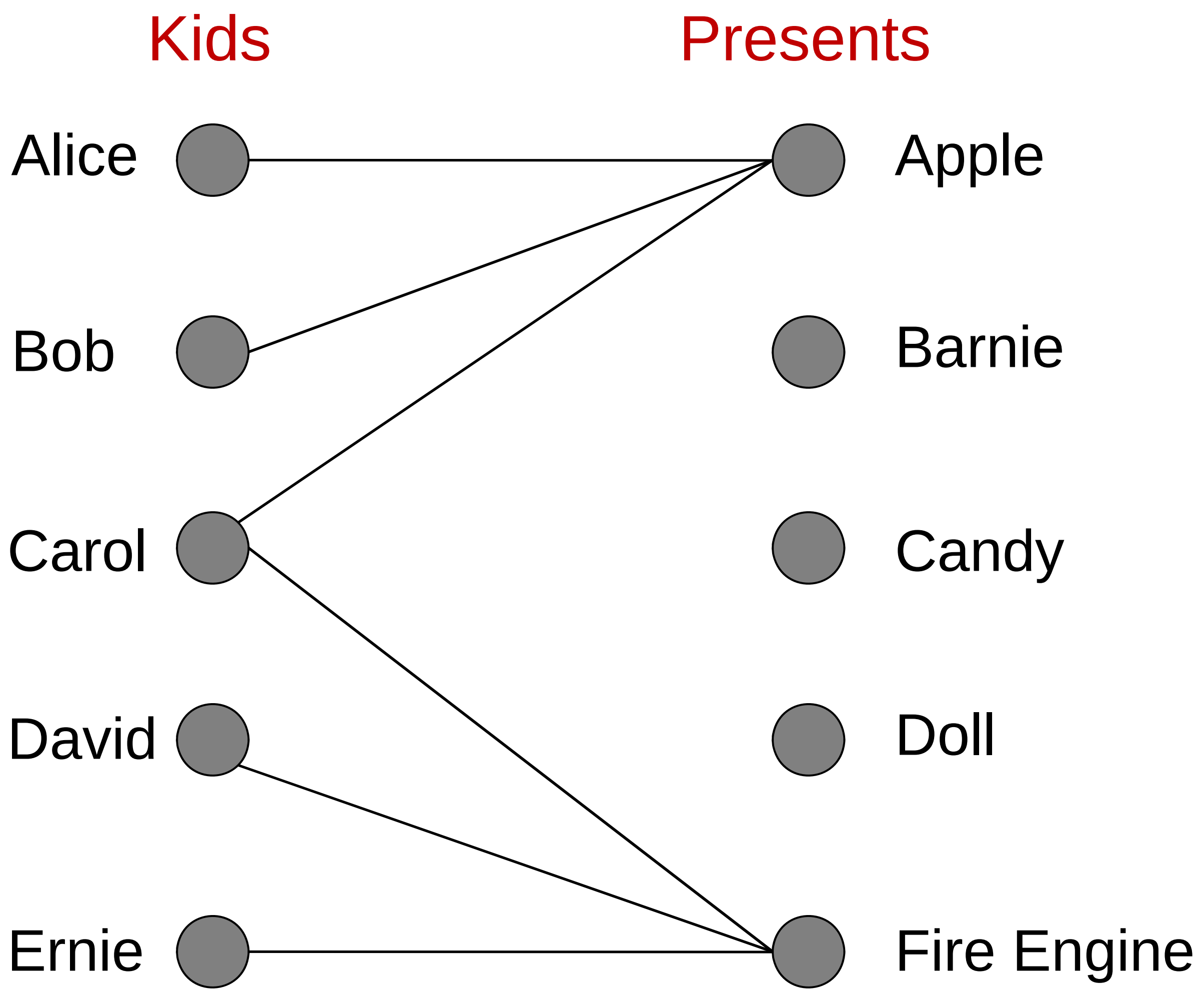
Which present is given to which kid?

- Maximize the number of kids that get a present.

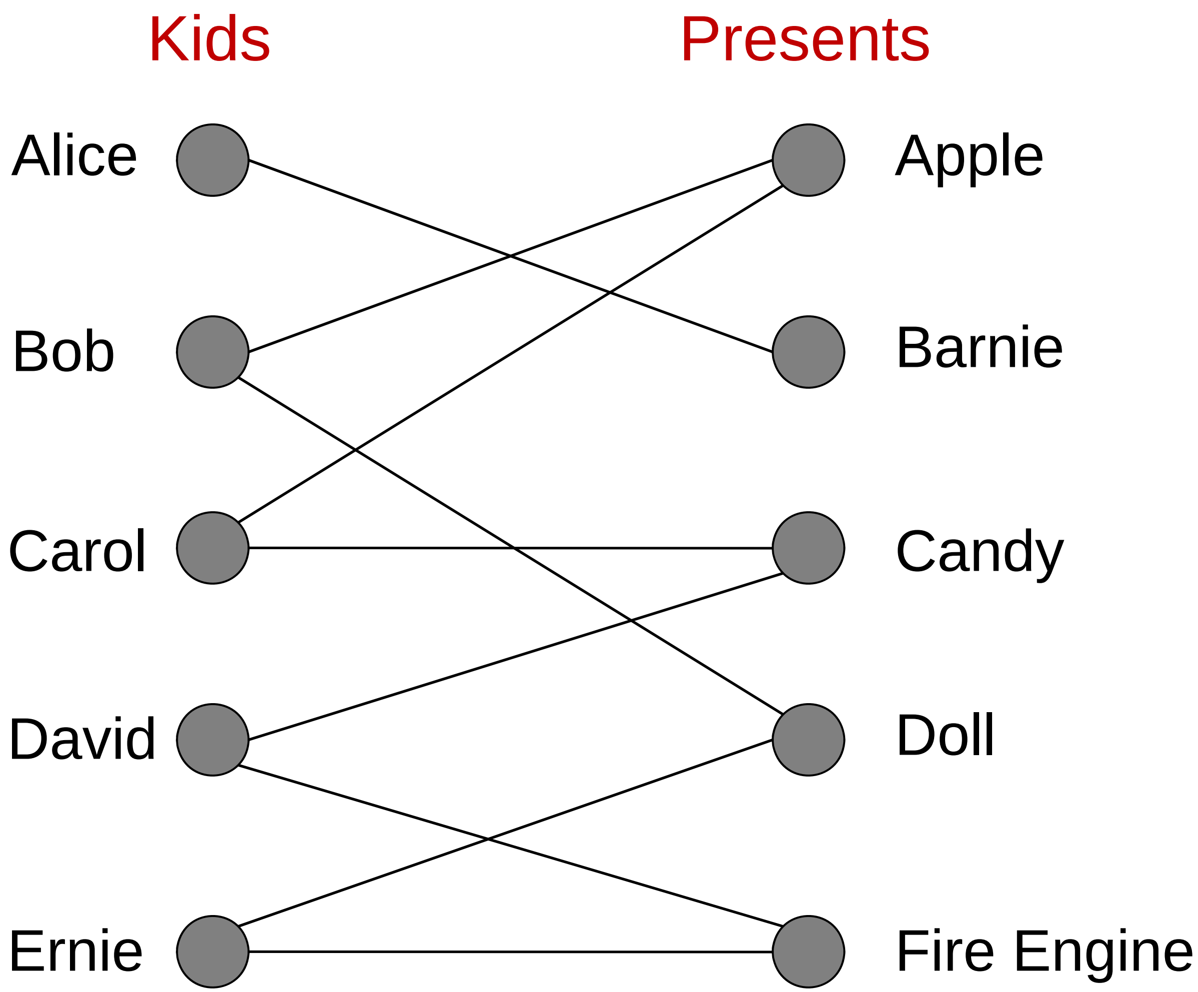
Christmas Time!



Christmas Time!

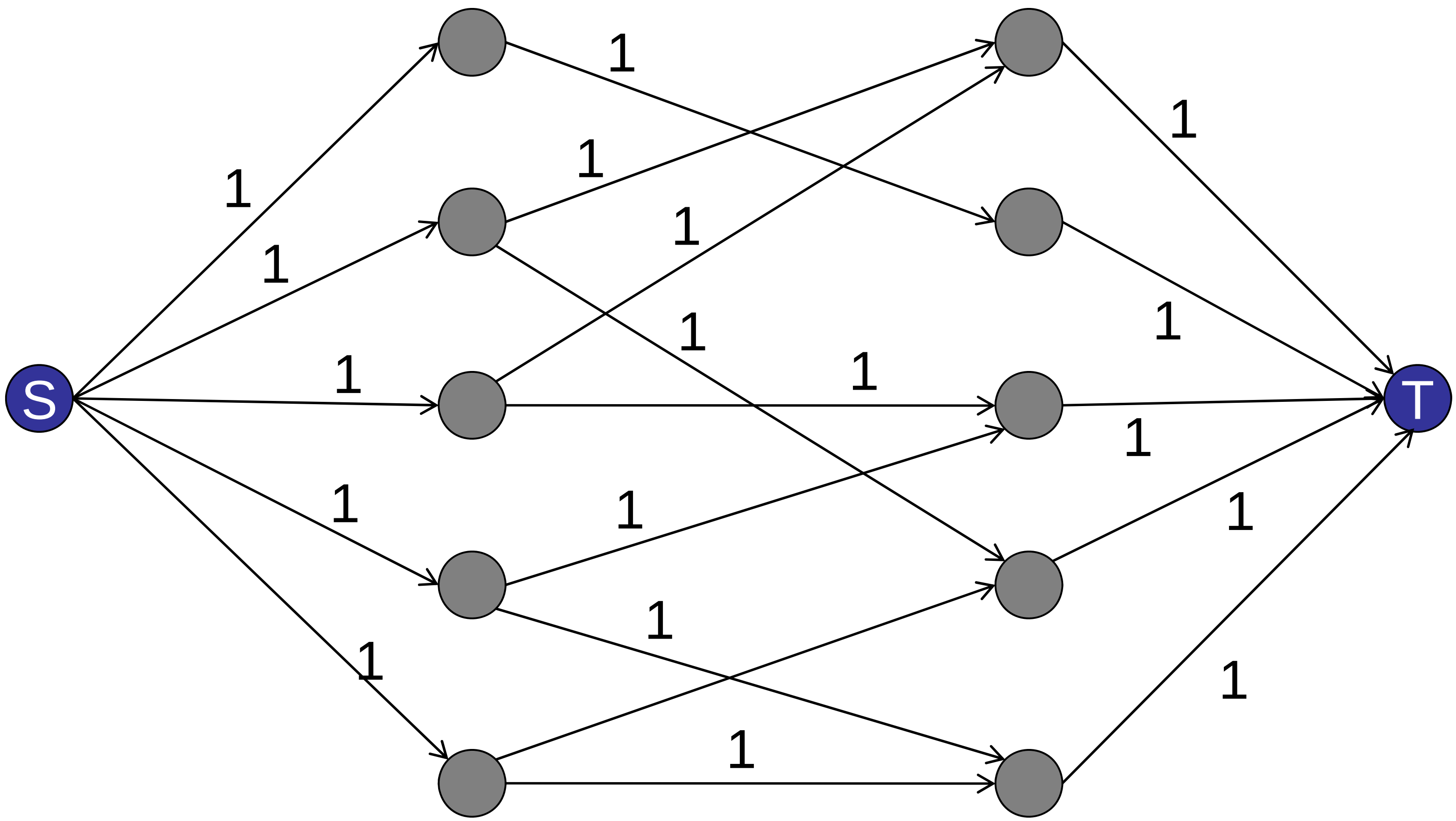


Maximum Bipartite Matching



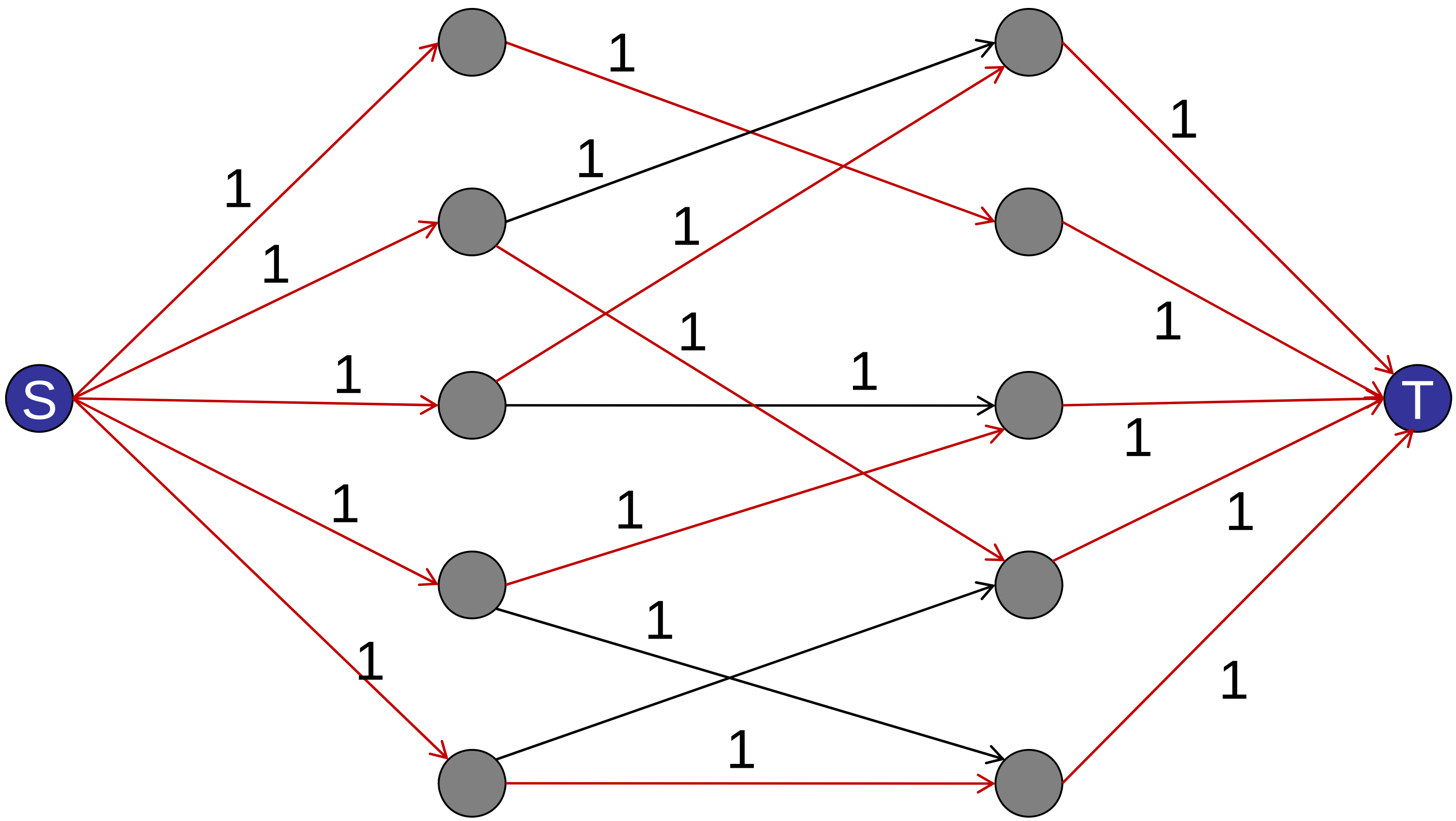
Maximum Bipartite Matching

Define a flow network with unit cost edges.



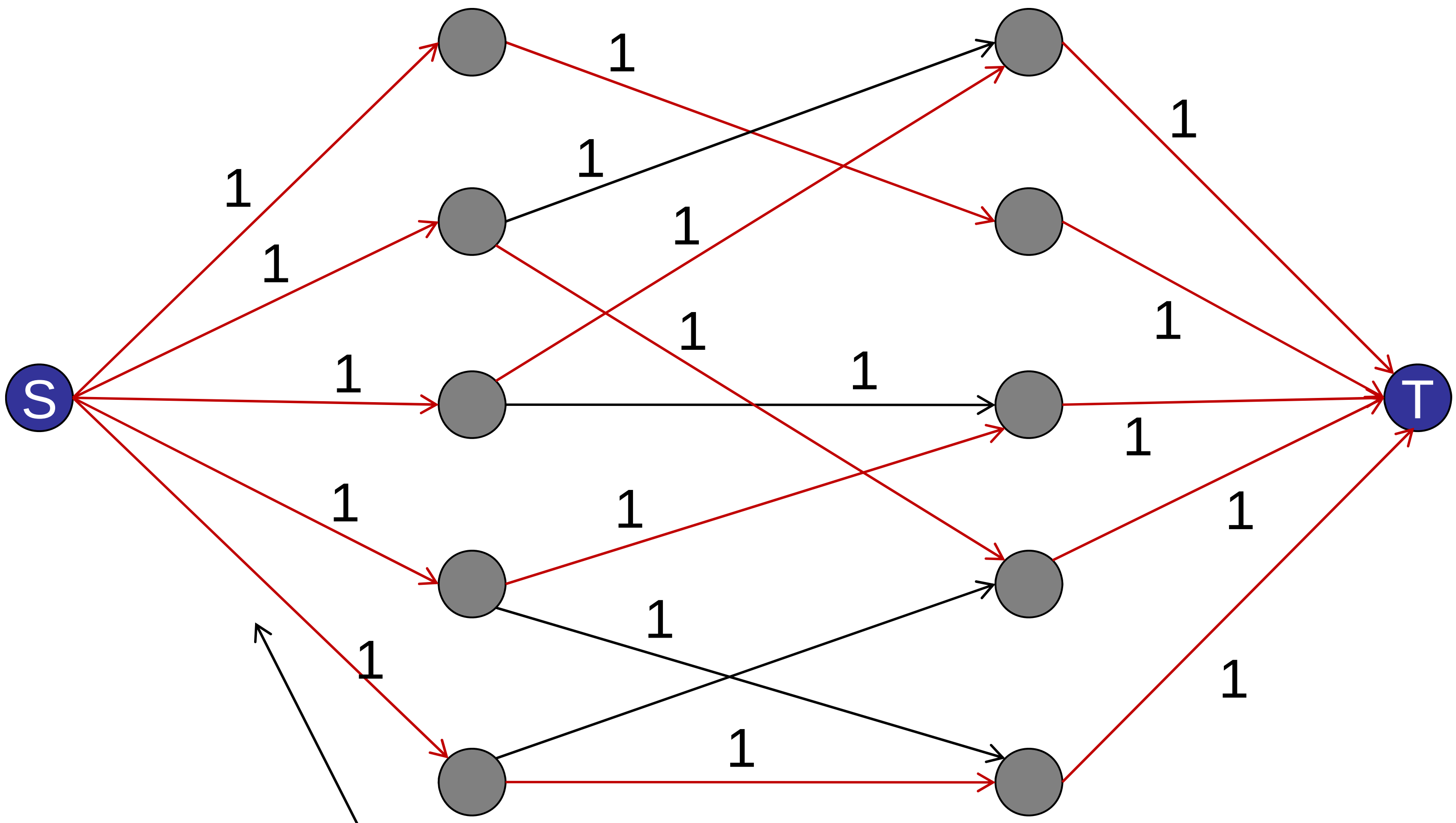
Maximum Bipartite Matching

Find a maximum st-flow.



Maximum Bipartite Matching

Find a maximum st-flow.

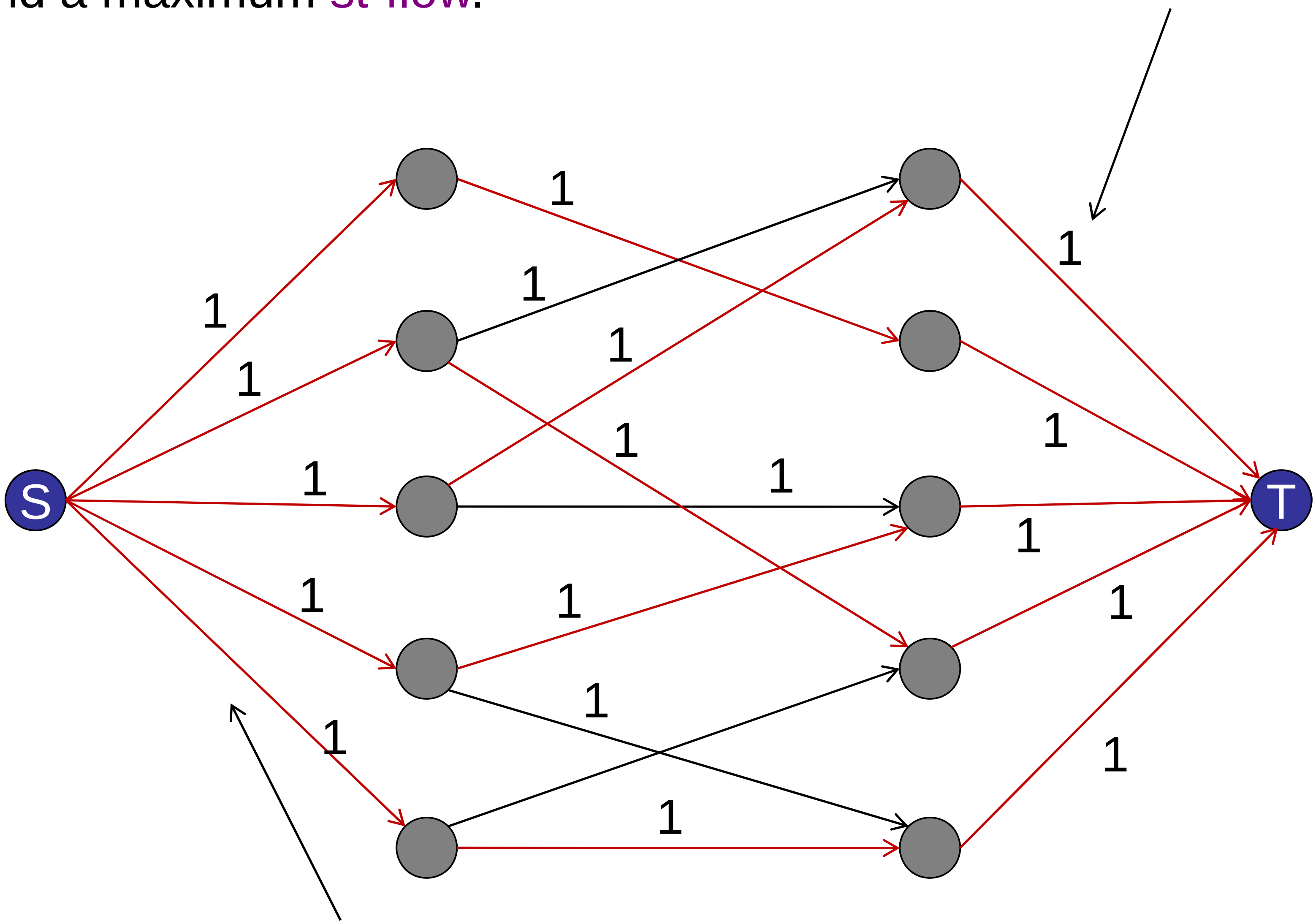


value(flow) = # children given a present

Maximum Bipartite Matching

Find a maximum st-flow.

value(flow) = # presents given

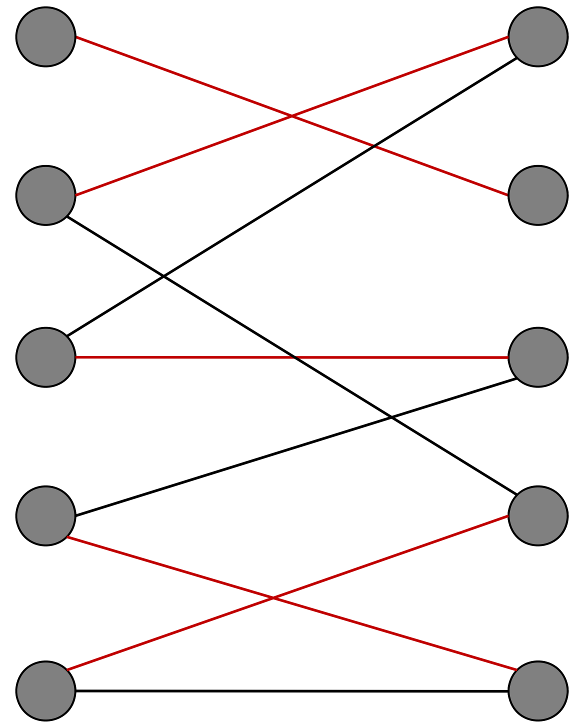


value(flow) = # children given a present

Bipartite Matching

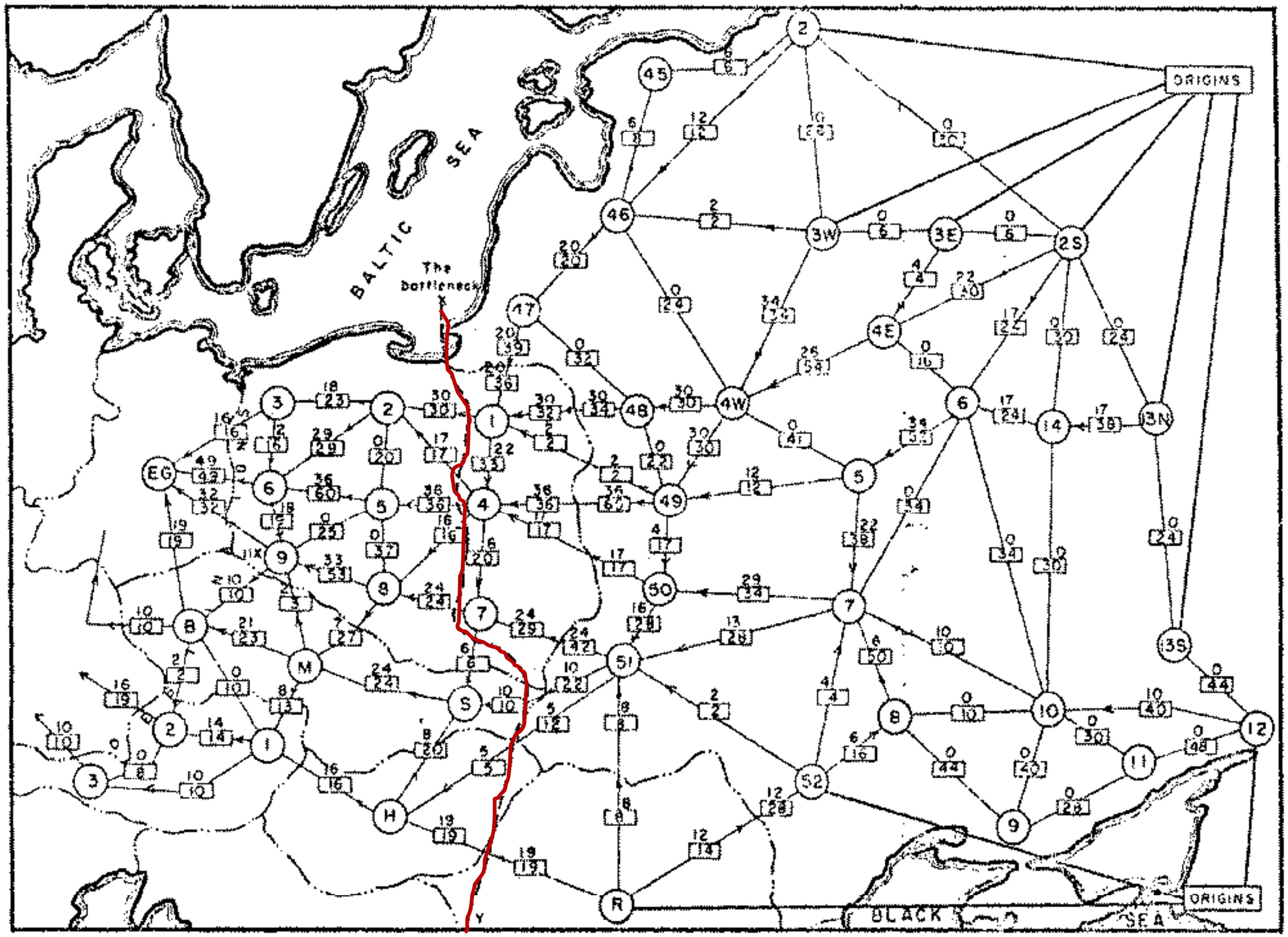
General problem:

- Bipartite graph: (A, B, E)
  - Vertices A
  - Vertices B
  - Edges E $\subseteq$ A x B
- Matching:
  - Subset of edges E
  - Each vertex has at most one adjacent edge.
- Goal: maximize matching



perfect matching: every vertex is matched.

Sample Problems



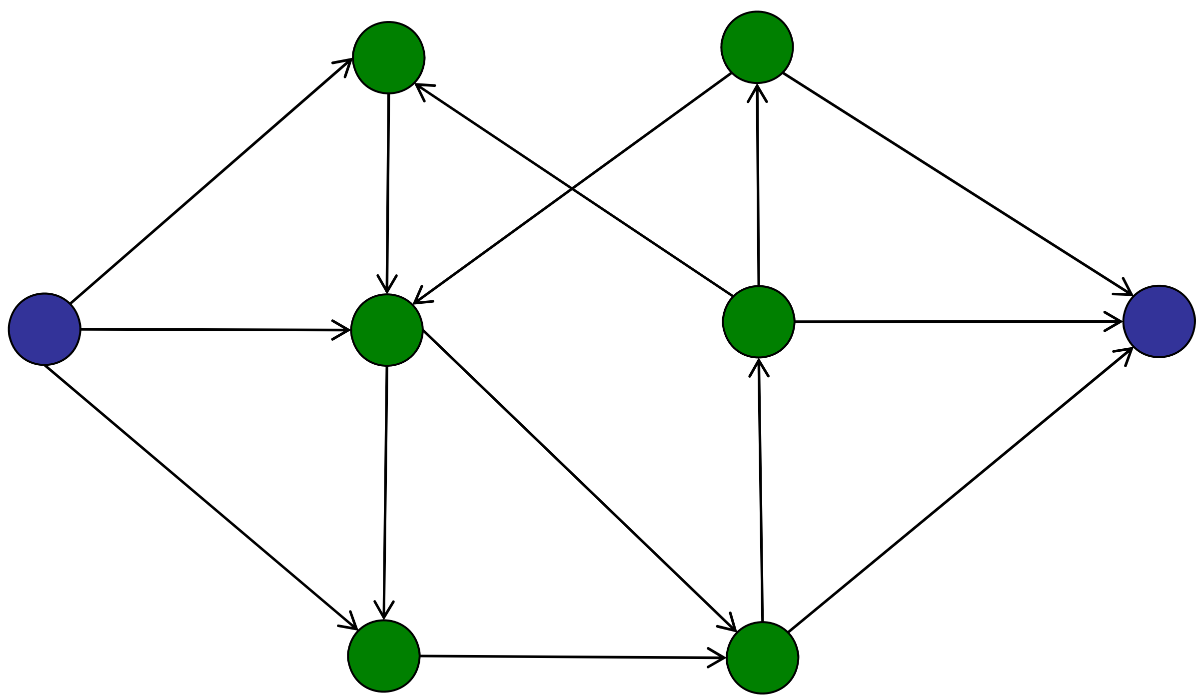
Declassified US schematic of the railway network connecting Eastern Europe and the Soviet Union. Cut capacity = 163,000 tons.

From: Harris and Ross [1955], via

Schrijver "On the history of the transportation and maximum flow problems."

Network Redundancy

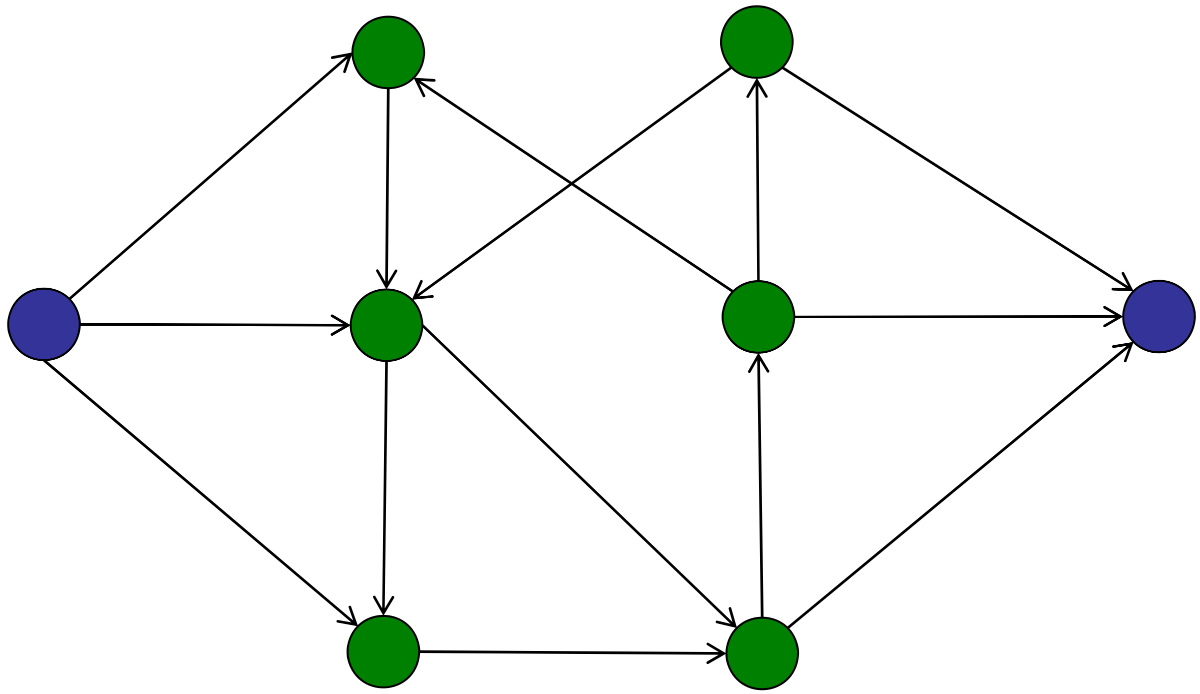
A <u>source</u> and a <u>target</u> are k-edge-connected if there are k **<u>edge-disjoint</u>** paths from the source to the target.

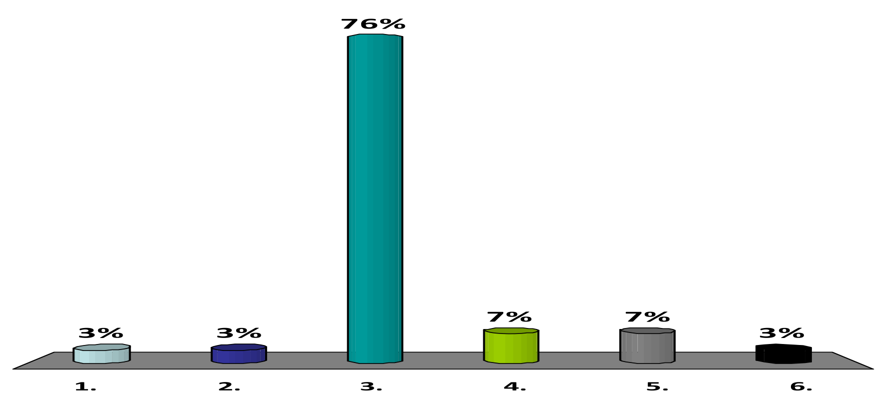


If (k-1) links fail, network is still connected.

How connected are the source and target?

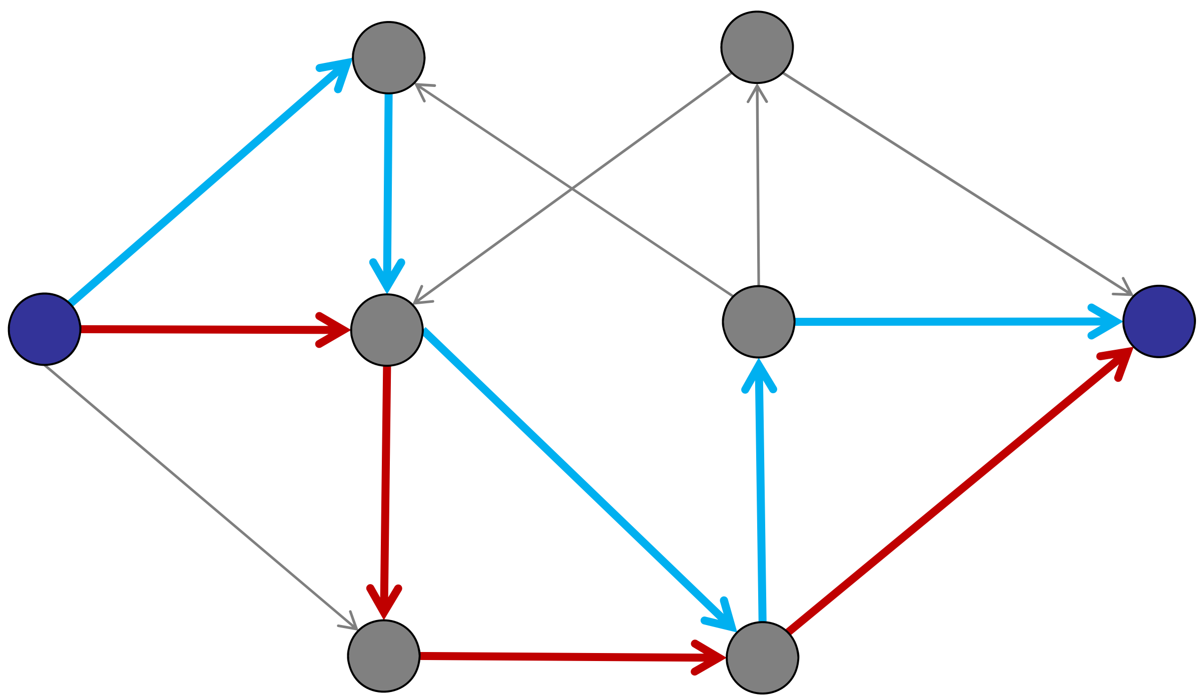
1. 0
2. 1
3. 2
4. 3
5. 4
6. I'm lazy.





Network Redundancy

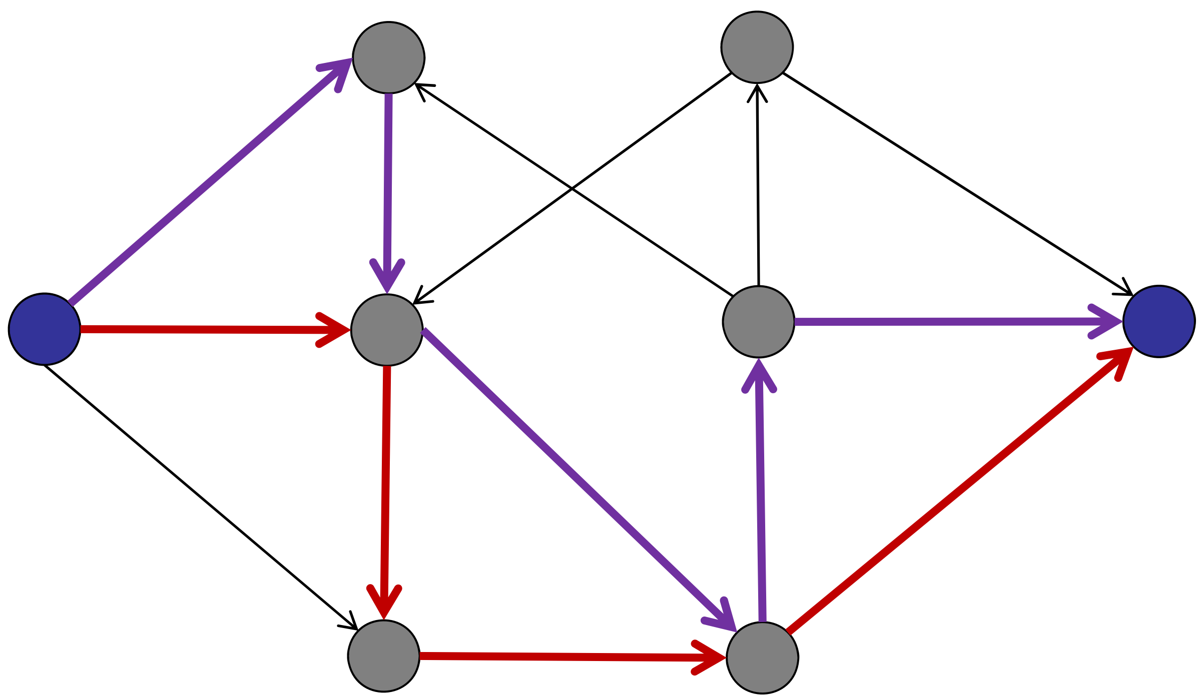
A source and a target are k-edge-connected if there are k edge-disjoint paths from the source to the target.



If 1 link fails, network is still connected.

Network Redundancy

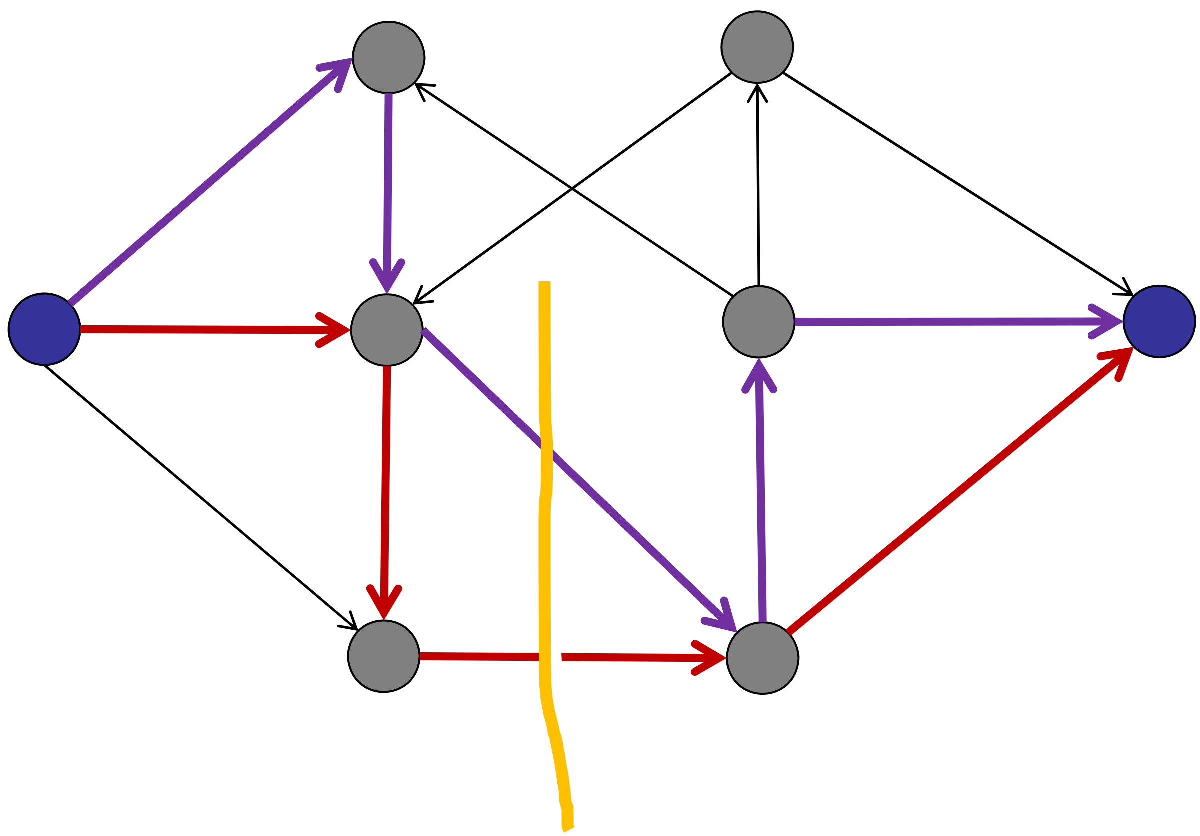
How connected is the network?

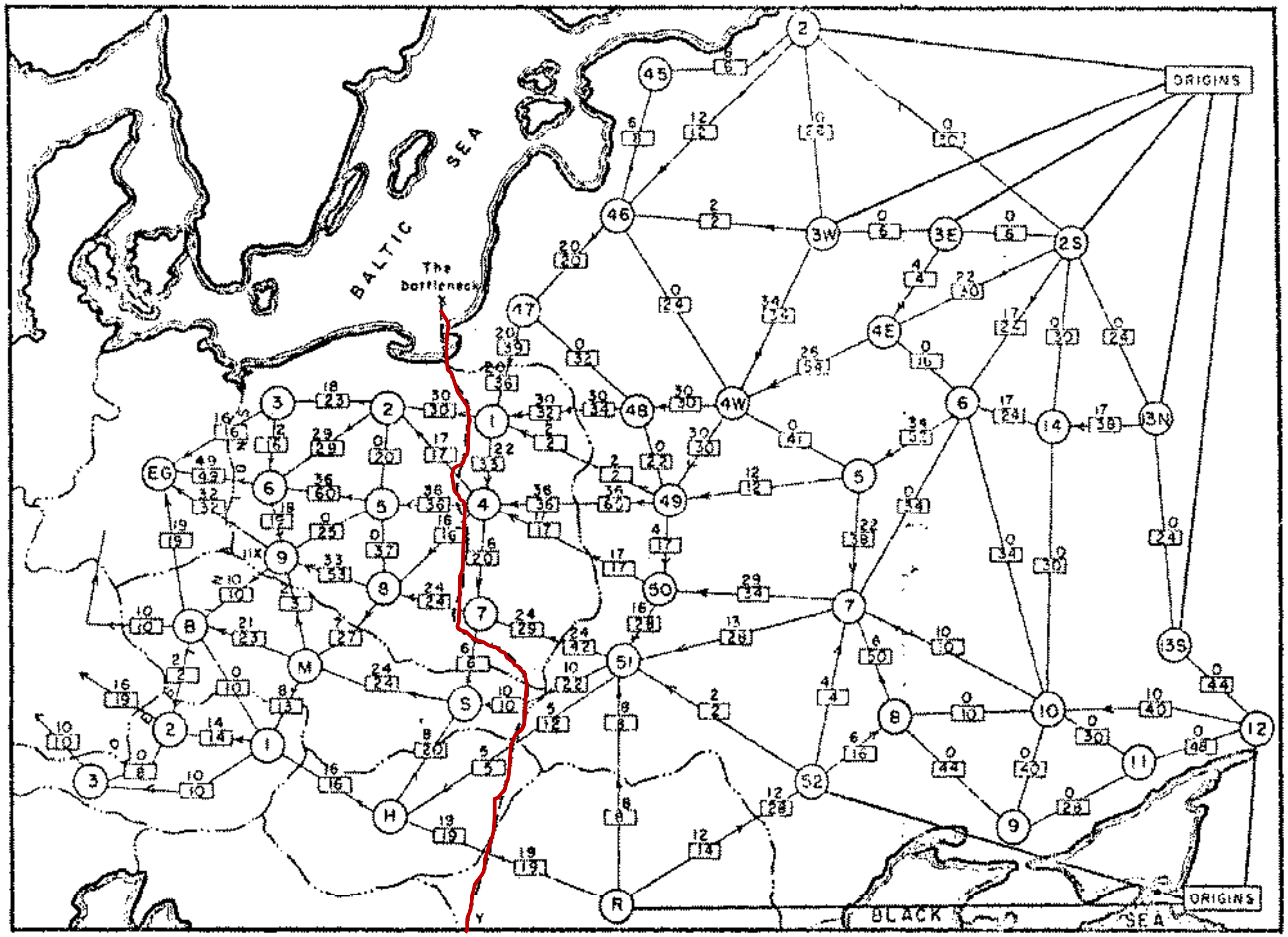


If 1 link fails, network is still connected.

Network Redundancy

Which links to cut to disrupt the network?



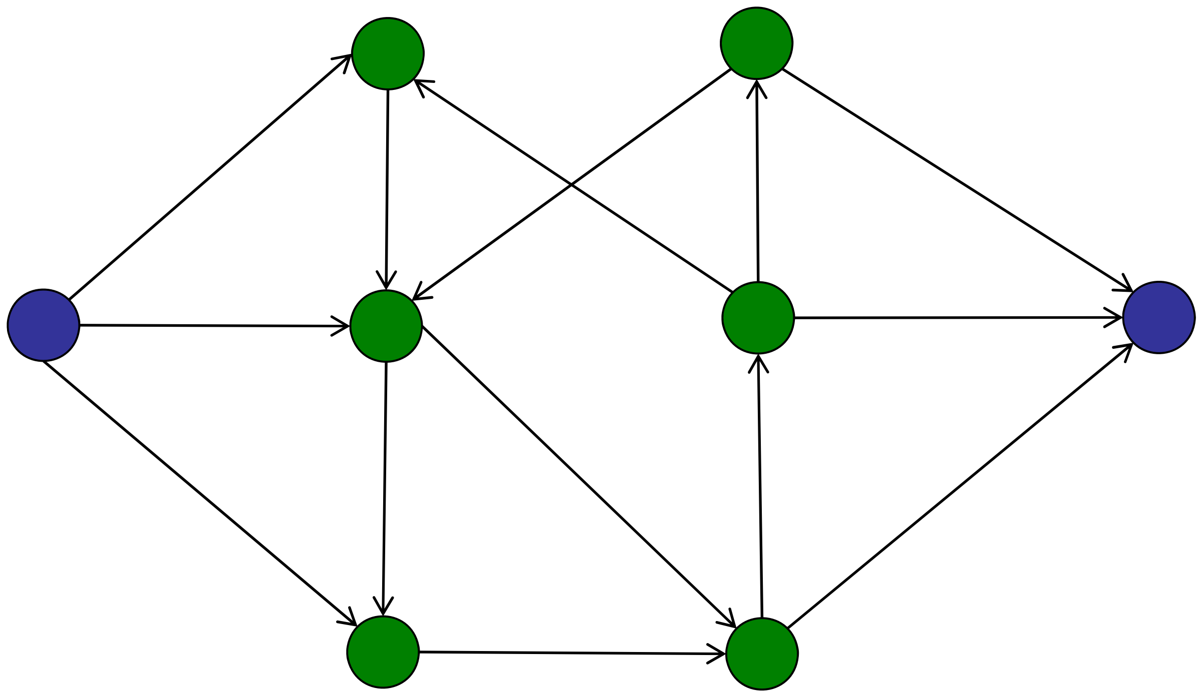


Declassified US schematic of the railway network connecting Eastern Europe and the Soviet Union. Cut capacity = 163,000 tons.

From: Harris and Ross [1955], via

Schrijver "On the history of the transportation and maximum flow problems."

How connected are the source and target?



Solution postponed (follows immediately from max-flow / min-cut).

Roadmap

Network Flows

a. Network flows defined

b. Sample problems

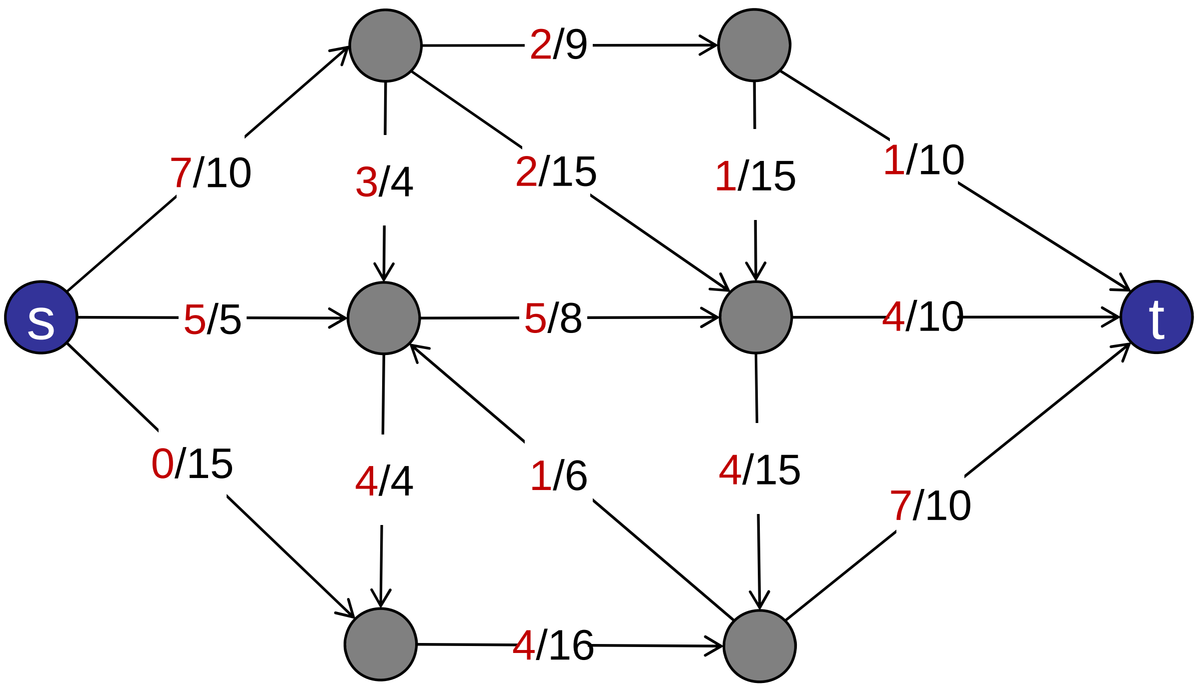
c. Ford-Fulkerson algorithm

d. Max-Flow / Min-Cut Theorem

Max Flow

Goal:

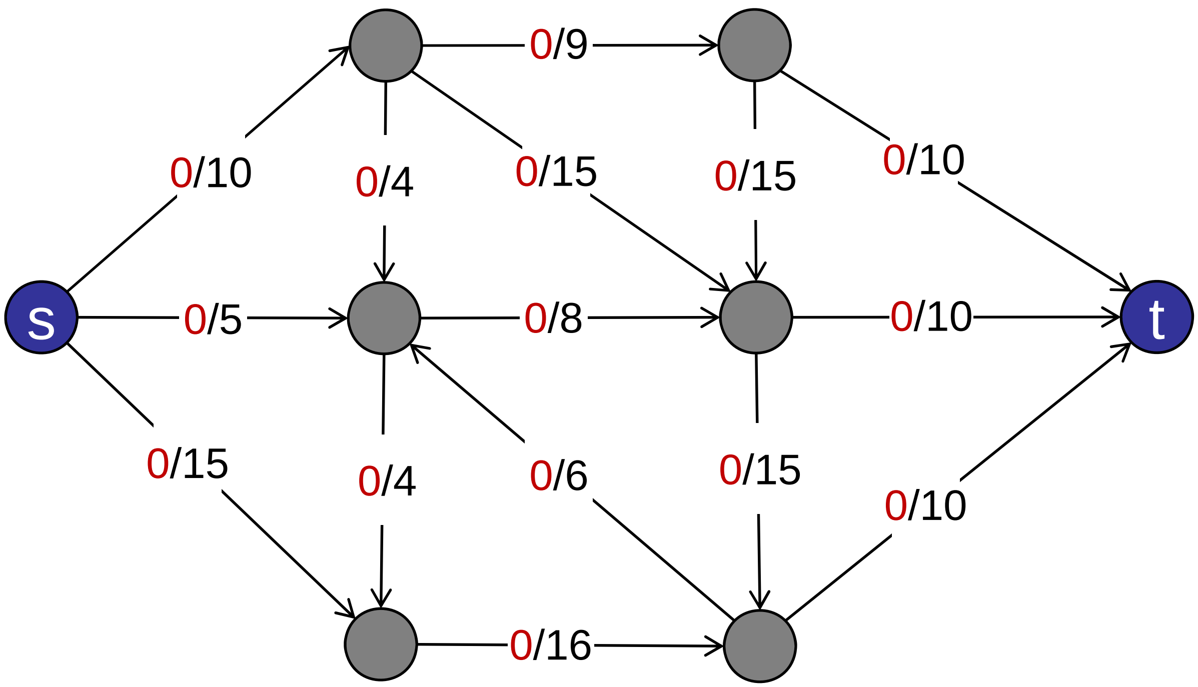
Find an st-flow with maximum value.



Ford-Fulkerson

Initially:

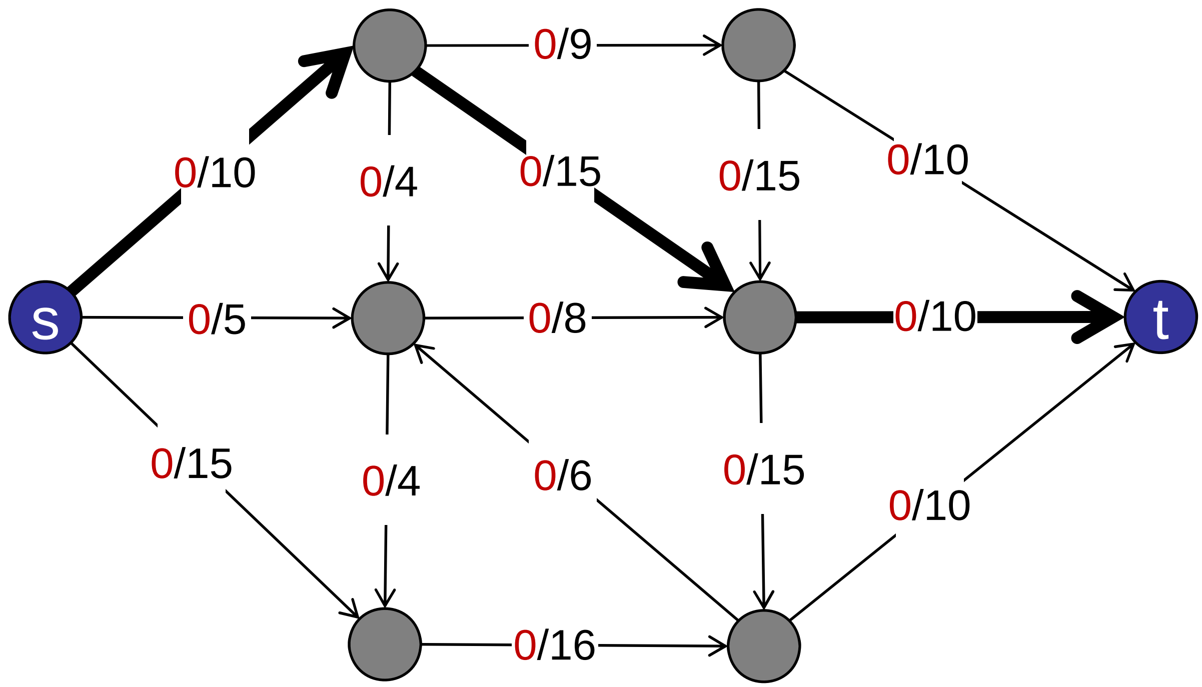
All flows are 0.



value = 0

Ford-Fulkerson

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

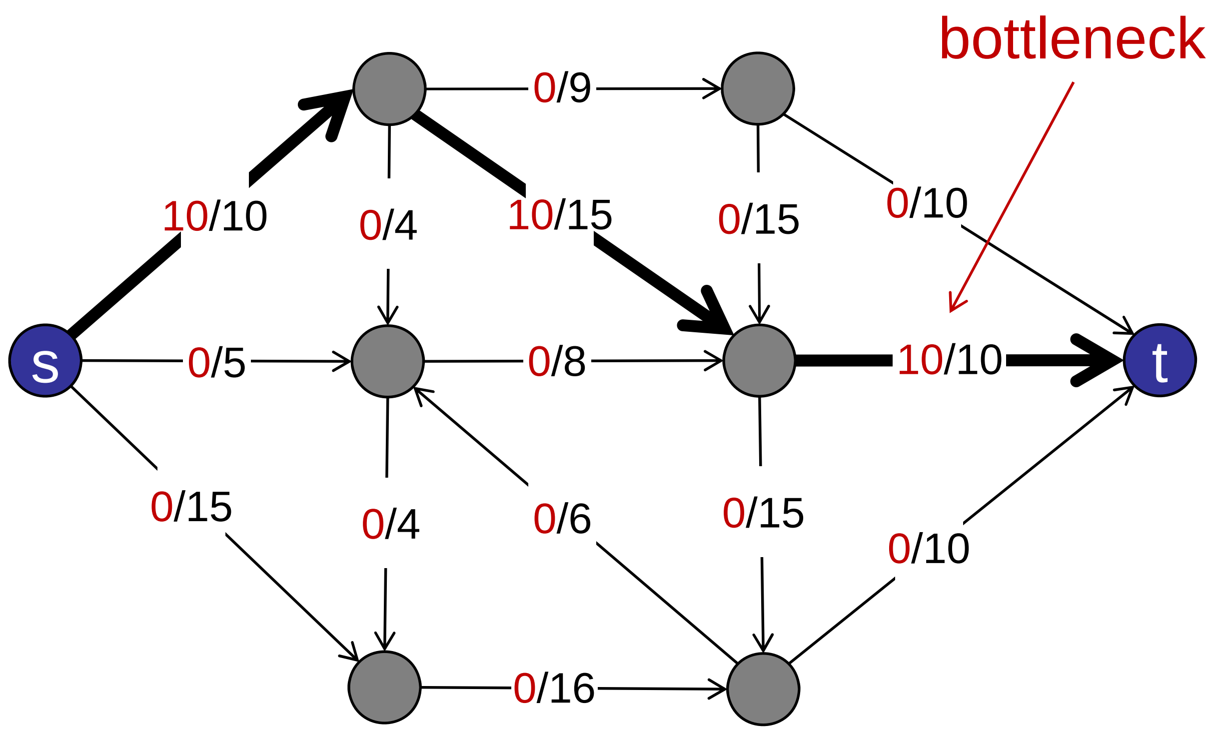


value = 0

Ford-Fulkerson

Augmenting path: directed path from s → t

- Can increase flow on all forward edges.

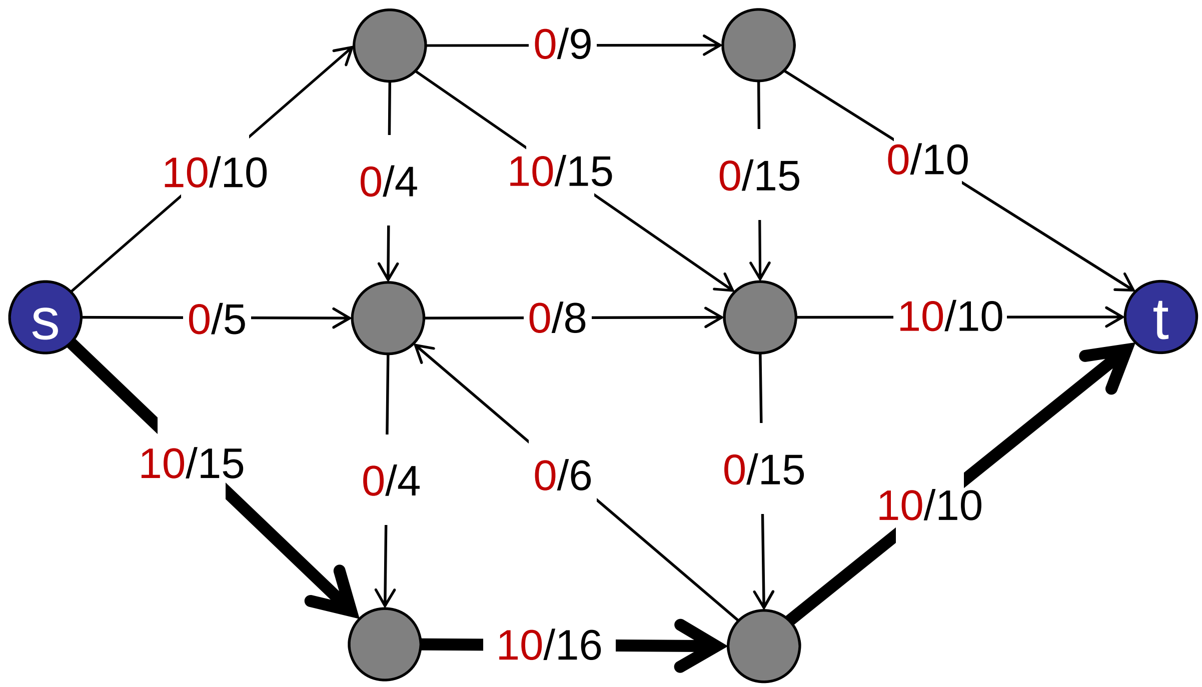


value = 0 + 10

Ford-Fulkerson

Augmenting path: directed path from s → t

- Can increase flow on all forward edges.

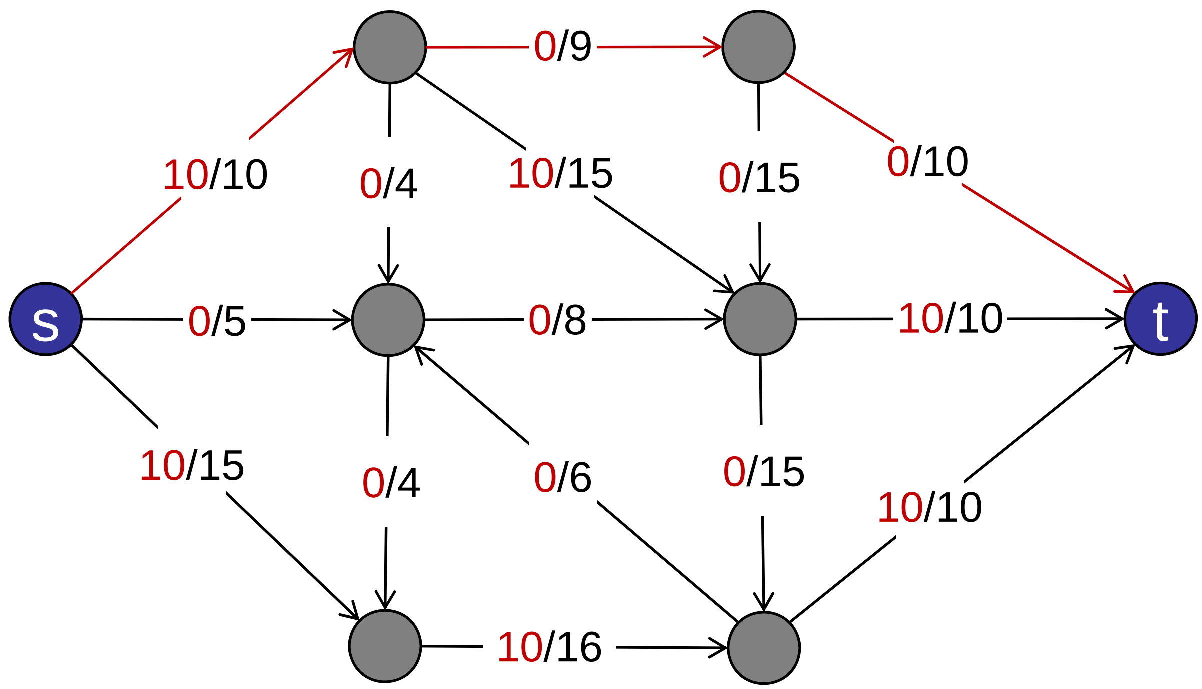


value = 0 + 10 + 10

Ford-Fulkerson

Augmenting path: directed path from s → t

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

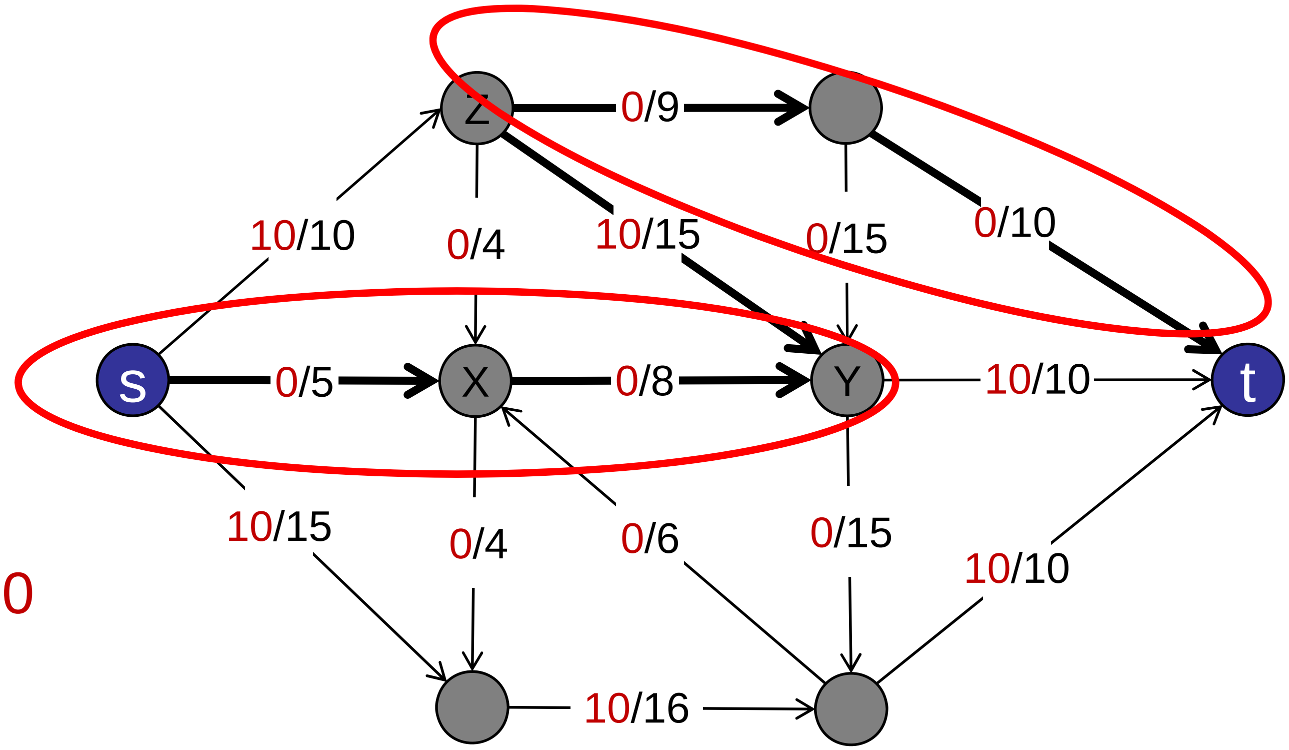


value = 0 + 10 + 10

Ford-Fulkerson

Augmenting path: directed path from s → t

– Can increase flow on all forward edges.

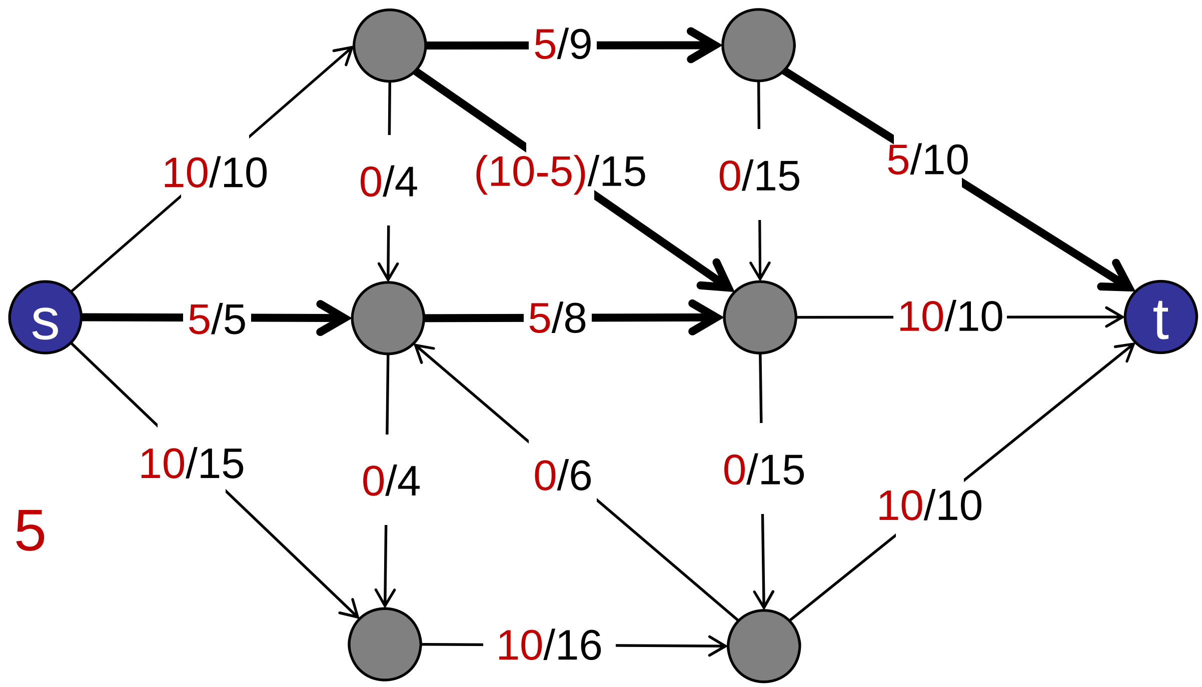


value = 0 + 10 + 10
= 20

Ford-Fulkerson

Augmenting path: Undirected path from s → t

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

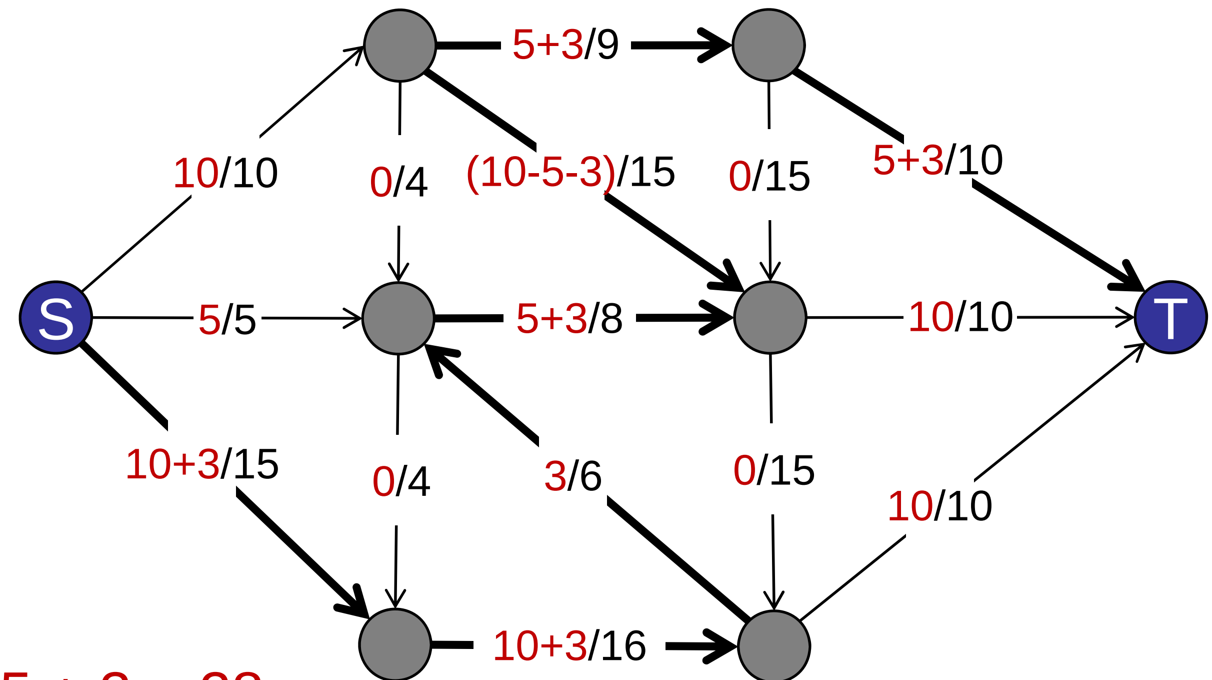


value = 0 + 10 + 10 + 5
= 25

Ford-Fulkerson

Augmenting path: Undirected path from s → t

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

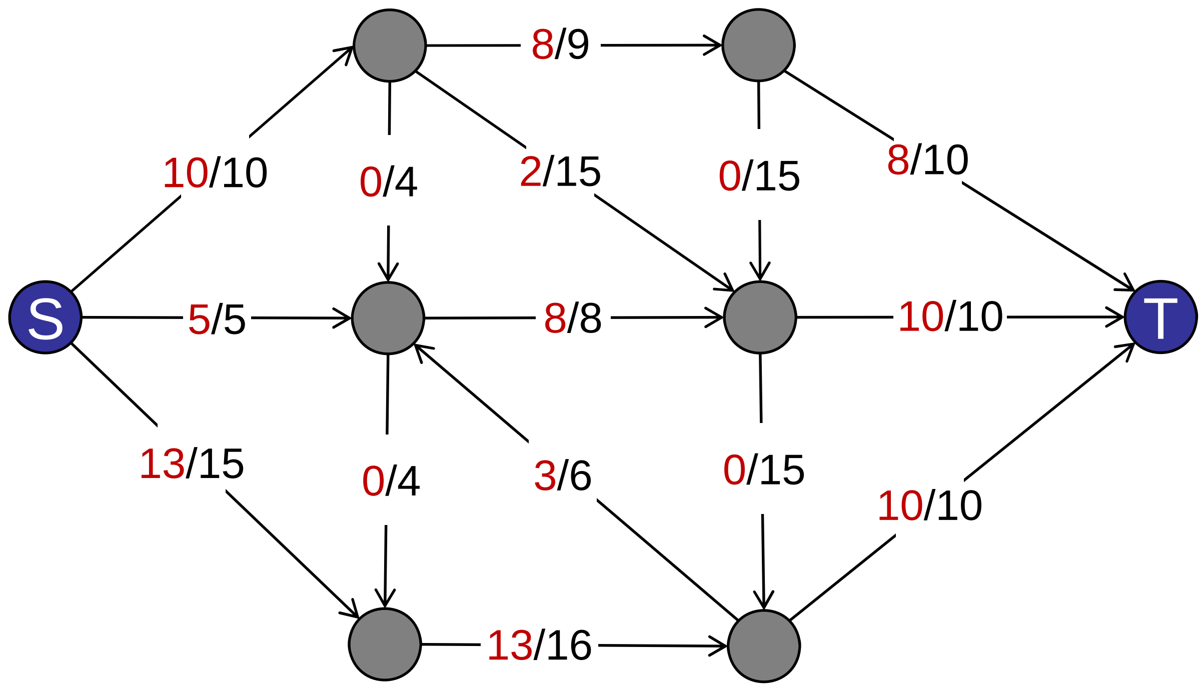


value = 0 + 10 + 10 + 5 + 3 = 28

Ford-Fulkerson

Augmenting path: Undirected path from s → t

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



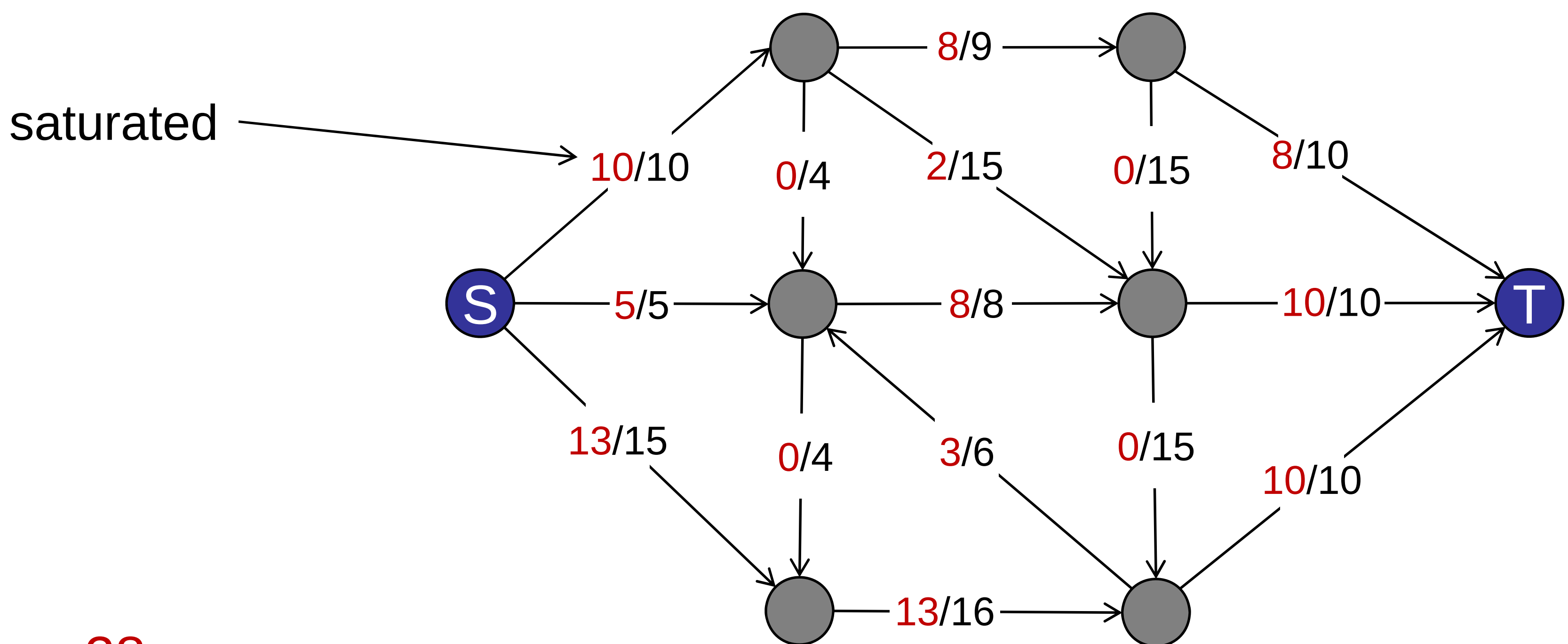
value = 28

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

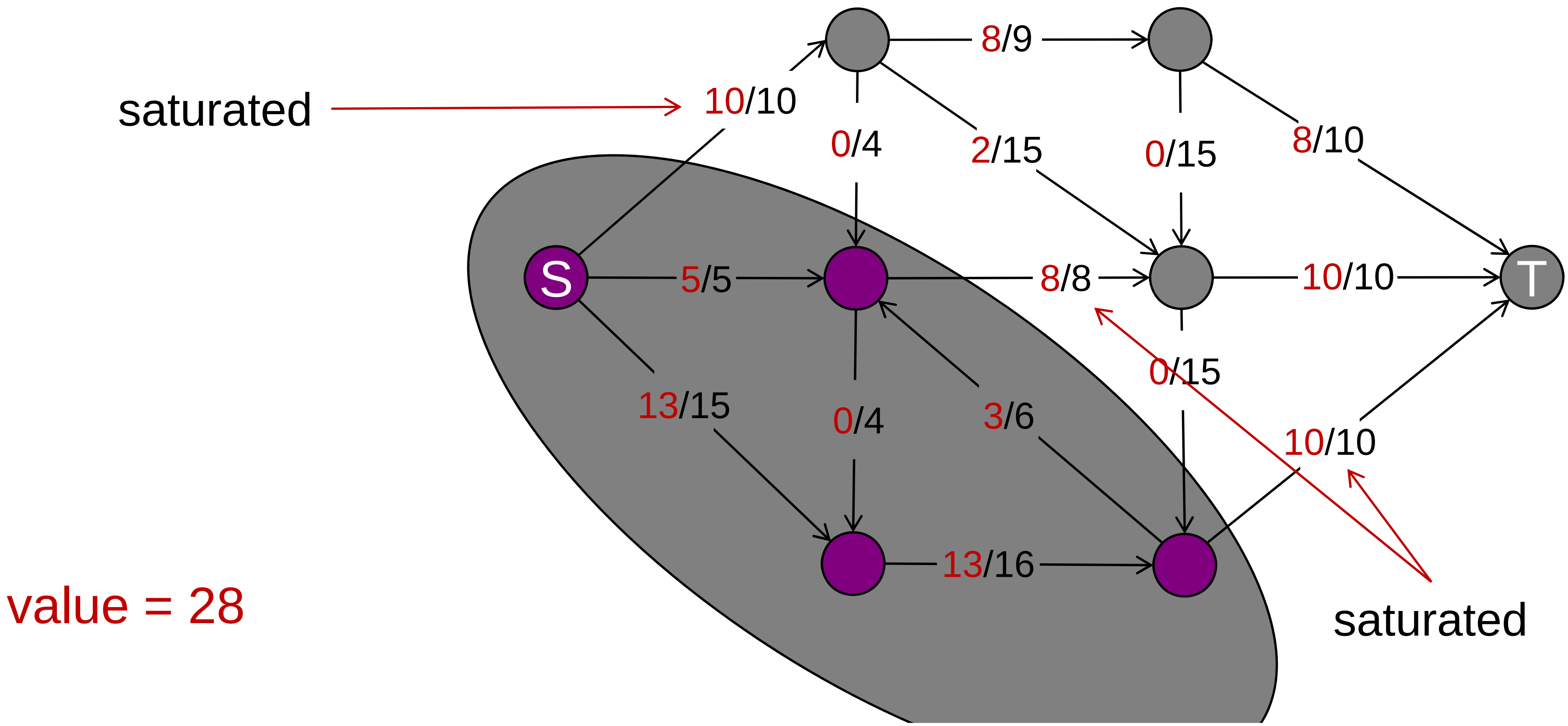


value = 28

Ford-Fulkerson

Augmenting path: Undirected path from s → t

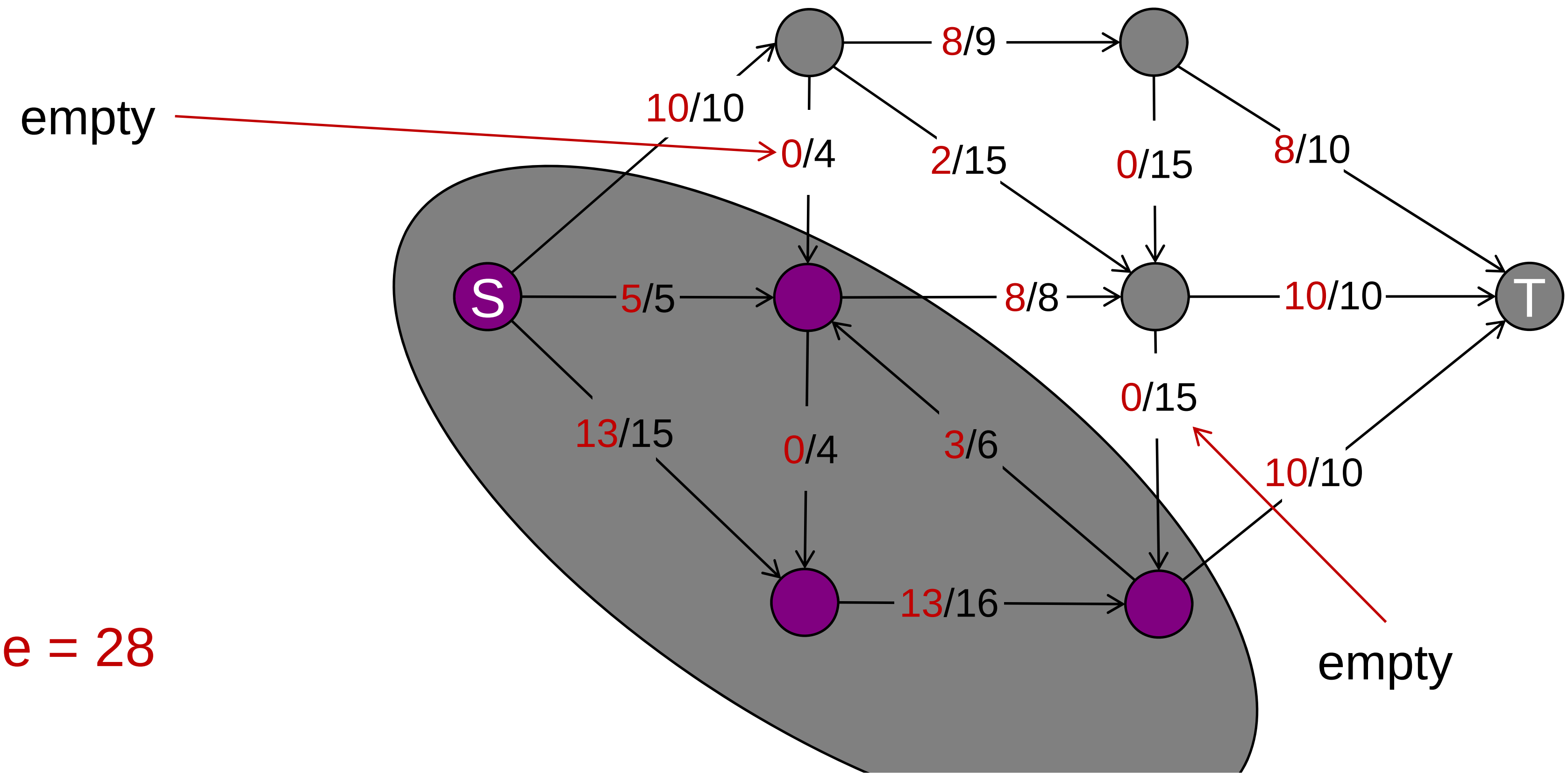
- Can increase flow on all forward edges OR
- Can decrease flow on backward edges



Ford-Fulkerson

Augmenting path: Undirected path from s → t

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

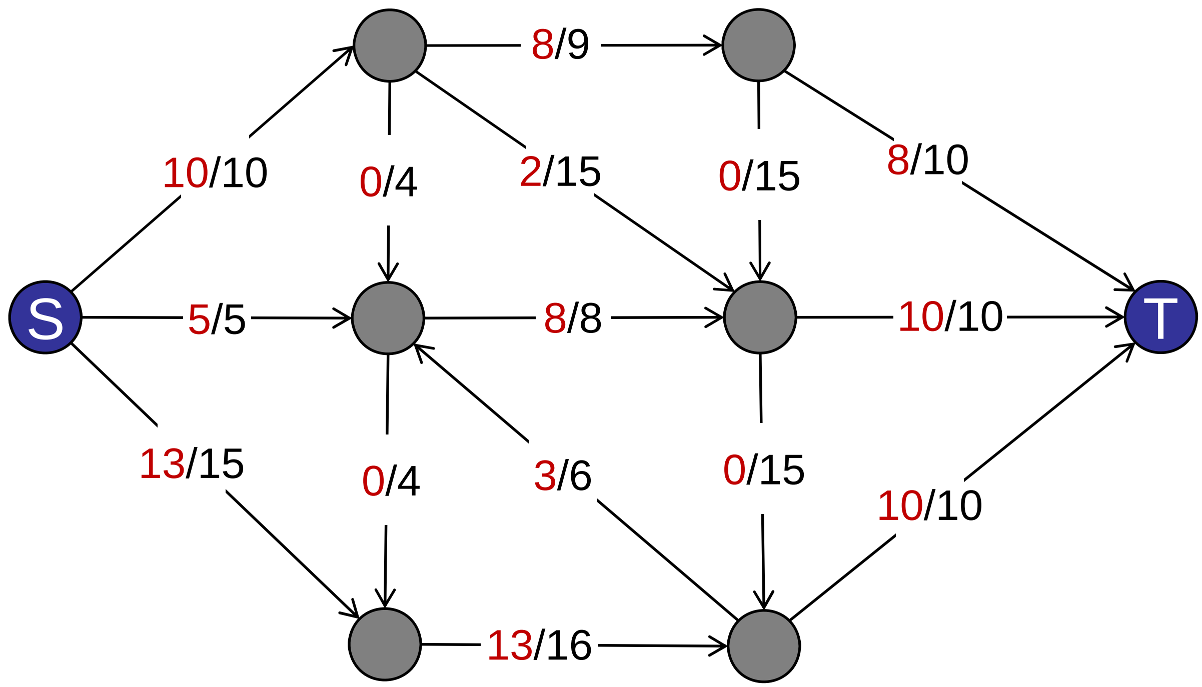


value = 28

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:

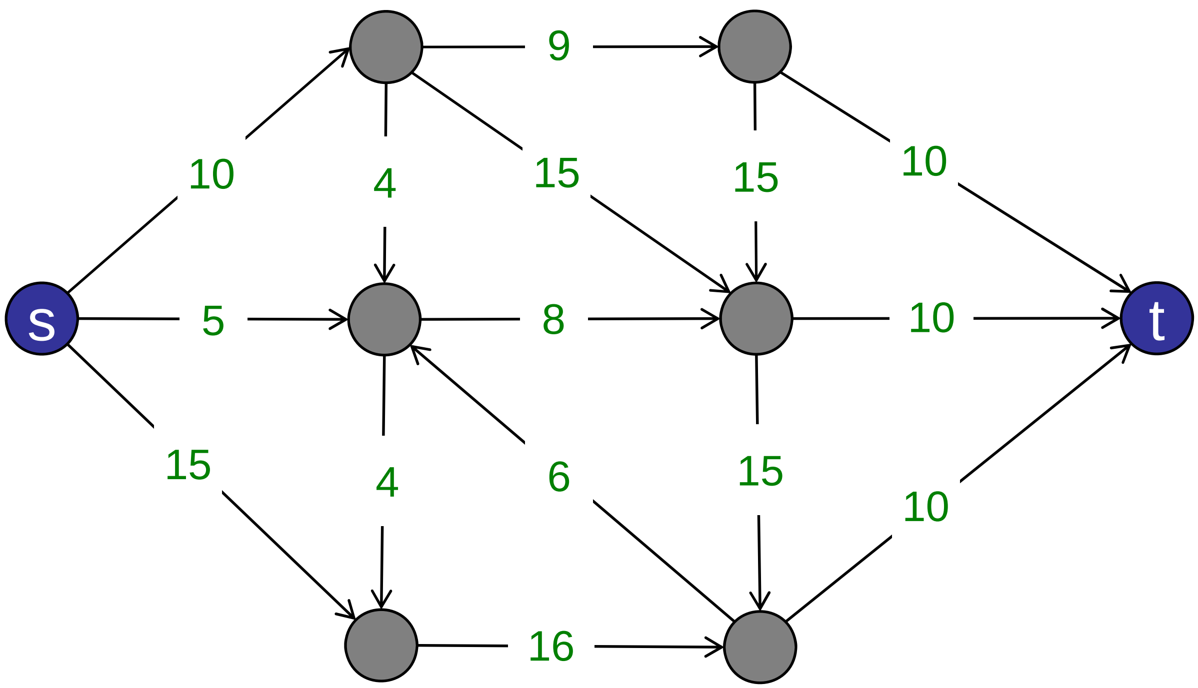
- How to find an augmenting path? The bottleneck capacity?
- Does Ford-Fulkerson always terminate? How fast?
- If it terminates, does it always find a max-flow?

Finding an Augmenting Path

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)

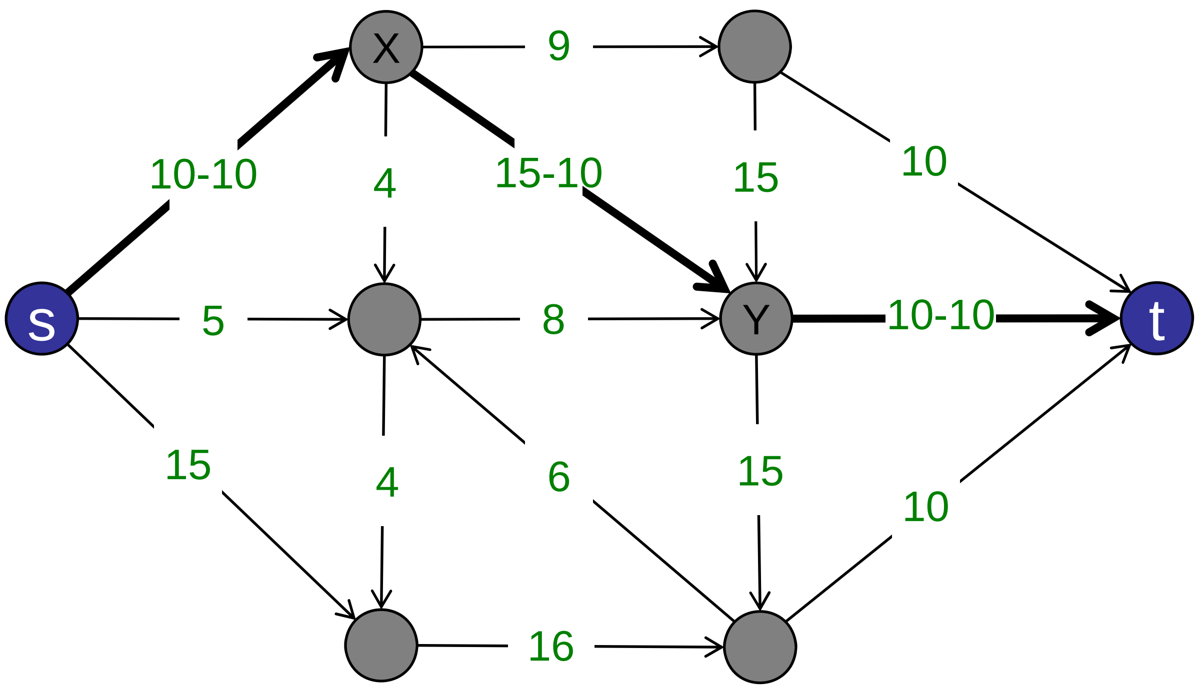


Initial graph: flow = 0 ➔ residual = capacity.

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)

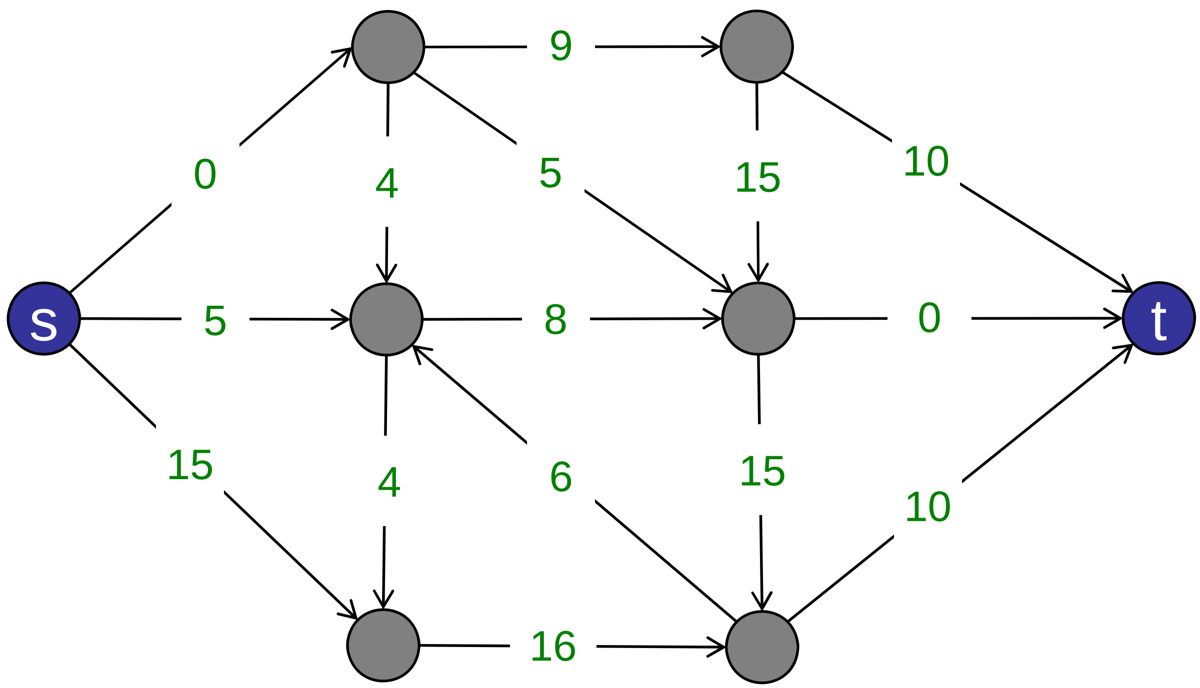


Step 1: augmenting path of flow 10.

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

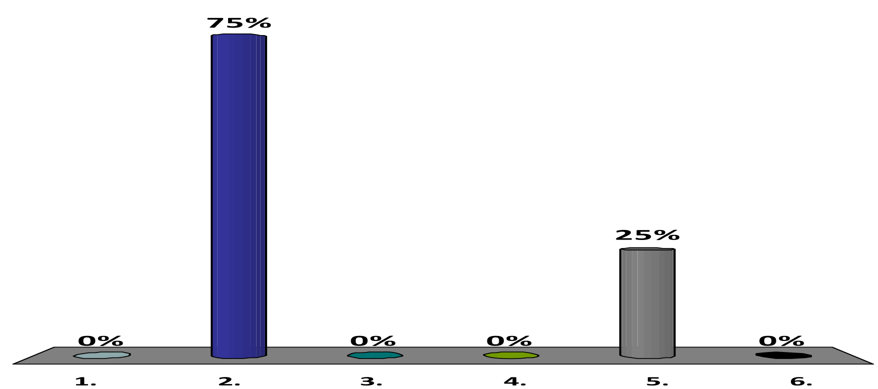
residual(e) = capacity(e) – flow(e)



After augmenting path of flow 10.

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

1. BFS
2. DFS
3. Bellman-Ford
4. Dijkstra's
5. Floyd-Warshall
6. I have no idea.



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

For now:

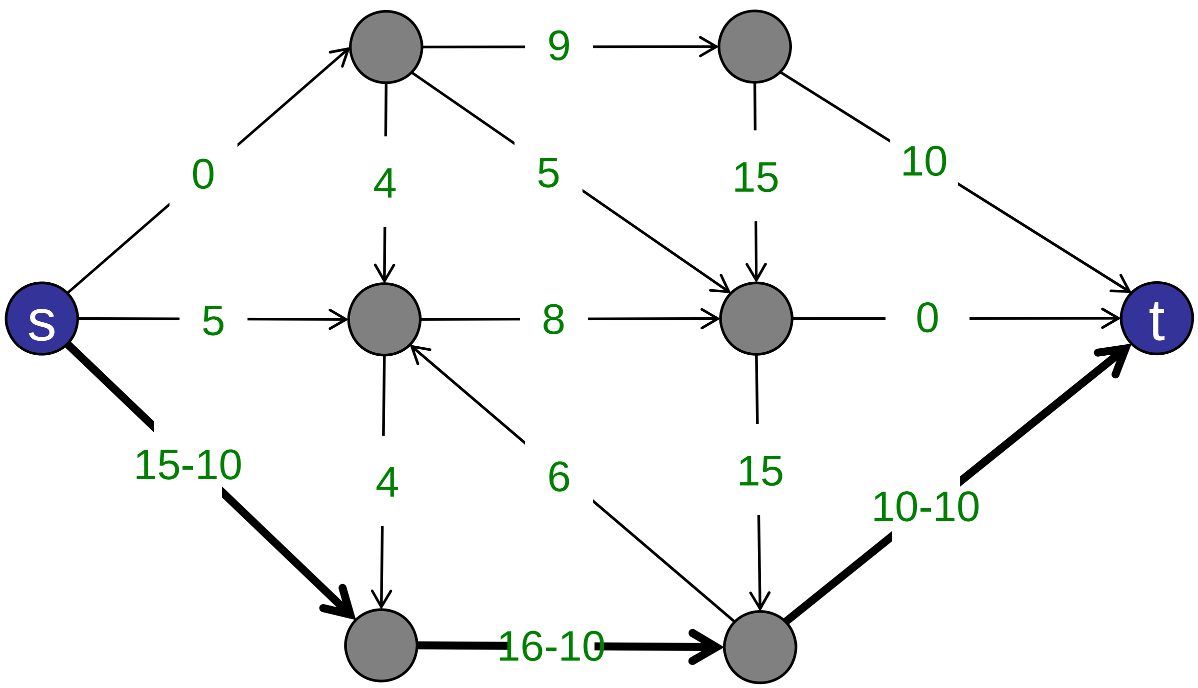
<u>Any</u> graph search will do (BFS, DFS, etc.)

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)

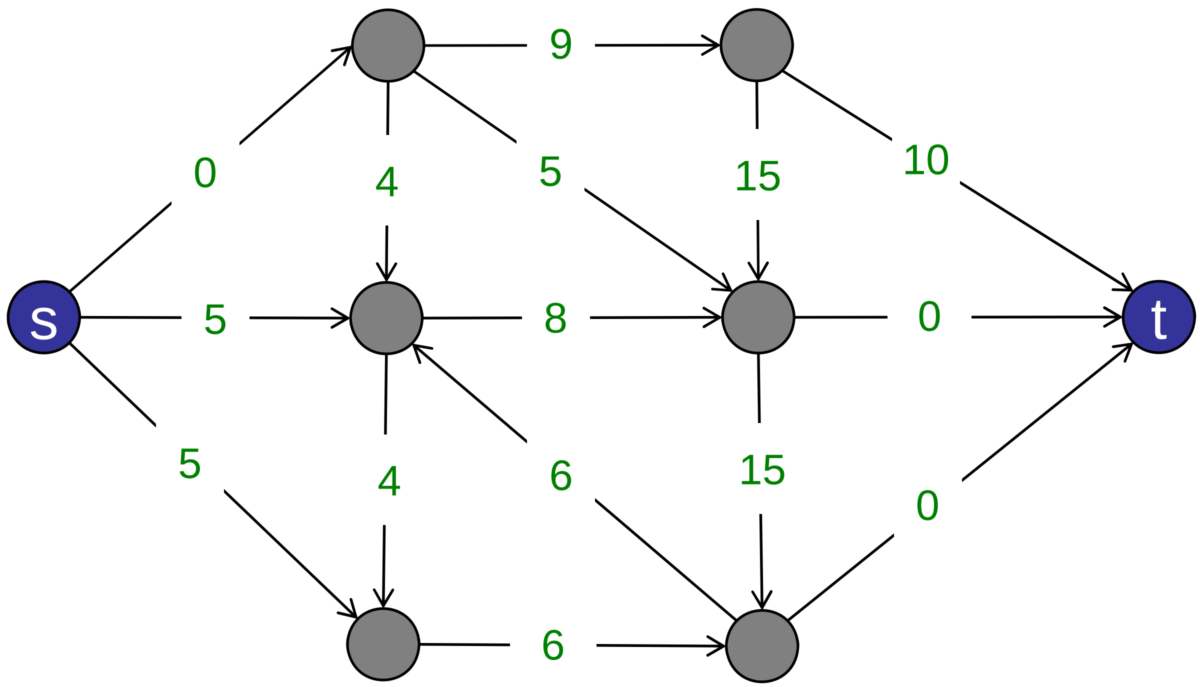


Step 2: augmenting path of flow 10.

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)



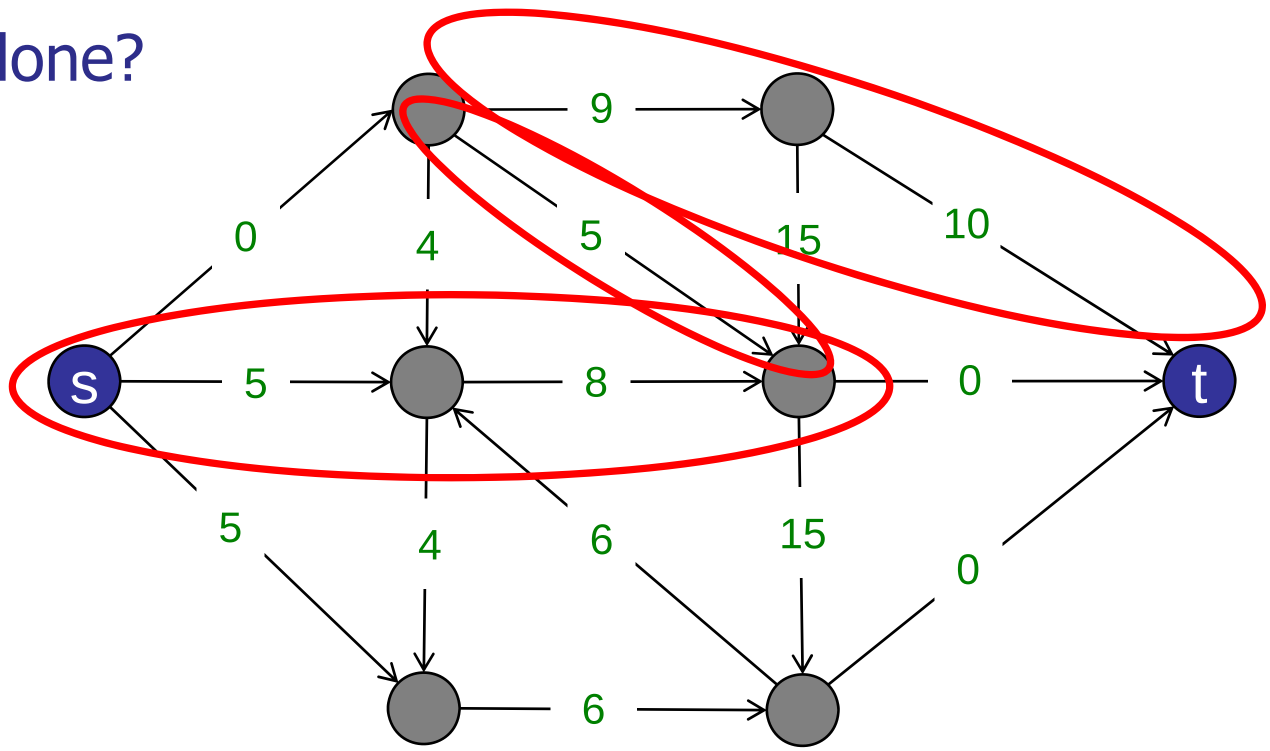
After step 2: augmenting path of flow 10.

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)

Are we done?

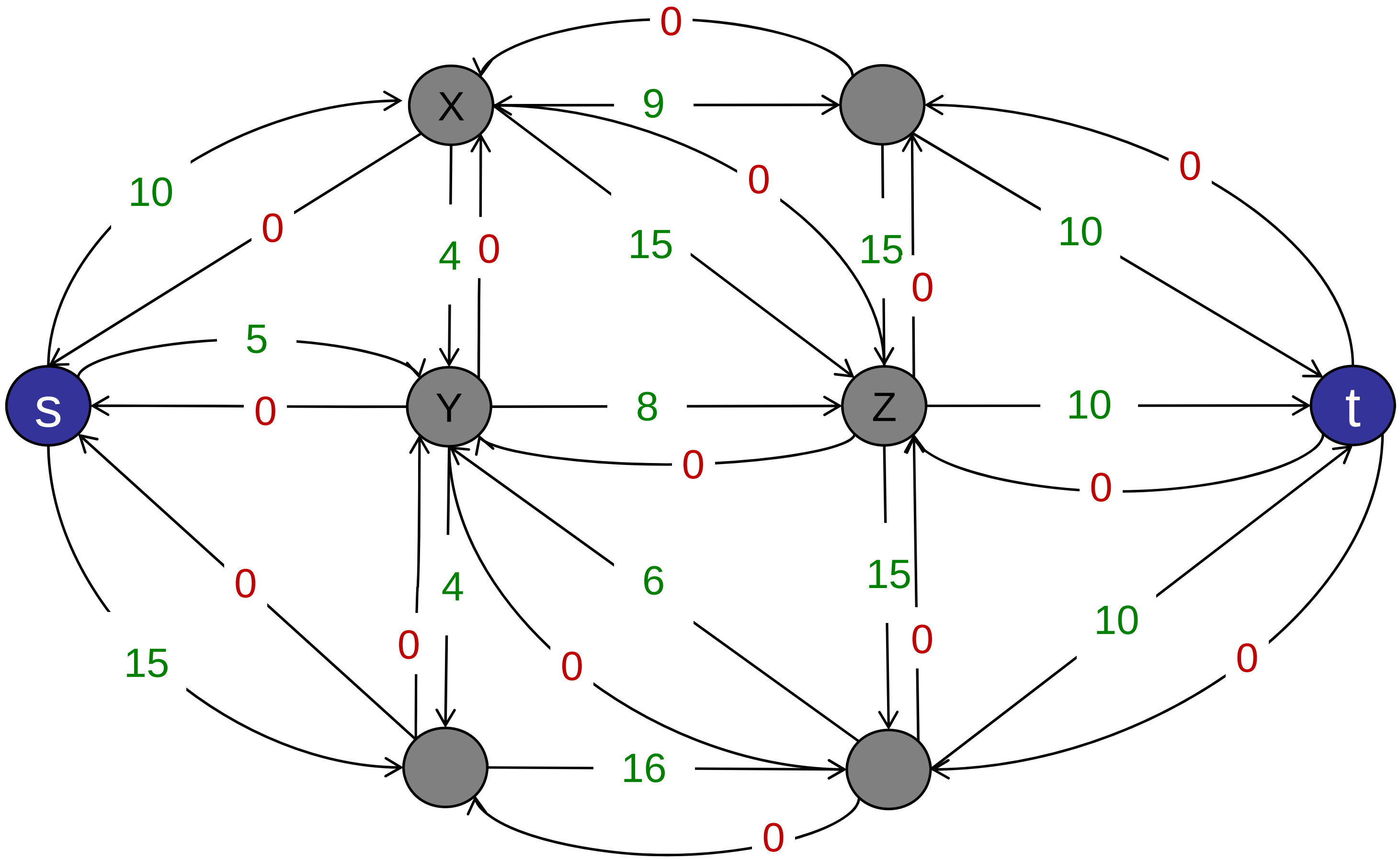


After step 2: augmenting path of flow 10.

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)

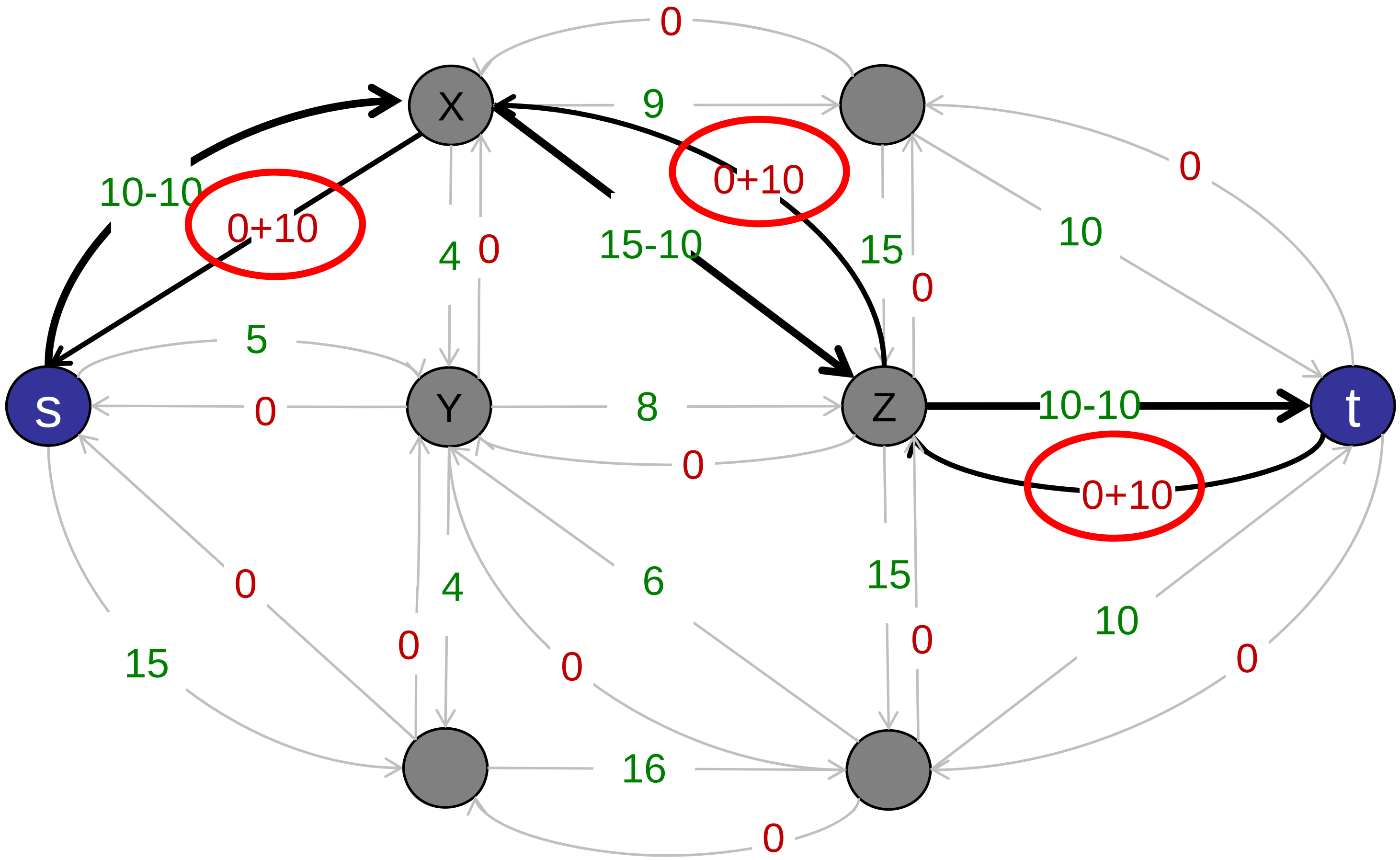


Add edges in both direction.

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)

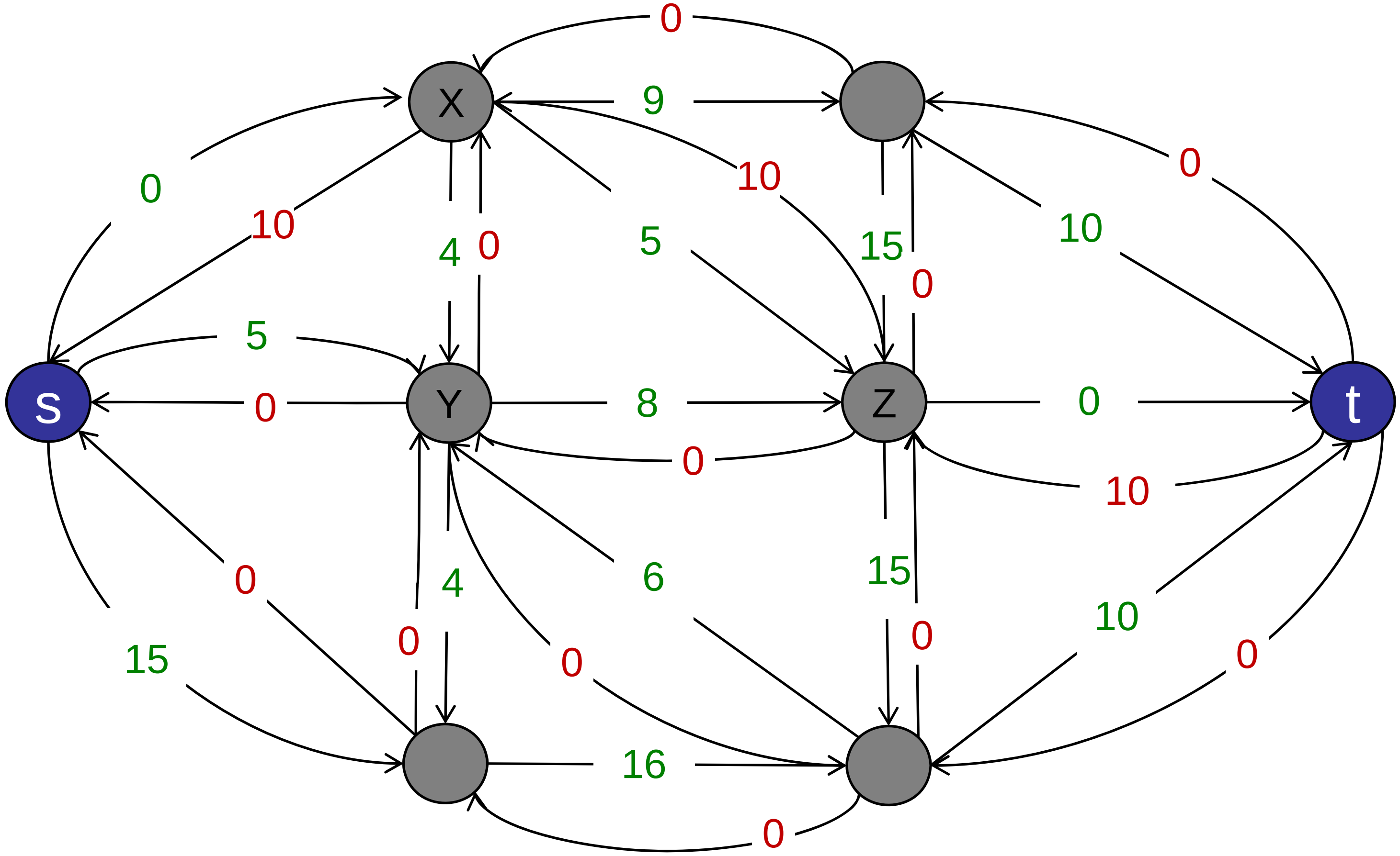


Augmenting path: residual flow in the other direction.

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)

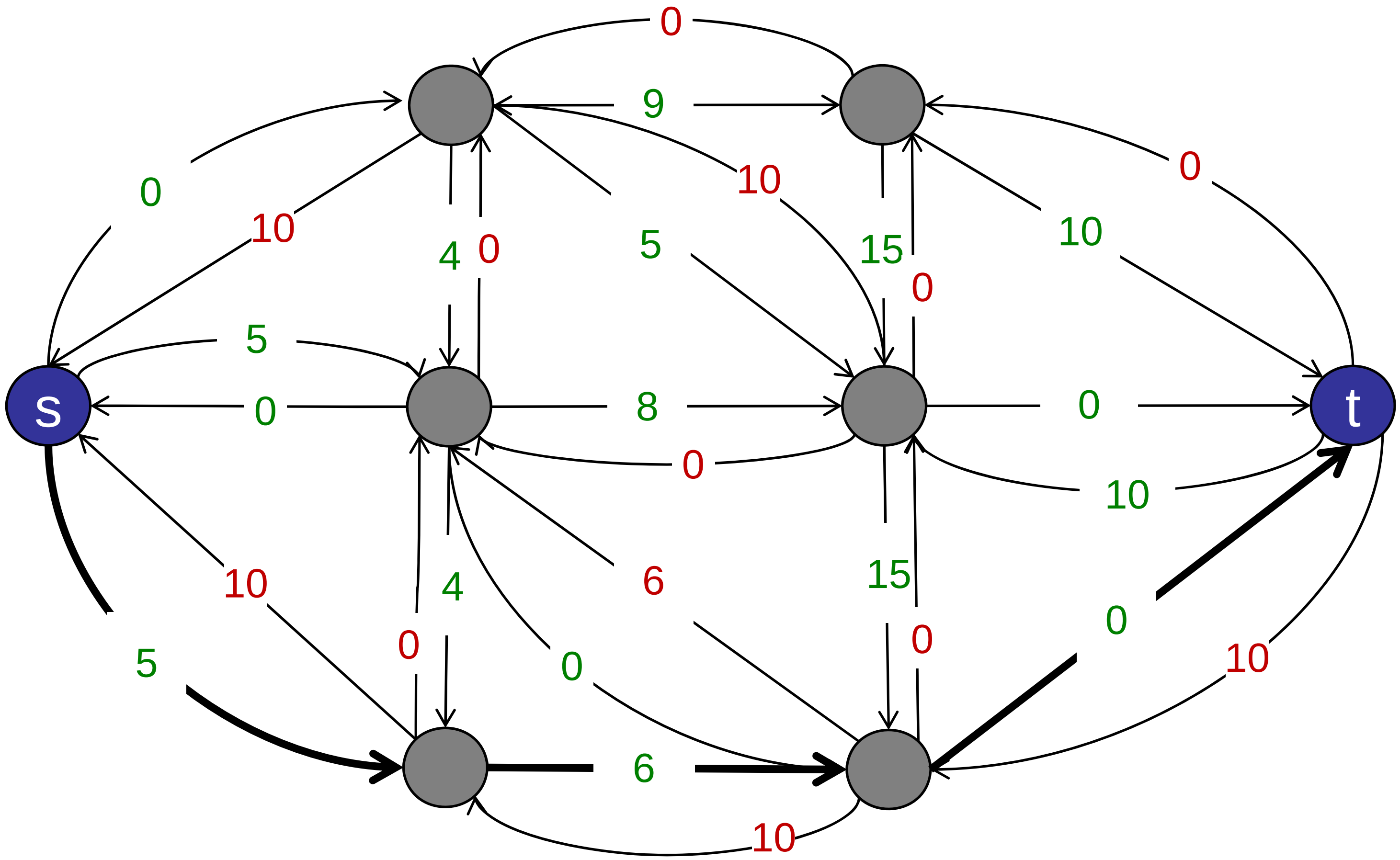


Augmenting path: residual flow in the other direction.

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)

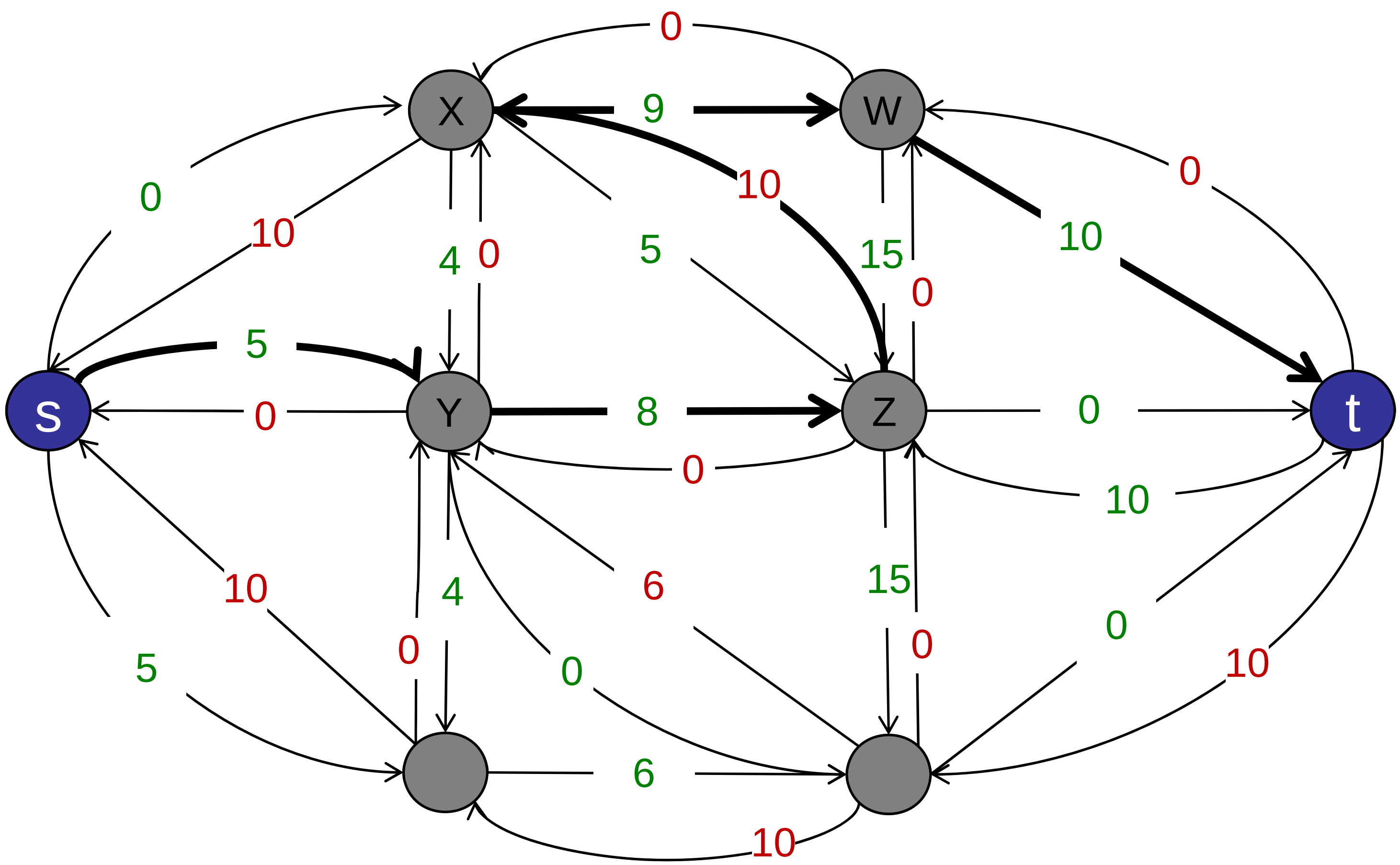


Augmenting path: residual flow in the other direction.

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)

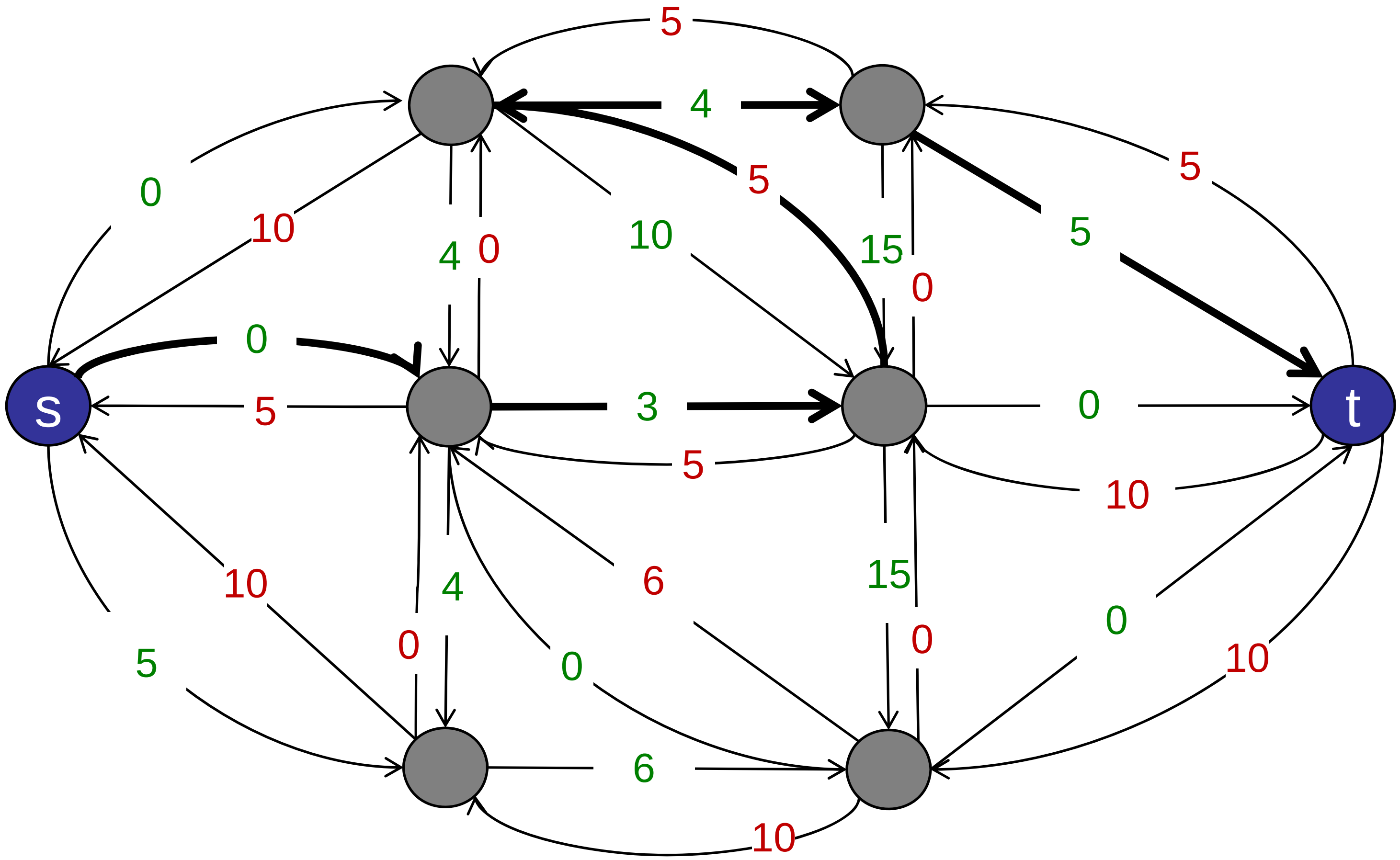


Augmenting path: residual flow in the other direction.

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)

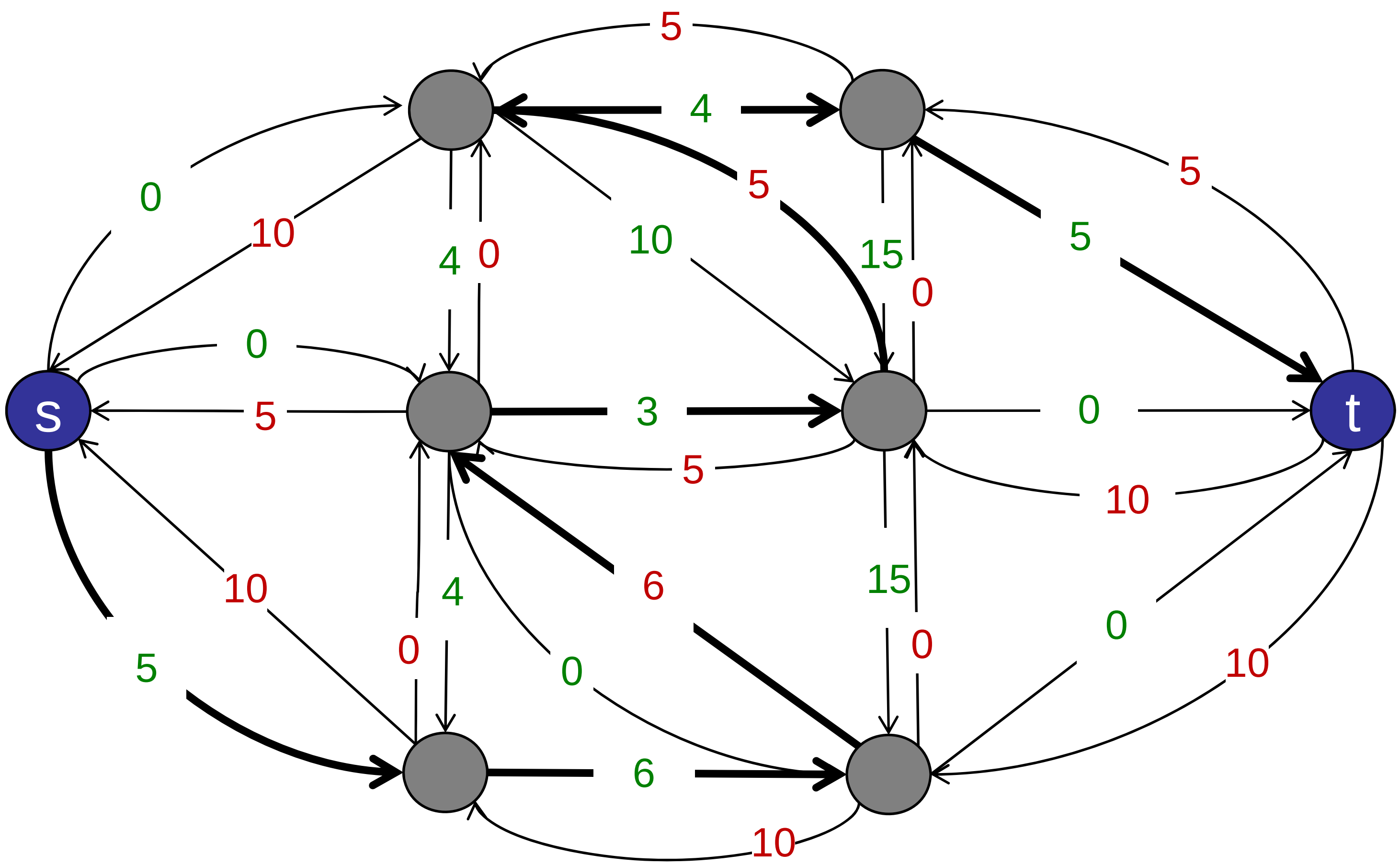


Augmenting path: residual flow in the other direction.

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)

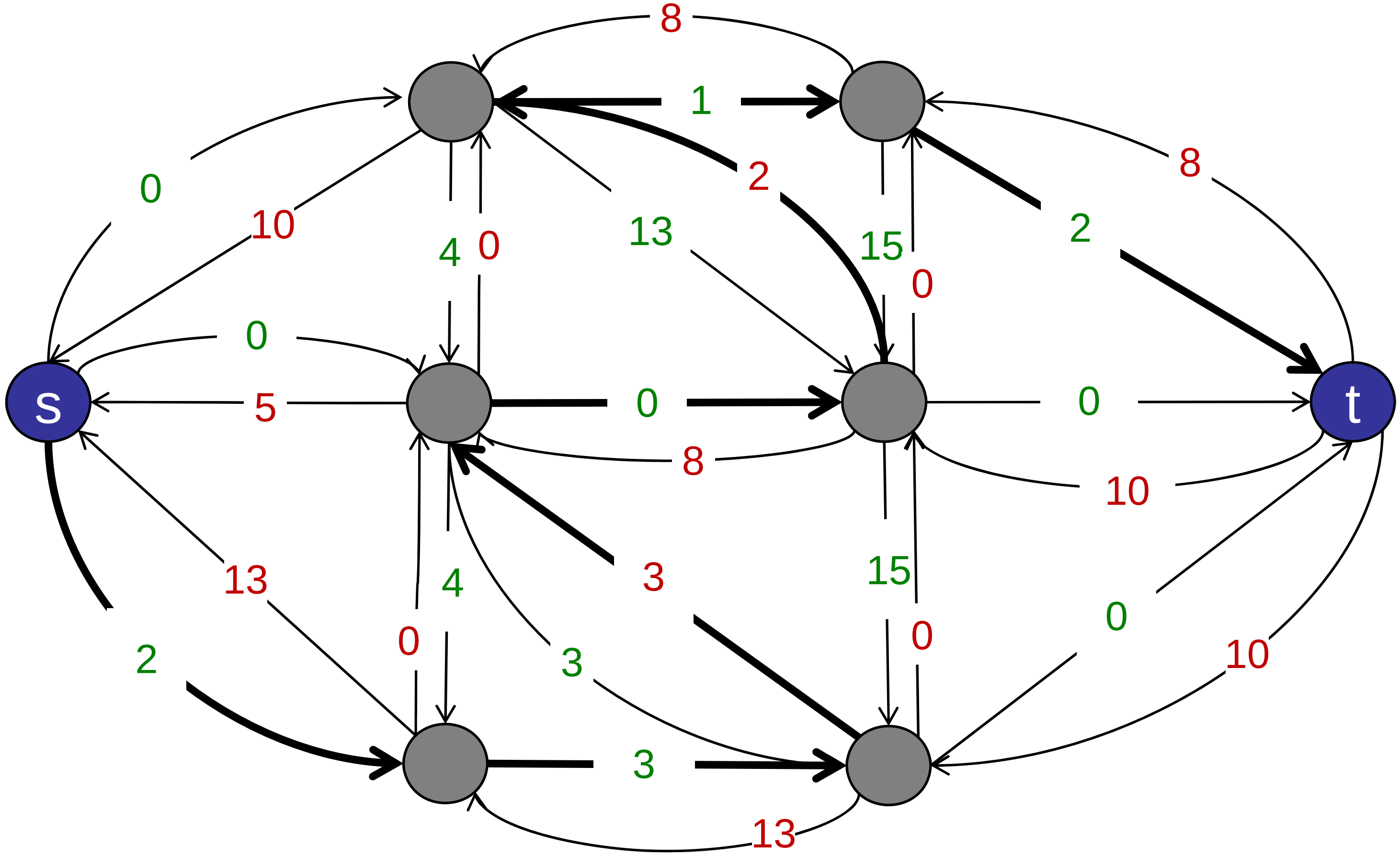


Augmenting path: residual flow in the other direction.

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)

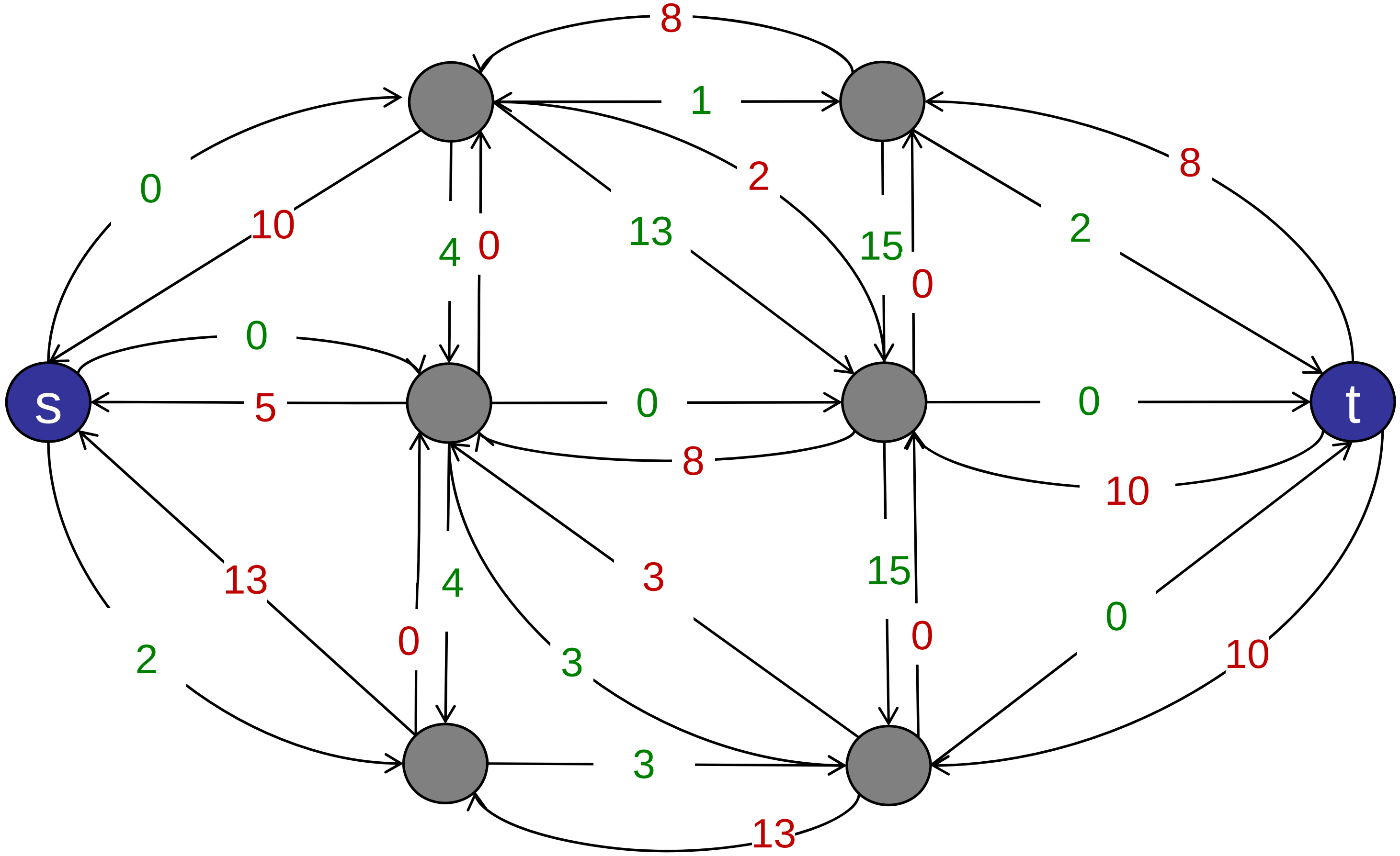


Augmenting path: residual flow in the other direction.

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)

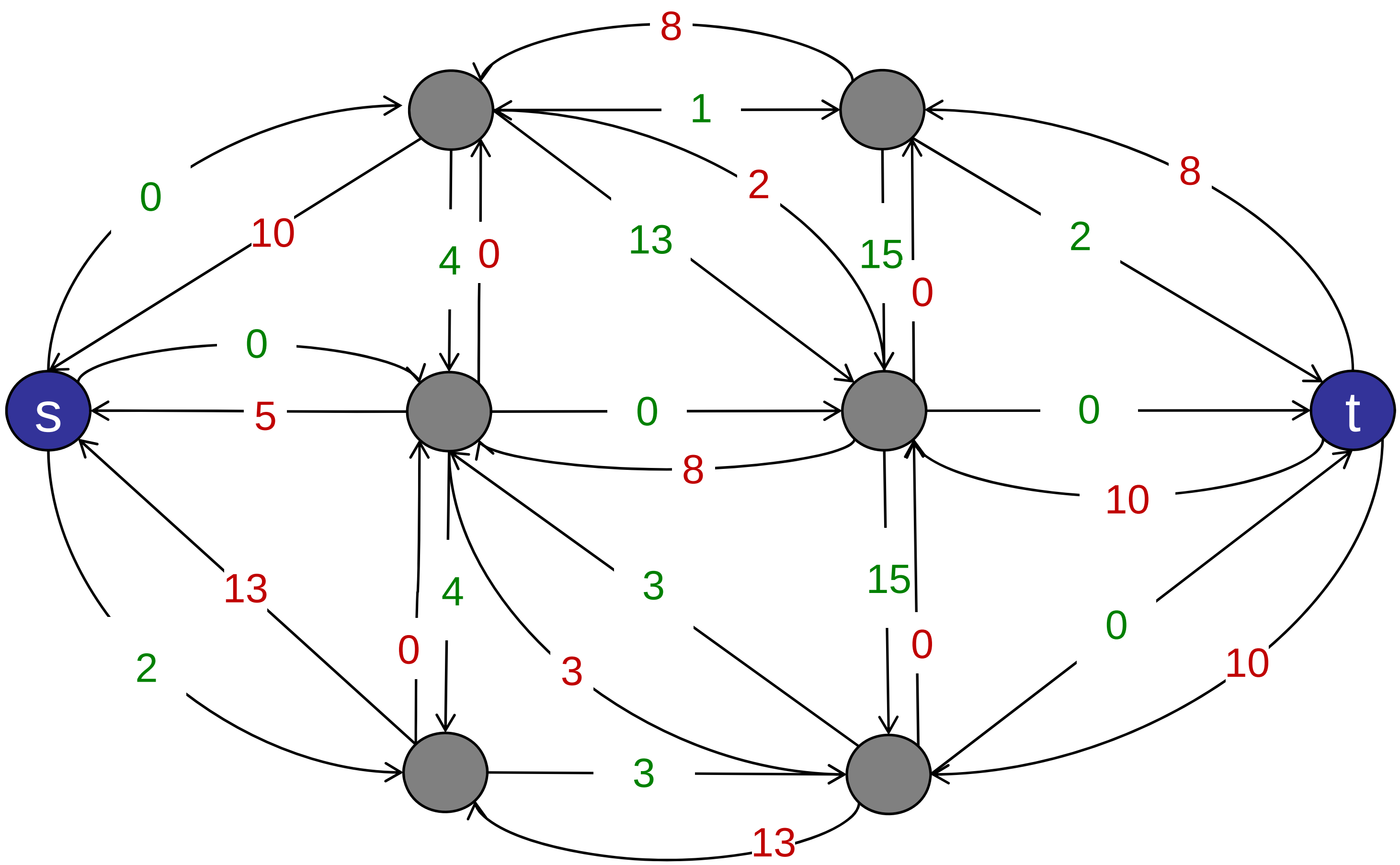


Augmenting path: residual flow in the other direction.

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.

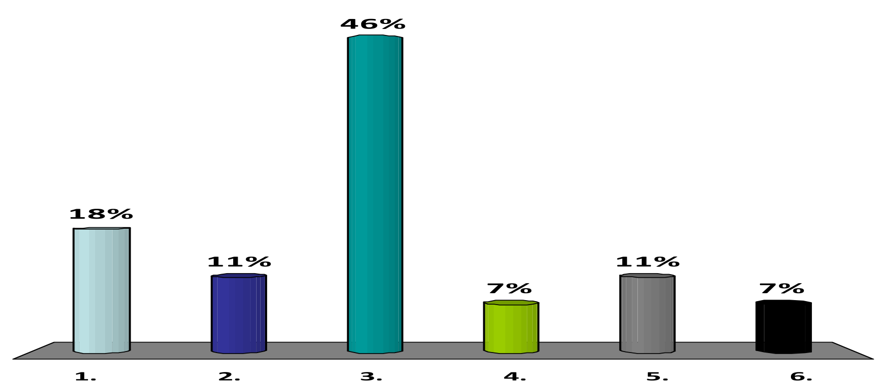
While there exists an augmenting path:

- 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.

How best to find the bottleneck capacity on the augmenting path?

1. TopoSort traversal
2. Ordered traversal
3. DFS traversal
4. Bellman-Ford
5. Dijkstra's
6. I have no idea.



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? The bottleneck capacity?
- – Does Ford-Fulkerson always terminate? How fast?
- – If it terminates, does it always find a max-flow?

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: FF may NOT terminate if capacities are irrational.

- Runs forever.
- Never converges to maximum flow.

Performance of Ford-Fulkerson

What is the cost of finding and applying an augmenting path?

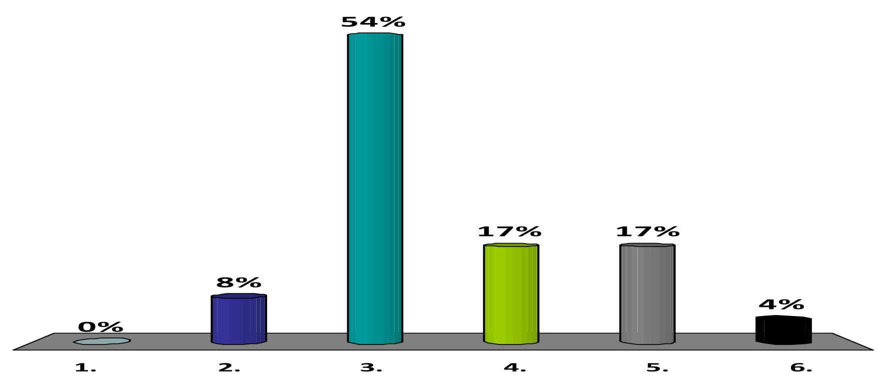
1. O(1)
2. O(V)
3. O(E)
4. O(E log V)
5. O(VE)
6. I have no idea.

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.**



Performance of Ford-Fulkerson

How many times do we find an augmenting path?

- Assume edge capacities are integers.
- Assume maximum flow = F.

In ever iteration:

- Each augmentation increases flow by at least 1.
- Each augmentation costs: O(E)

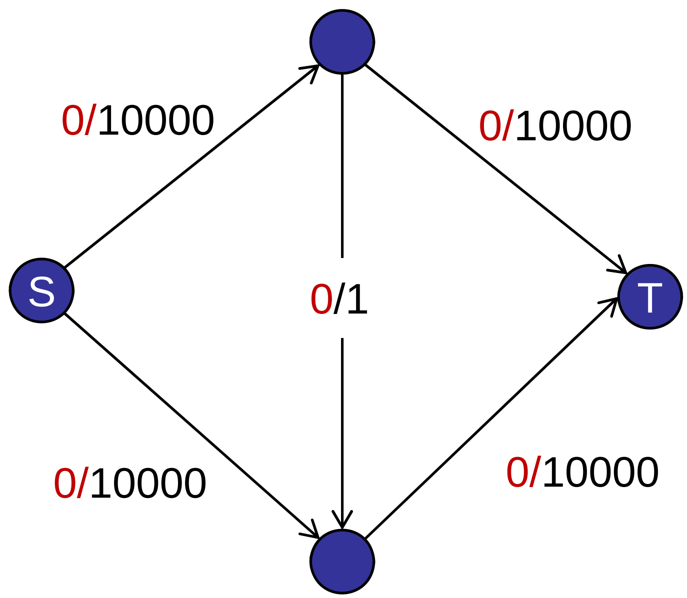
Maximum number of iterations: O(F)

Total: O(FE)

Performance of Ford-Fulkerson

If the maximum capacity $F = 2^{64}$...

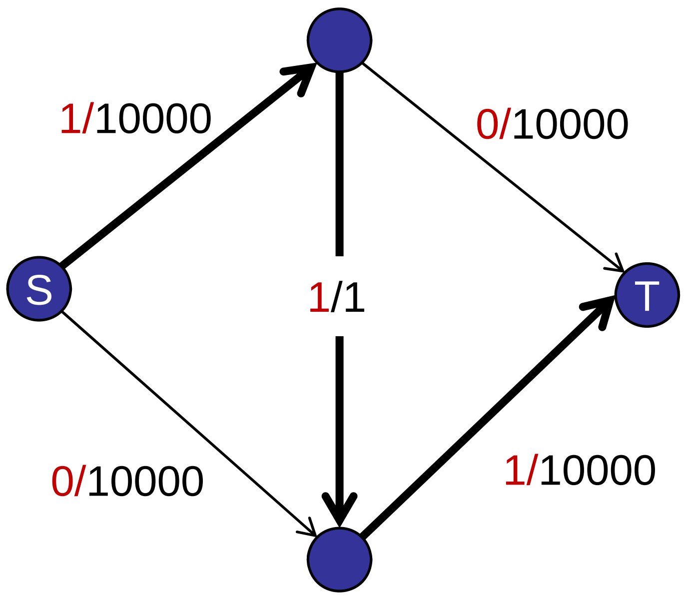
Is it really that bad?



Performance of Ford-Fulkerson

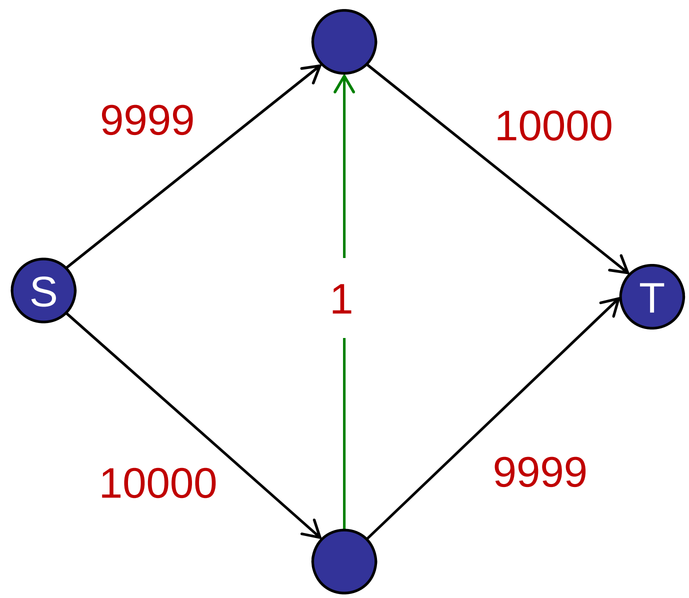
Worst-case performance:

– Step 1: augment with flow 1.



Performance of Ford-Fulkerson

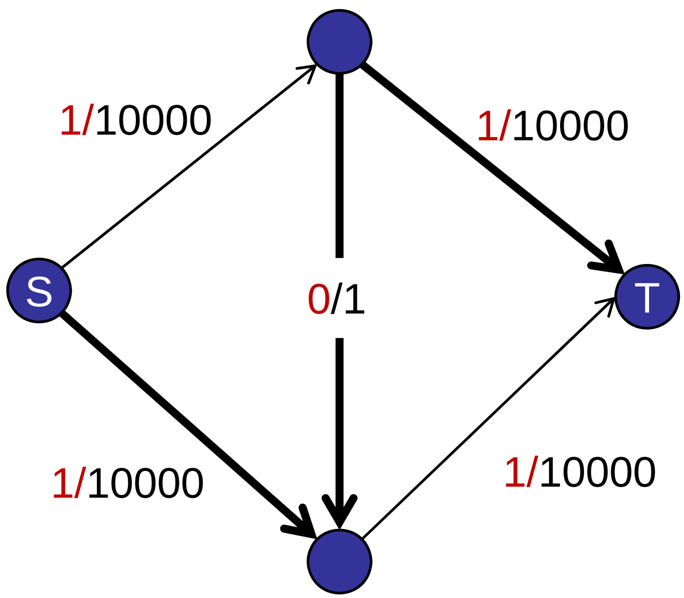
Residual graph:



Performance of Ford-Fulkerson

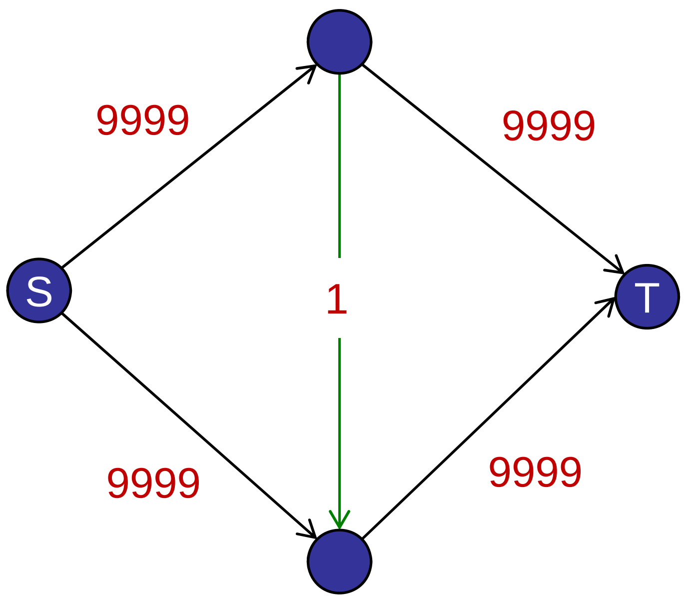
Worst-case performance:

- Step 2: augment with flow 1.



Performance of Ford-Fulkerson

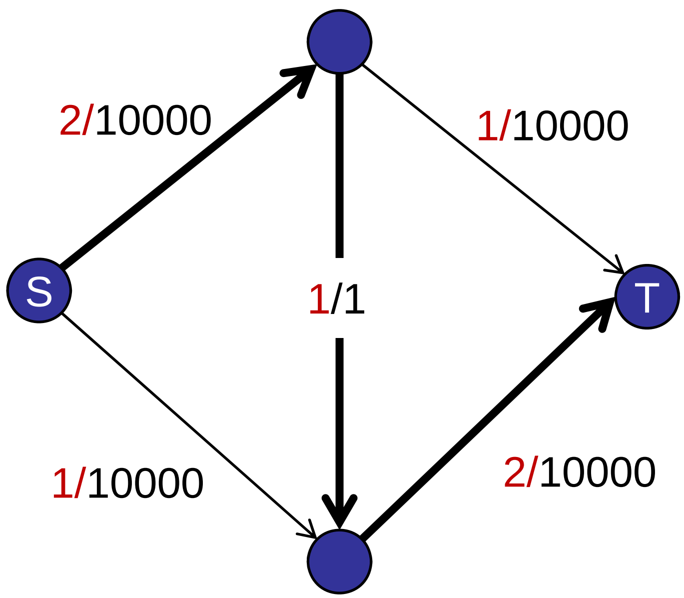
Residual graph:



Performance of Ford-Fulkerson

Worst-case performance:

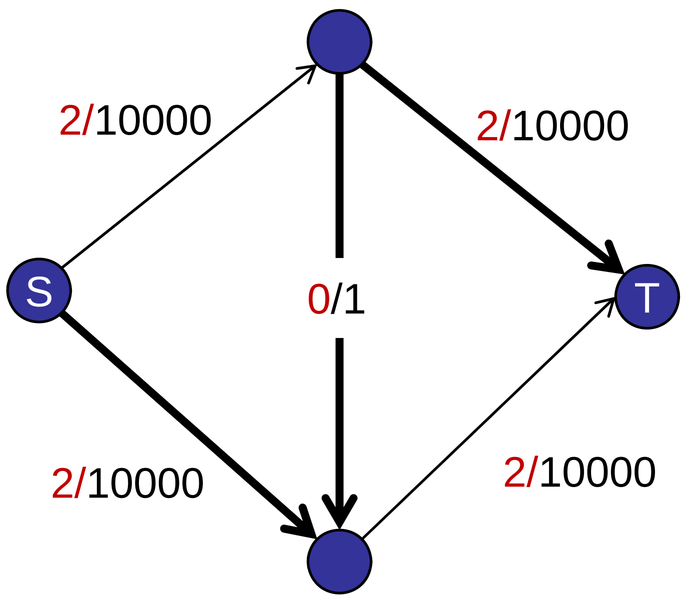
– Step 3: augment with flow 1.



Performance of Ford-Fulkerson

Worst-case performance:

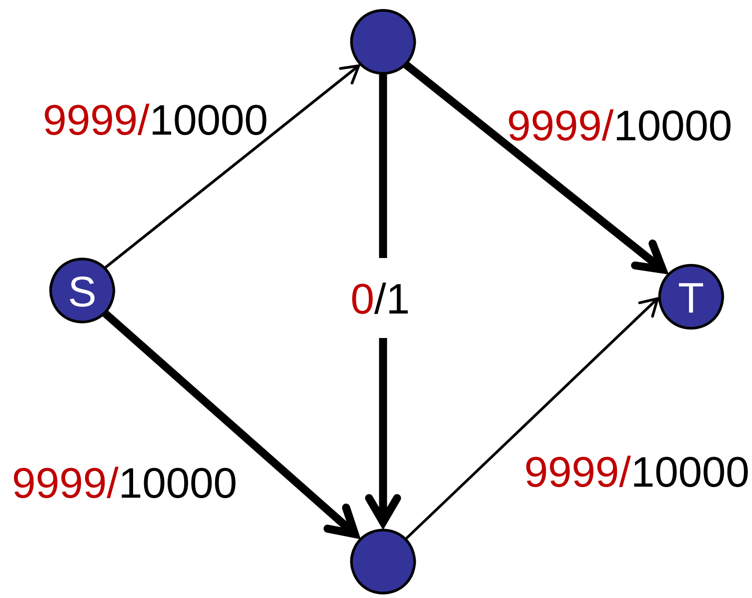
– Step 4: augment with flow 1.



Performance of Ford-Fulkerson

Worst-case performance:

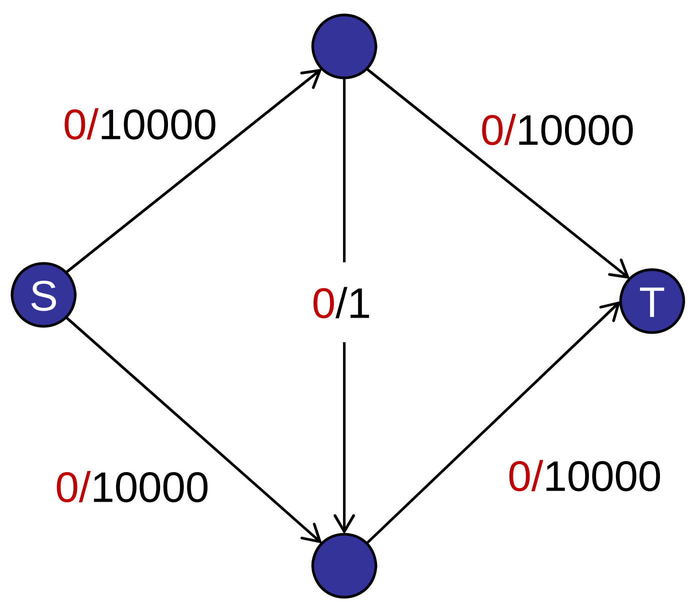
– Step **<u>20000</u>**: augment with flow 1.



Performance of Ford-Fulkerson

Worst-case performance:

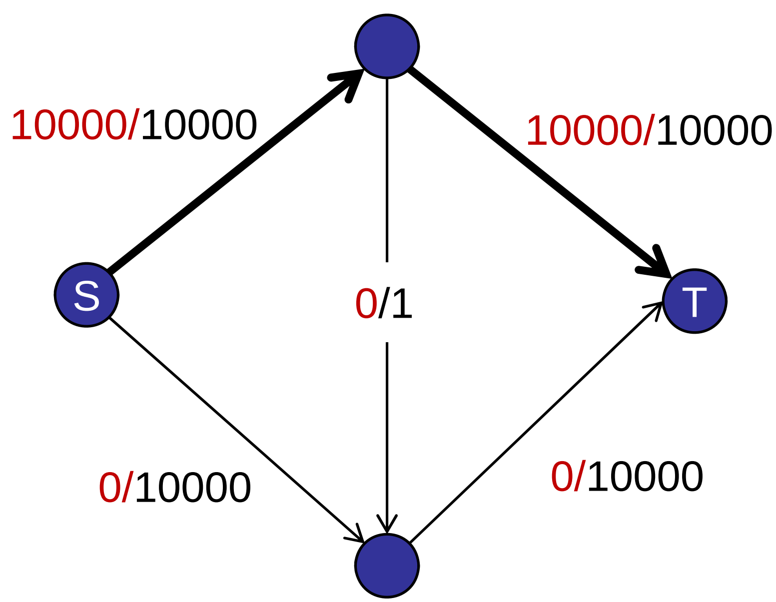
- Problem: bad choice of augmenting paths!
- We only needed 2 steps!



Performance of Ford-Fulkerson

Worst-case performance:

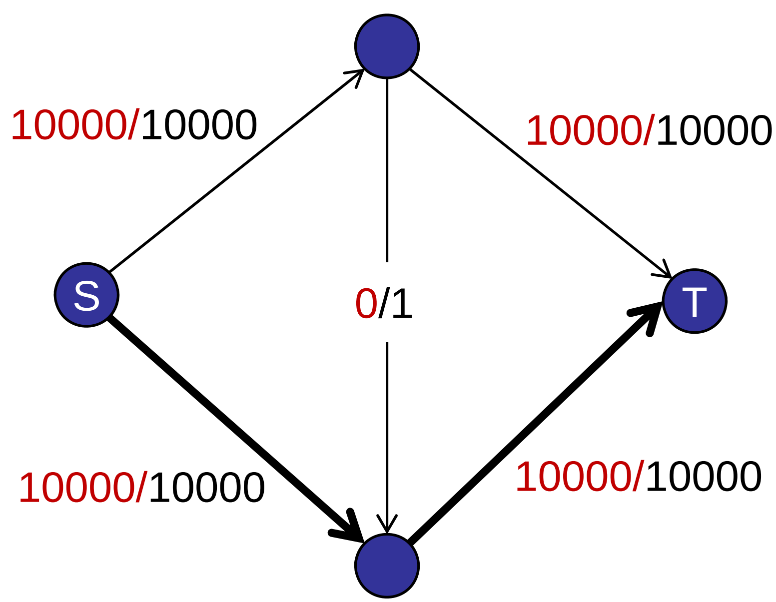
- Problem: bad choice of augmenting paths!
- We only needed 2 steps!



Performance of Ford-Fulkerson

Worst-case performance:

- Problem: bad choice of augmenting paths!
- We only needed 2 steps!



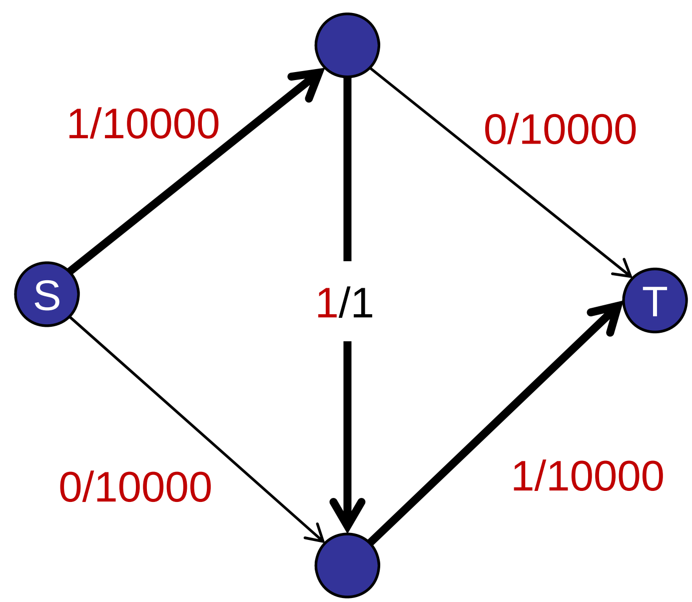
Performance of Ford-Fulkerson

How to choose augmenting paths?

- DFS : O(F·E)

- Breadth-First Search
  - Shortest augmenting path
  - Edmonds-Karp

- Fattest-Path Search
  - Fattest augmenting path
  - Maximize bottleneck edge



Performance of Ford-Fulkerson

How to choose augmenting paths?

- DFS : $O(F \cdot E)$

- Breadth-First Search : $O(VE^2)$
  - Shortest augmenting path (minimum # of hops)
  - Edmonds-Karp

- Fattest-Path Search
  - Fattest augmenting path
  - Maximize bottleneck edge

Performance of Ford-Fulkerson

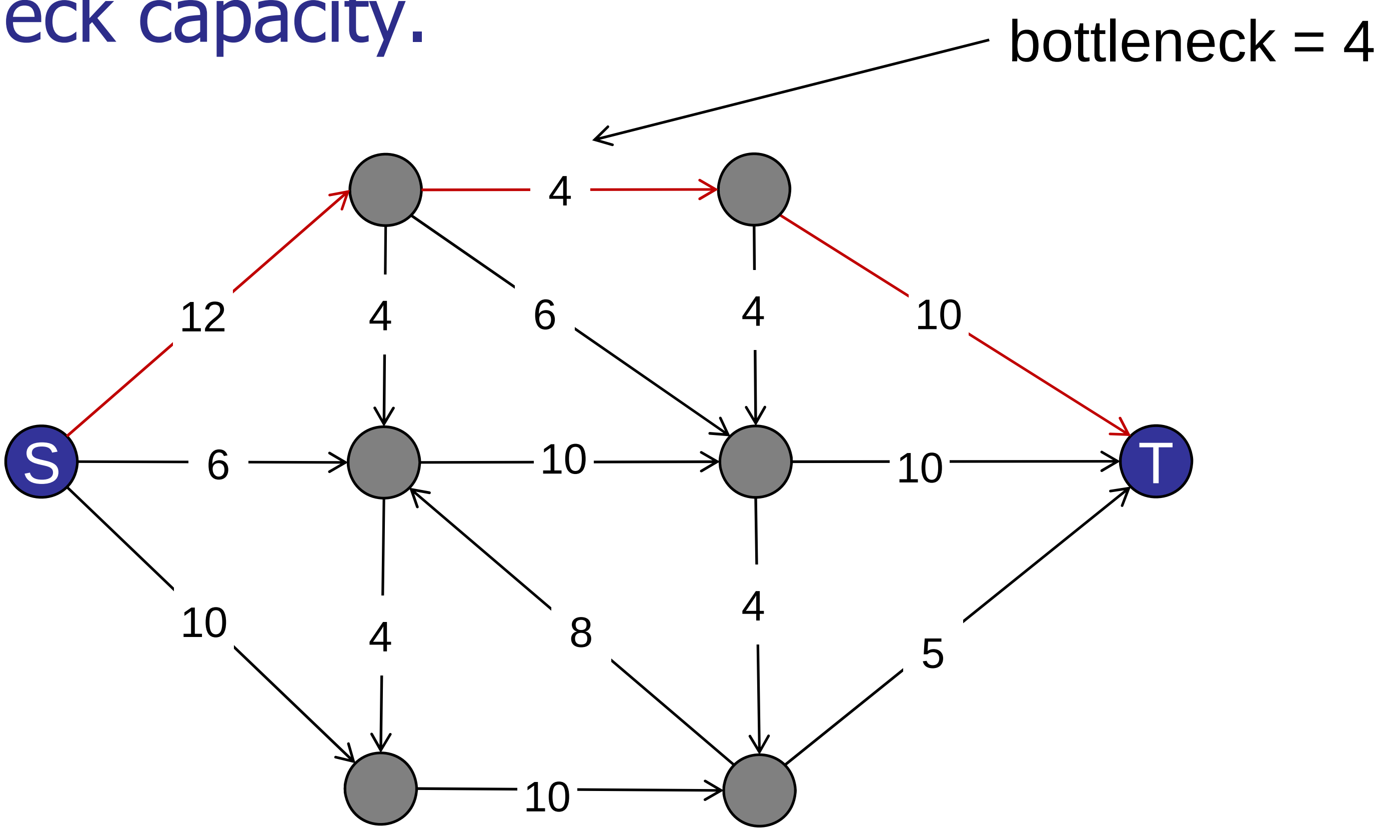
How to choose augmenting paths?

- DFS : $O(F \cdot E)$
- Breadth-First Search : $O(V \cdot E^2)$
- Dinitz : $O(V^2 E)$
  - Use BFS to find **all** shortest paths from ($S \rightarrow T$)
  - Use DFS on shortest-path-tree to find as many augmenting paths as possible.
- Fattest-Path Search

Fattest Path Heuristic

How to choose augmenting paths?

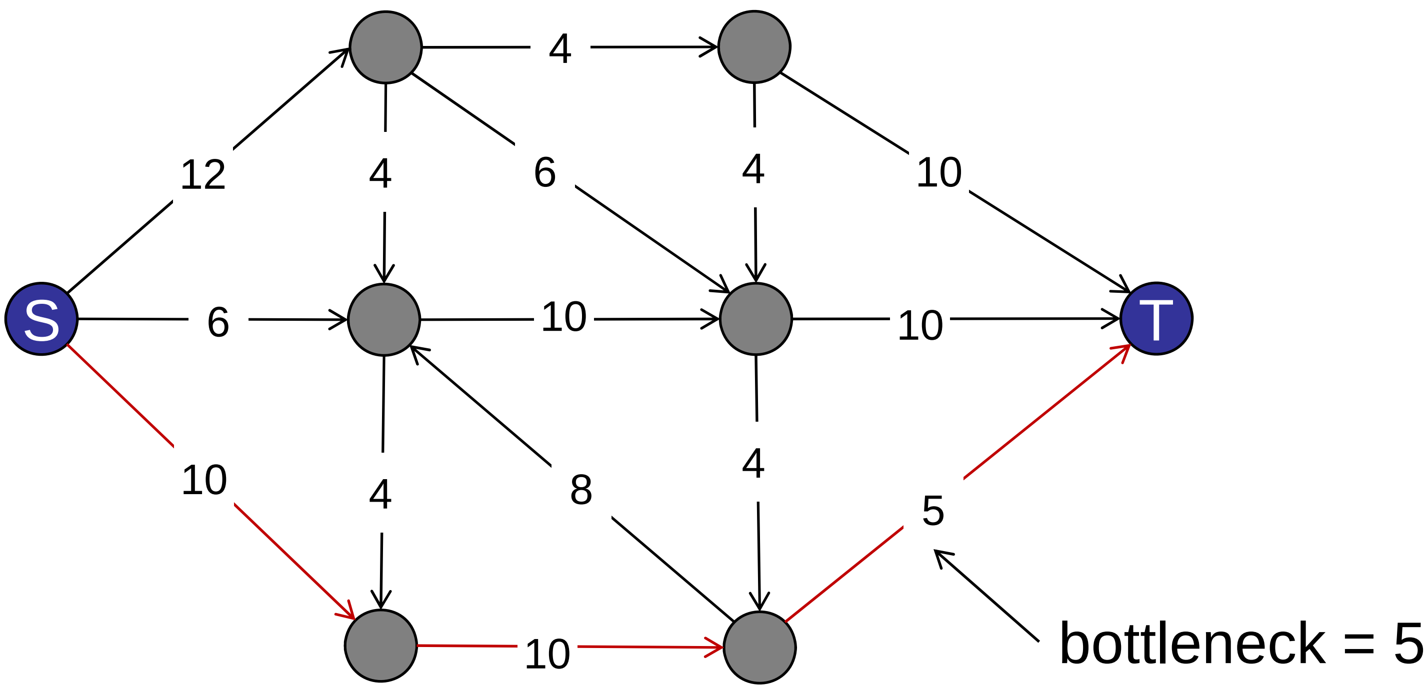
Choose an augmenting path with maximum bottleneck capacity.



Fattest Path Heuristic

How to choose augmenting paths?

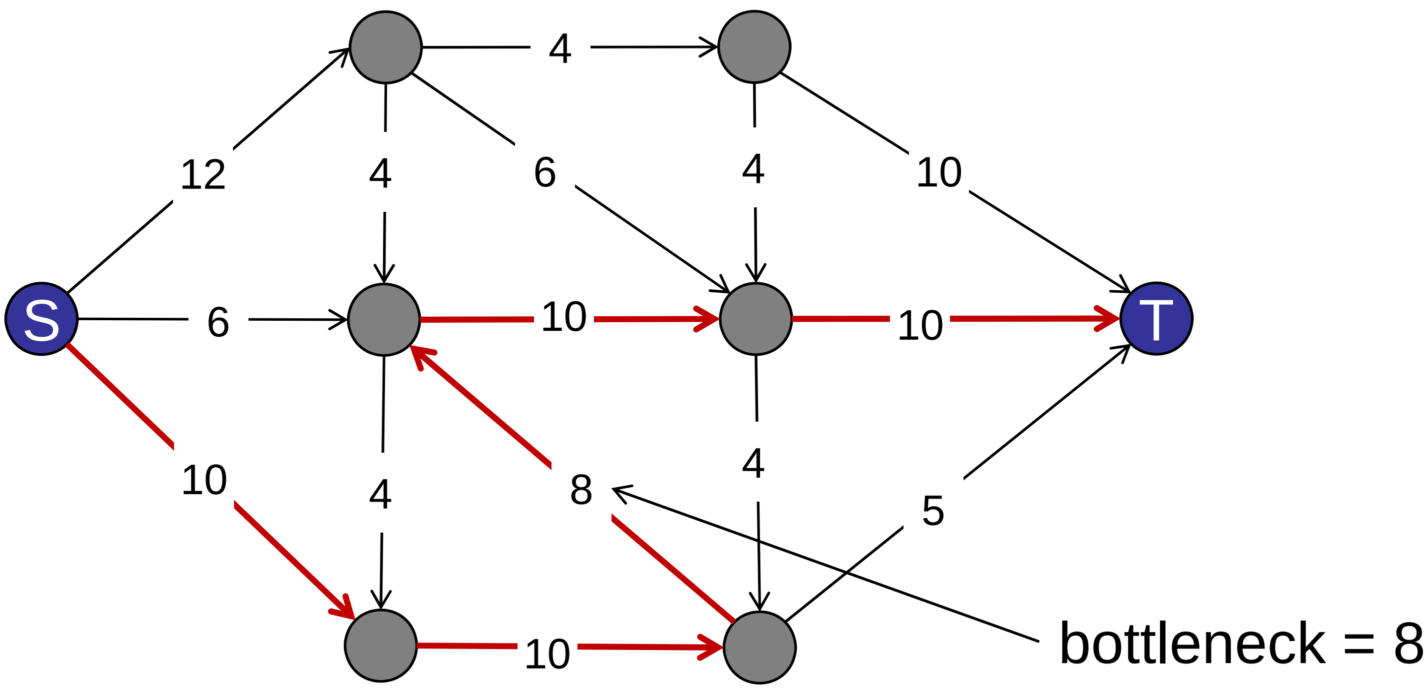
- Choose an augmenting path with maximum bottleneck capacity.



Fattest Path Heuristic

How to choose augmenting paths?

- Choose an augmenting path with maximum bottleneck capacity.



Performance of Ford-Fulkerson

How to choose augmenting paths?

- DFS : $O(F \cdot E)$
- Breadth-First Search : $O(VE^2)$
  - Shortest augmenting path
  - Edmonds-Karp
- Dinitz : $O(V^2E)$
- Fattest-Path Search : $O(E^2 \cdot \log(V) \cdot \log(F))$
  - Fattest augmenting path
  - Maximize bottleneck edge

A brief history of max flow…

year	method	performance	discovered
1951	simplex	$O(E^3F)$	Dantzig
1955	augmenting path	$O(EF)$	Ford-Fulkerson
1970	shortest augmenting path	$O(VE^2)$	Dinitz, Edmonds-Karp
1972	fattest augmenting path	$O(E^2 \log V \log (F))$	Dinitz, Edmonds-Karp
1986	push-relabel	$O(V^2E)$, $O(V^3)$, $O(VE \log(V))$	
1994		$O(VE \log_{E/V\log V} V)$	King-Rao-Tarjan
2006		$O(\min\{V^{2/3},E^{1/2}\}E \log ((V^2/E + 2)\log F)$	Goldberg-Rao
2012	Combo	$O(VE)$	Orlin, KRT

Roadmap

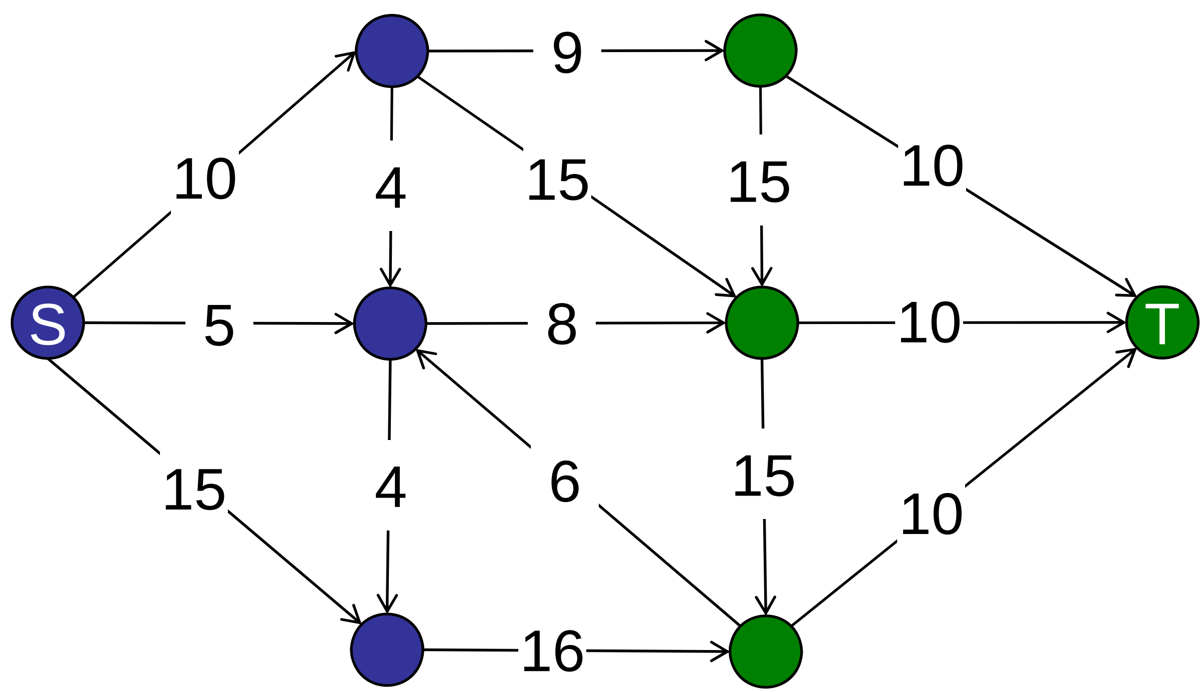
Network Flows

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

Cuts and Flows

Definition:

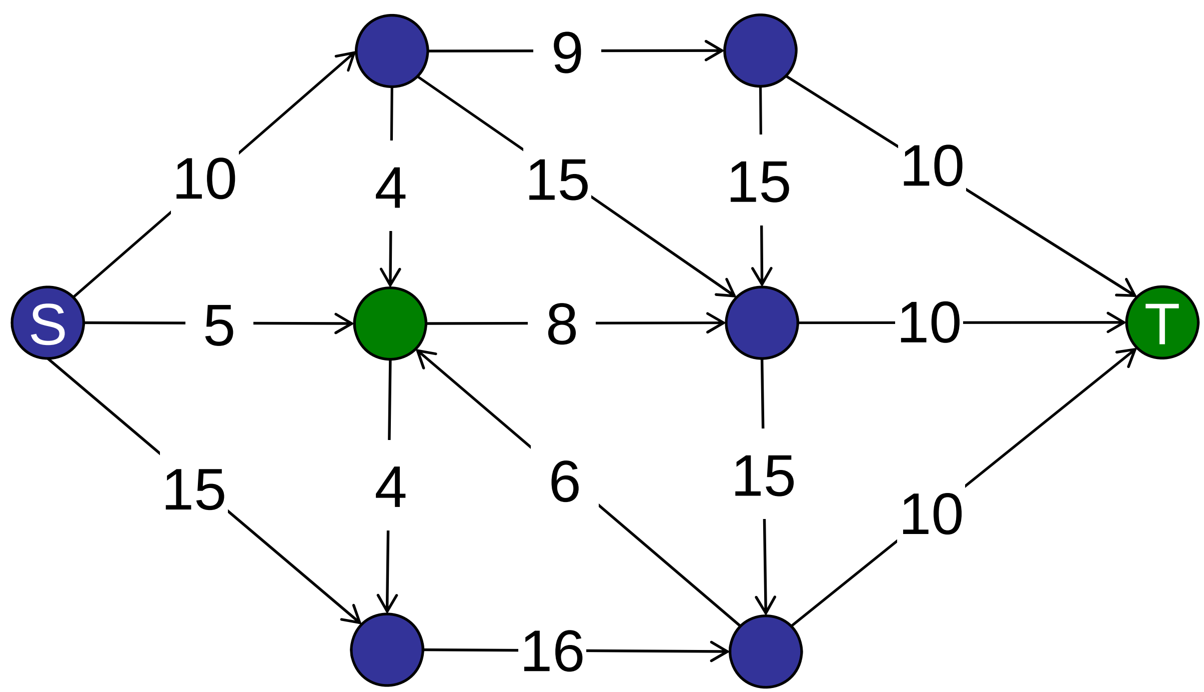
An <u>st-cut</u> partitions the vertices of a graph into two disjoint sets S and T where s $\in$ S and t $\in$ T.



Cuts and Flows

Definition:

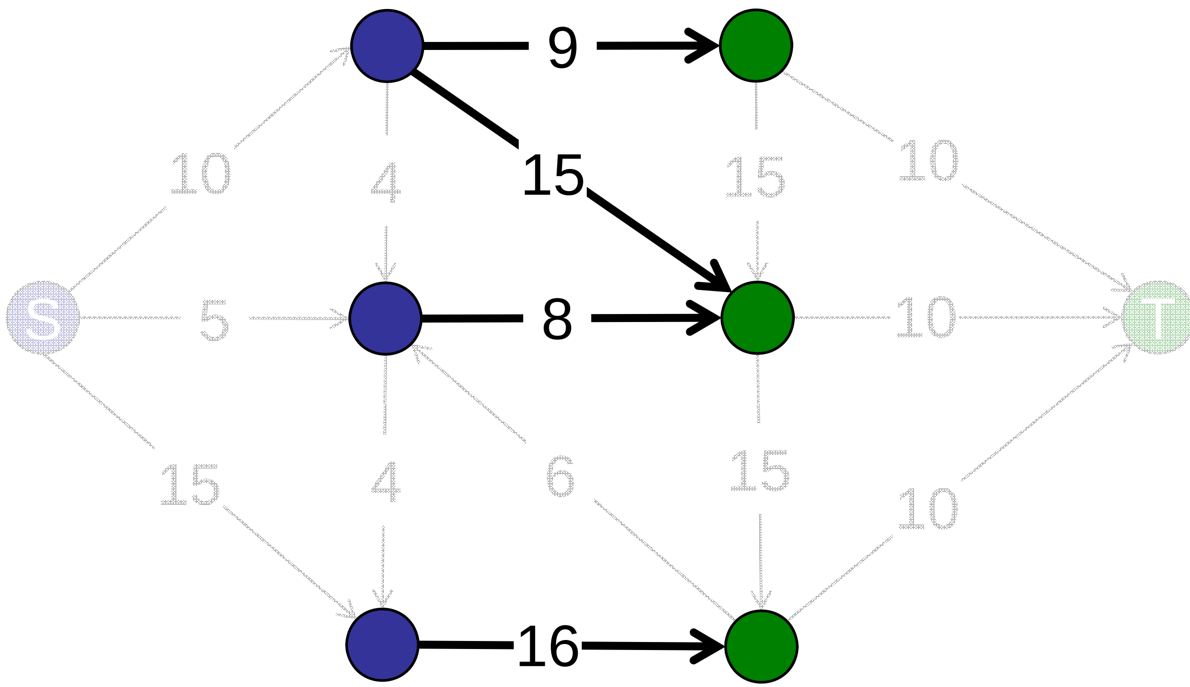
An st-cut partitions the vertices of a graph into two disjoint sets S and T where $s \in S$ and $t \in T$.



Cuts and Flows

Definition:

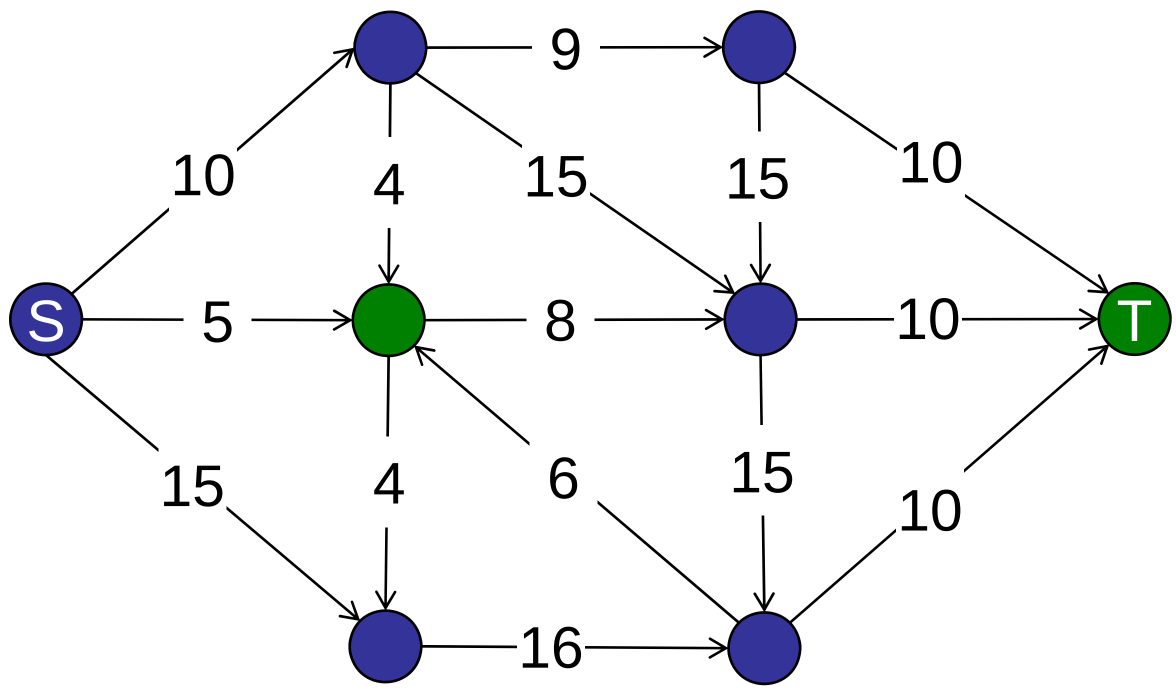
The **capacity** of an st-cut is the sum of the capacities of the edges that cross the cut from S to T.

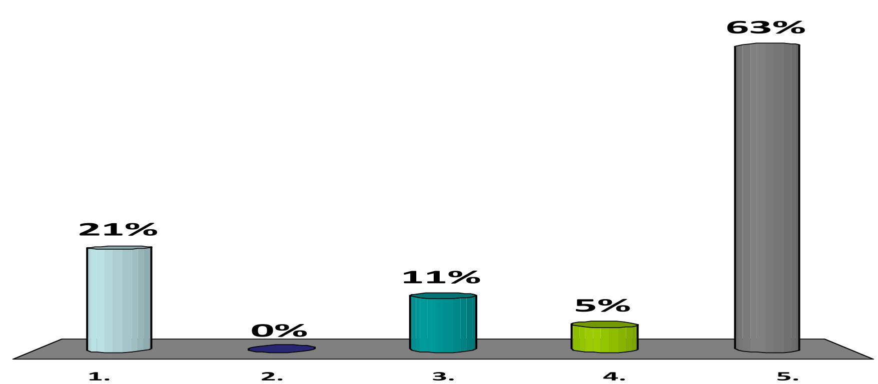


Capacity = 48

What is the capacity of this st-cut?

1. 30
2. 33
3. 35
4. 39
5. 45

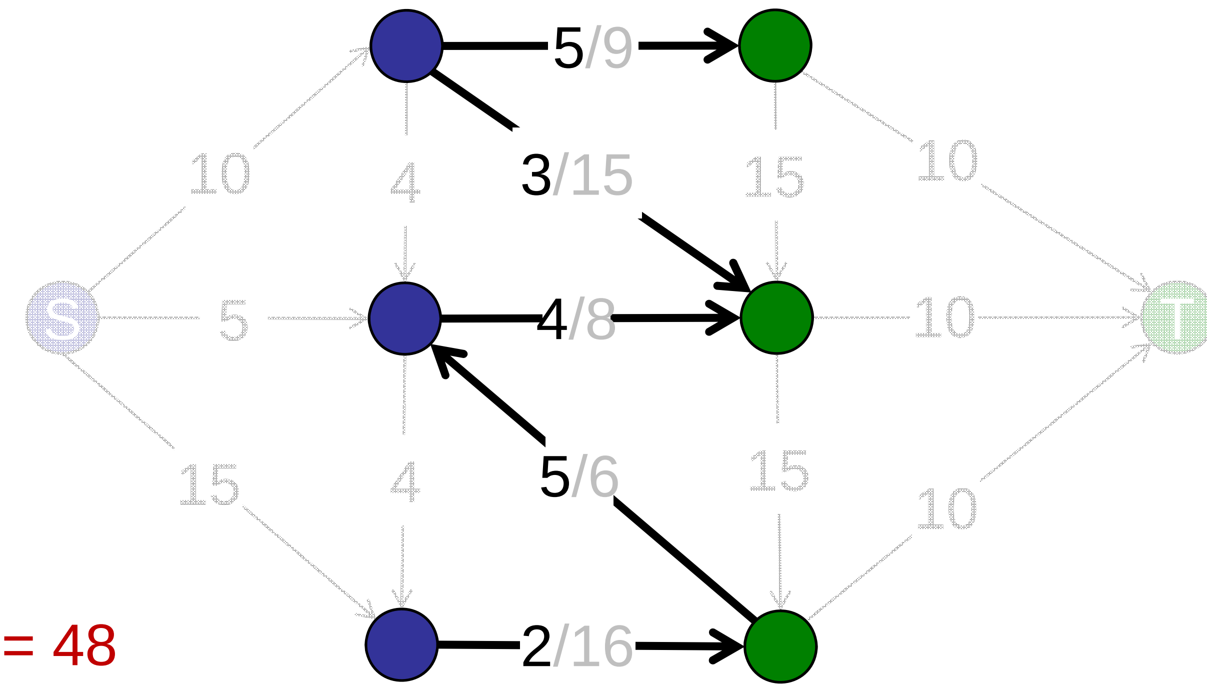




Cuts and Flows

Definition:

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



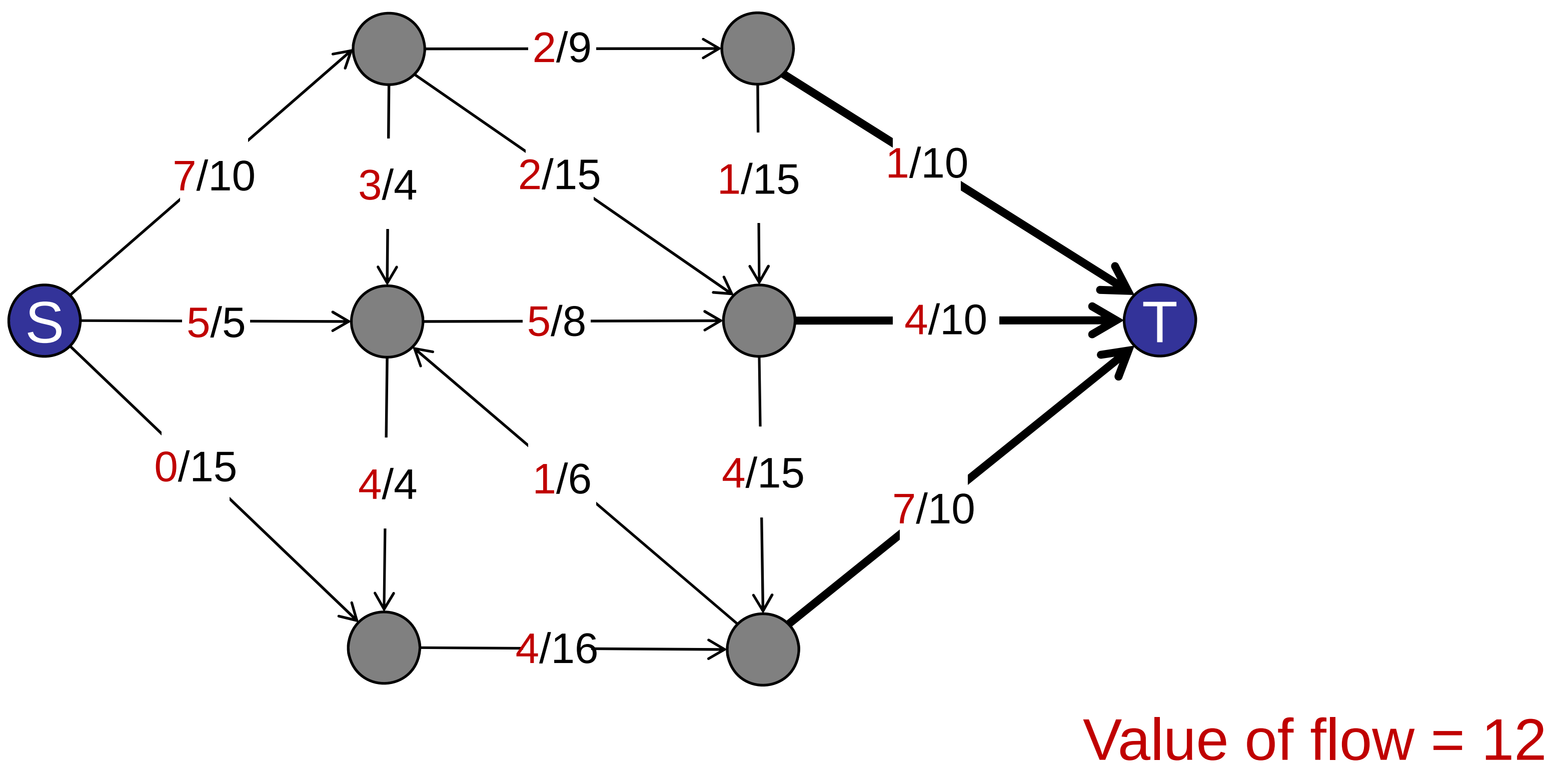
Capacity = 48
Net flow Φ = 9

Cuts and Flows

Proposition:

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

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

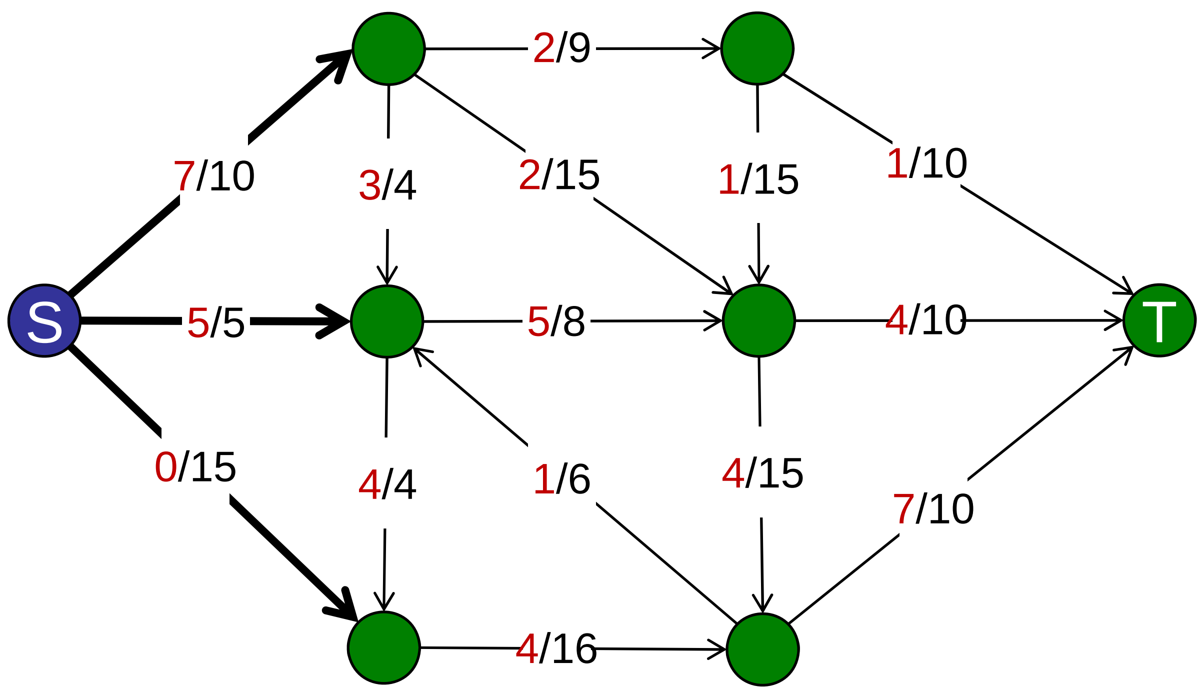


Cuts and Flows

Proposition:

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

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



Flow across cut = 12

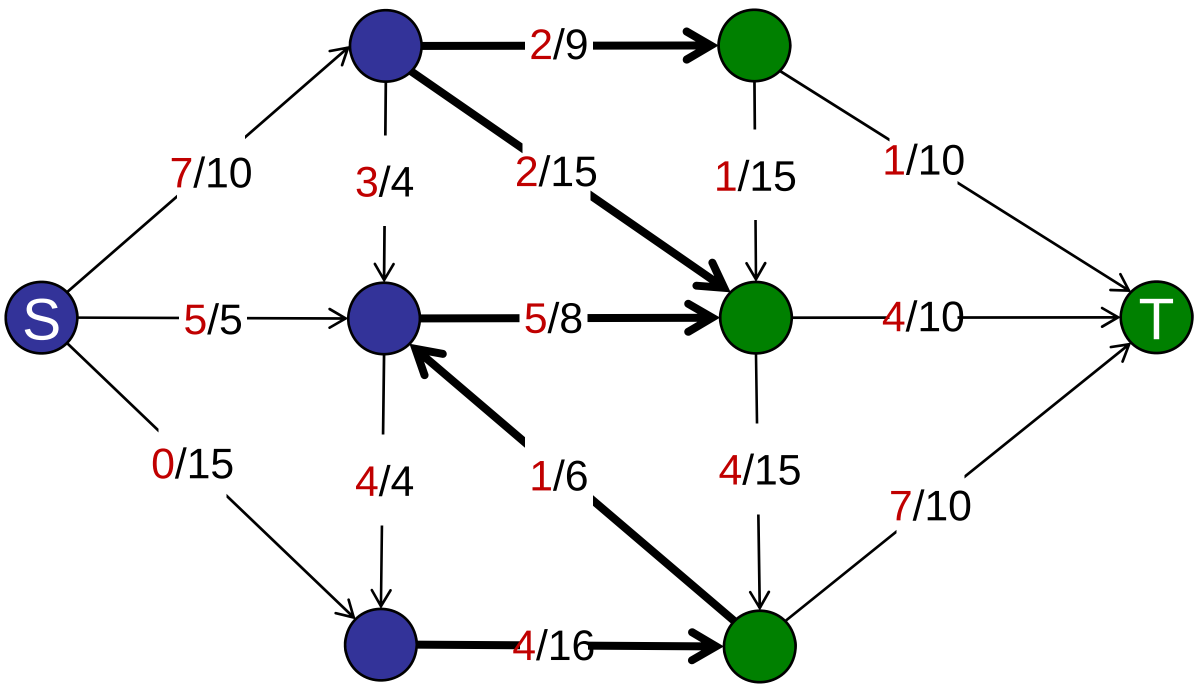
Value of flow = 12

Cuts and Flows

Proposition:

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

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



Flow across cut = 12

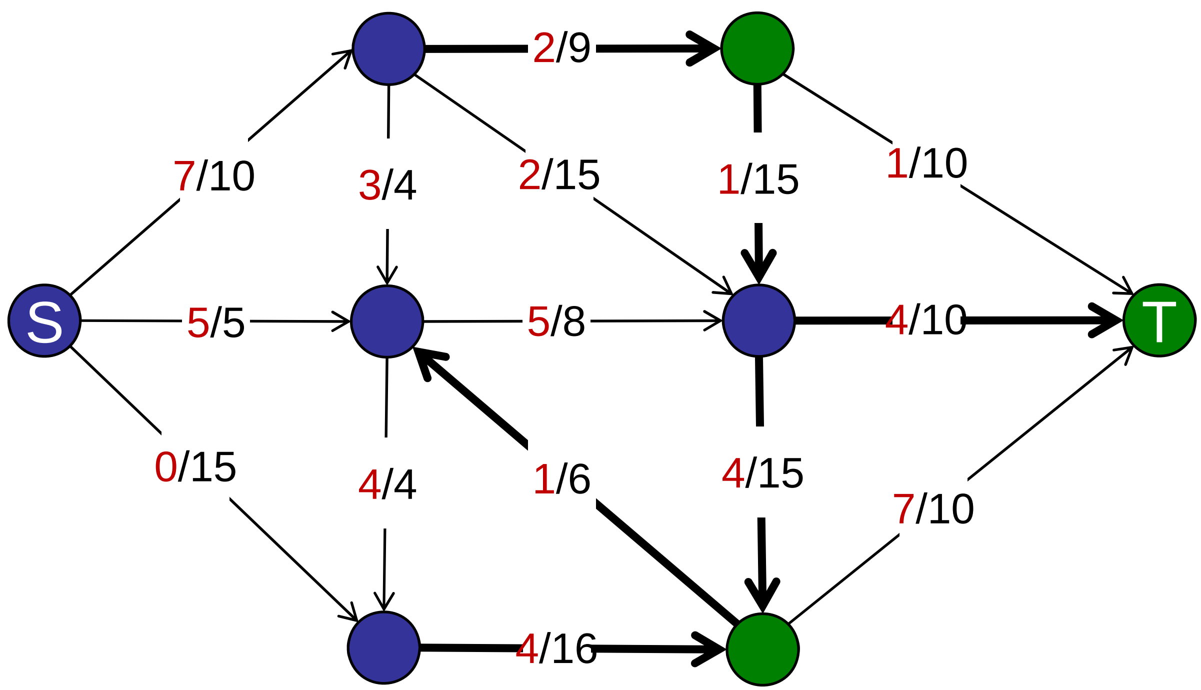
Value of flow = 12

Cuts and Flows

Proposition:

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

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



Flow across cut = 12

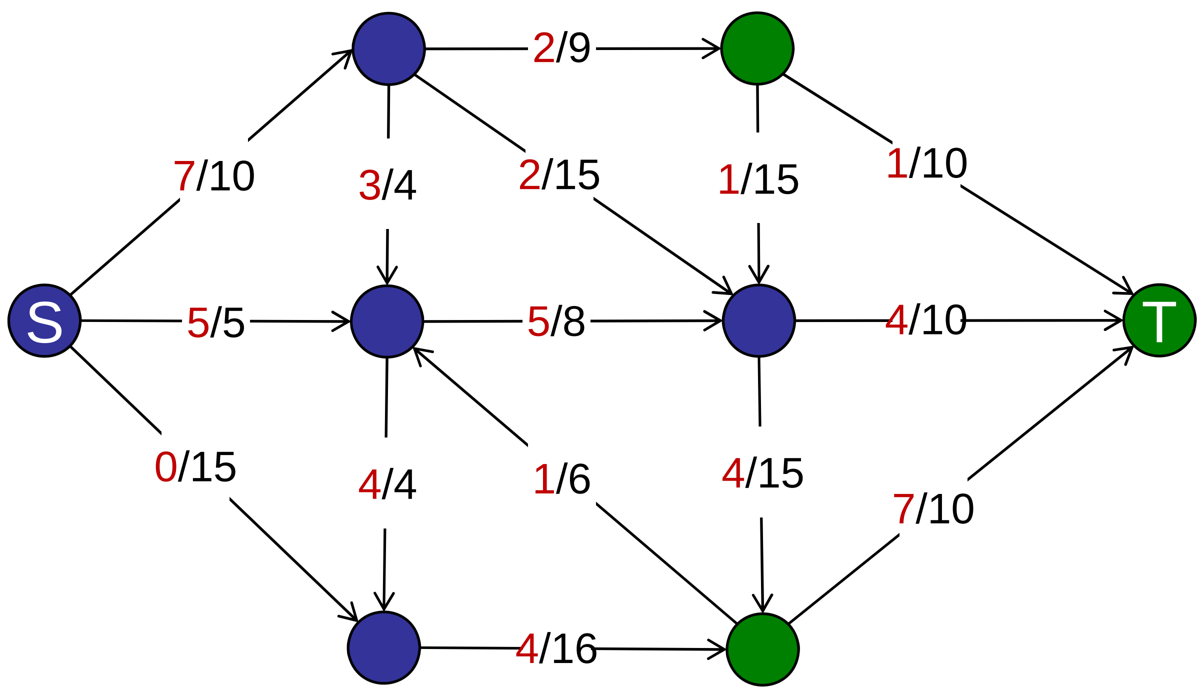
Value of flow = 12

Cuts and Flows

Proposition:

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

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



Flow across cut = 12

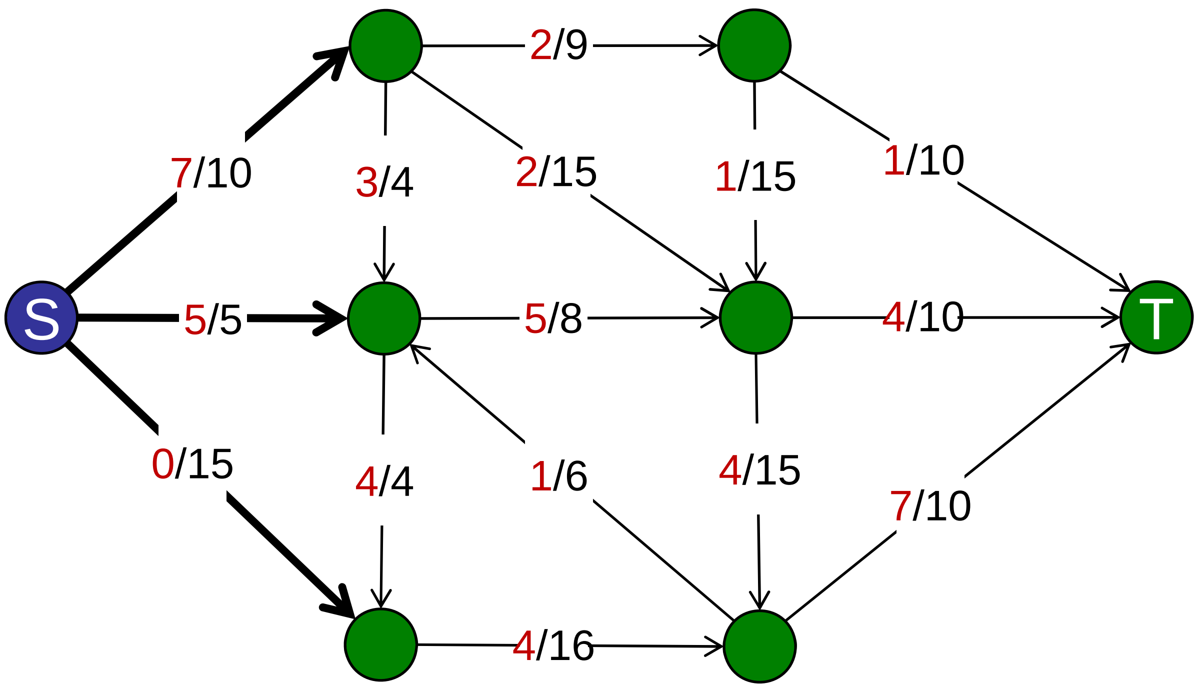
Value of flow = 12

Cuts and Flows

Proof: (by induction)

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

Define Φ = flow across cut.



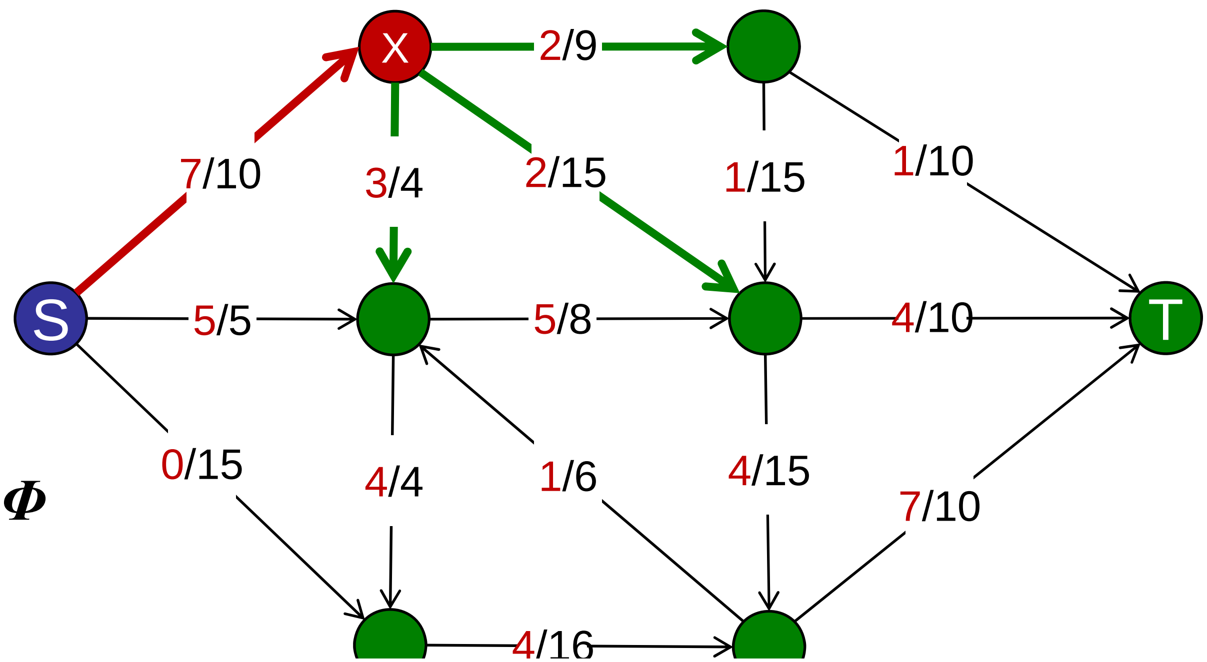
Cuts and Flows

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.

$\boldsymbol{\Phi} - 7 + (3 + 2 + 2) = \boldsymbol{\Phi}$



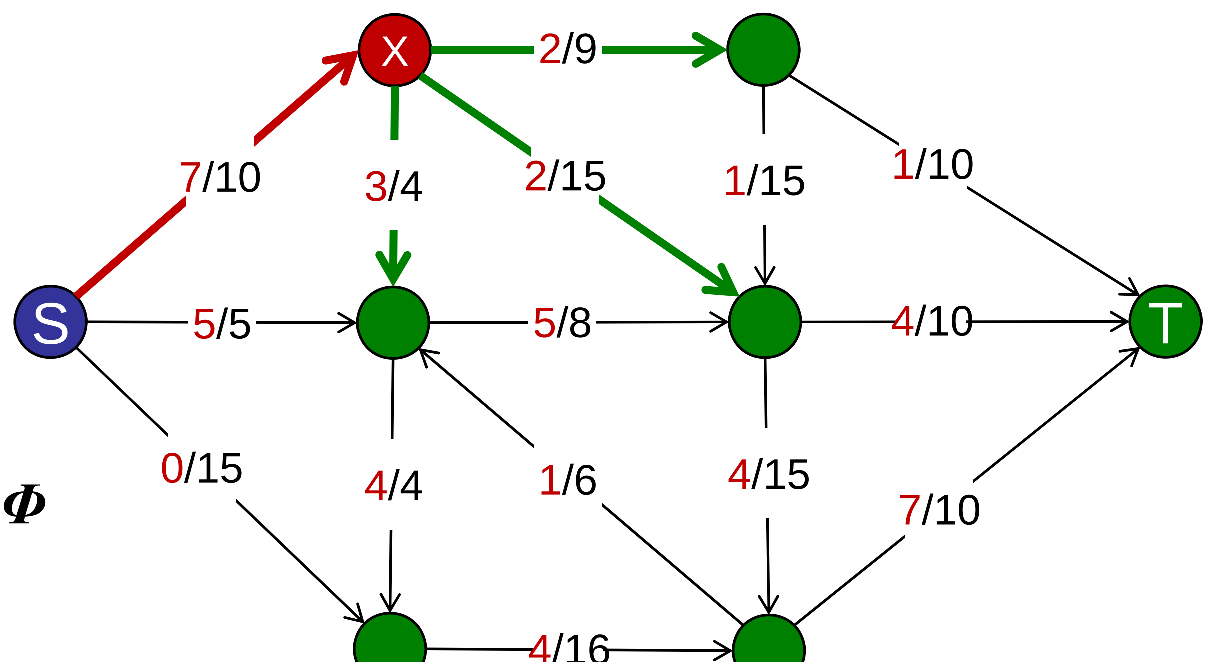
Cuts and Flows

Inductive step:

Conservation of flow: (equilibrium constraint)

- Flow into X equals flow out of X.
- Flow that crossed (old S)→X == X→(old T)
- $\boldsymbol{\Phi}$ remains unchanged

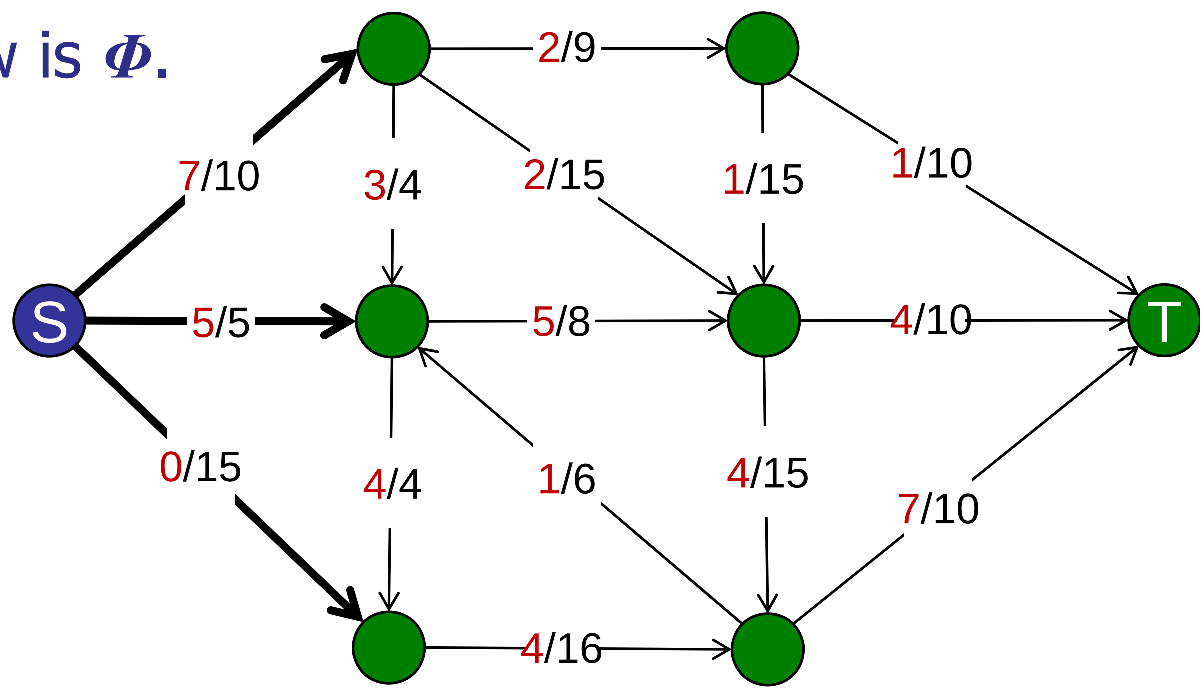
$\boldsymbol{\Phi} - 7 + (3 + 2 + 2) = \boldsymbol{\Phi}$



Cuts and Flows

Proof: (by induction)

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

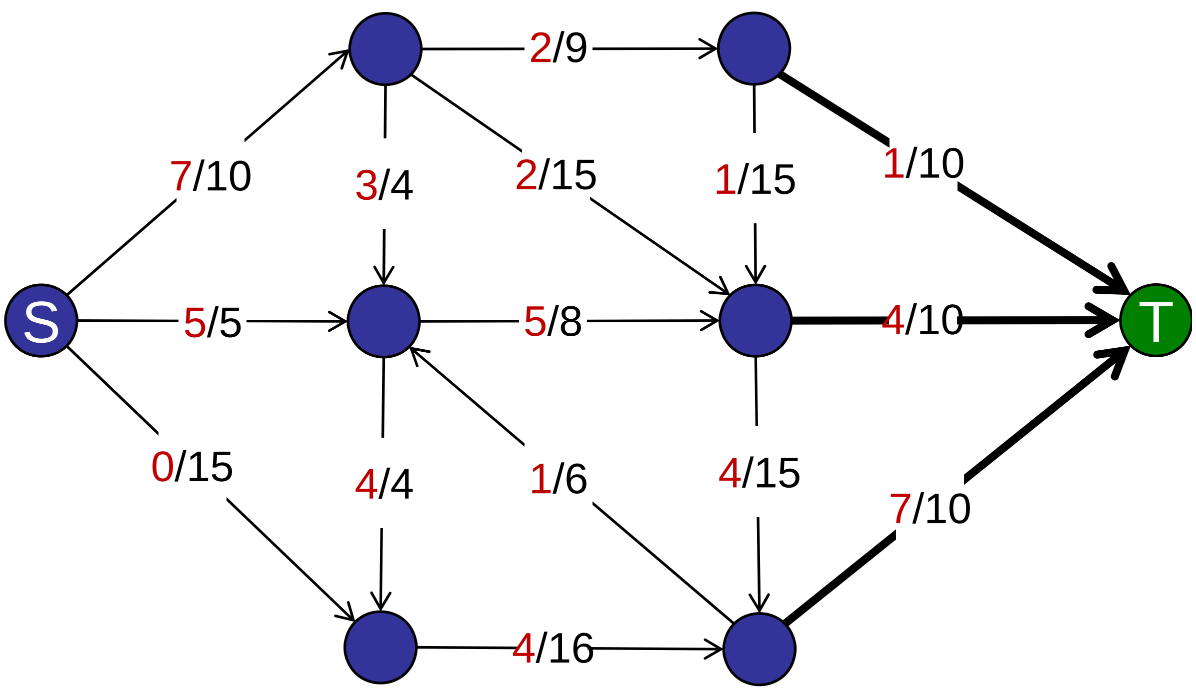


Cuts and Flows

Proof: (by induction)

What is Φ?

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

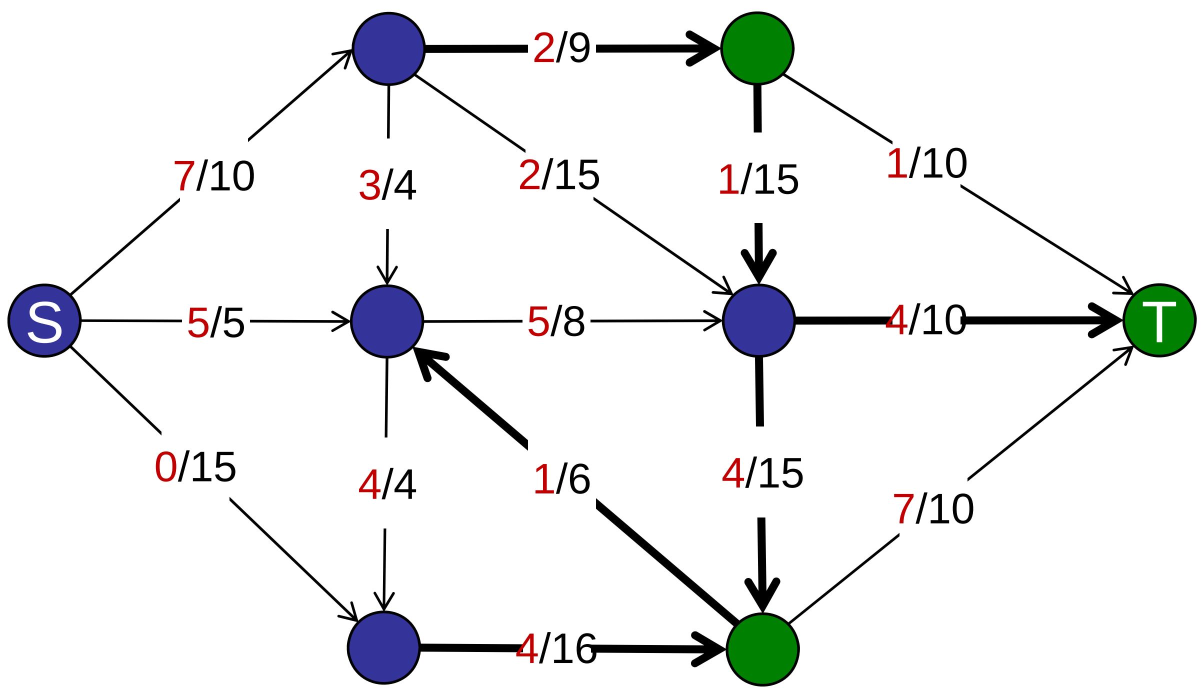


Cuts and Flows

Proposition (Flow Value):

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

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



Flow across cut = 12

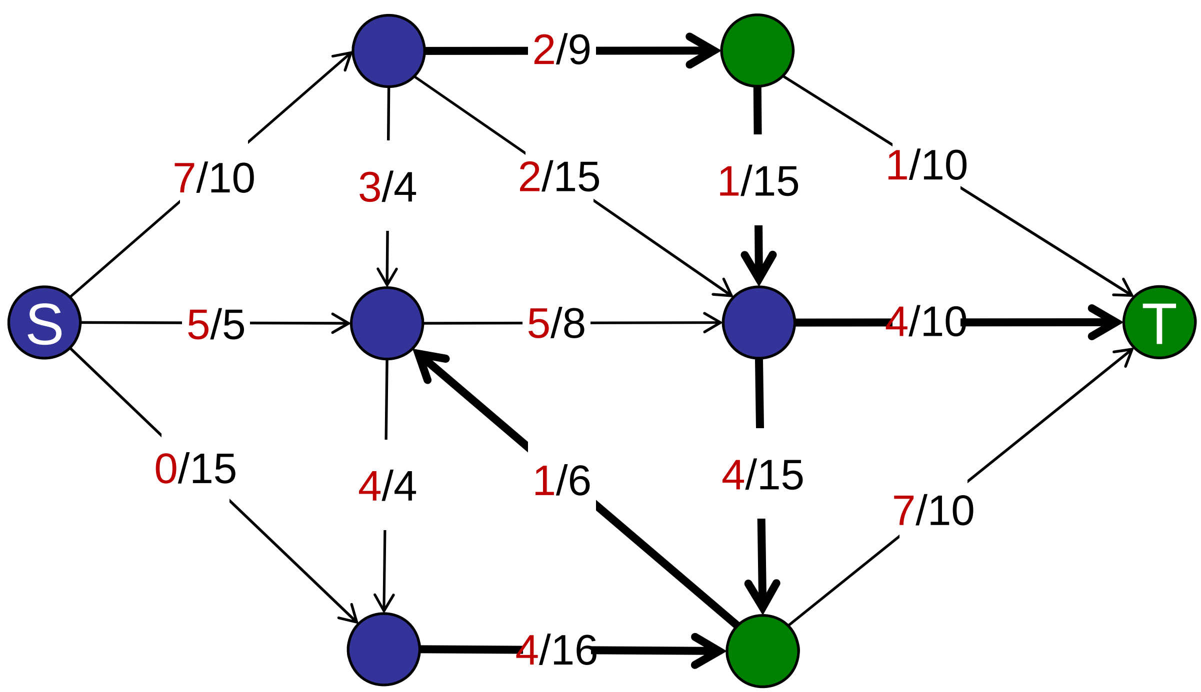
Value of flow = 12

Cuts and Flows

Weak duality:

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

Then value(f) ≤ 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) $\boldsymbol{\Phi} \leq$ capacity(S,T).

flow value proposition

flow is bounded by the capacity

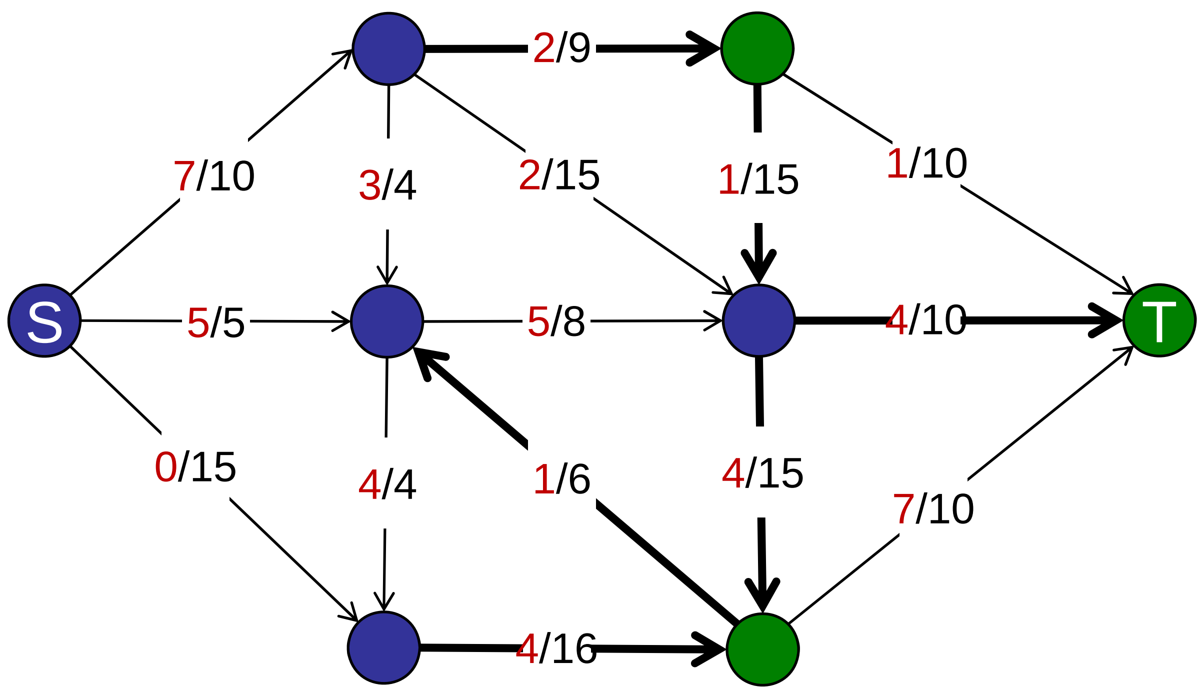
Cuts and Flows

MaxFlow-MinCut Theorem:

Let f be a maximum flow.

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

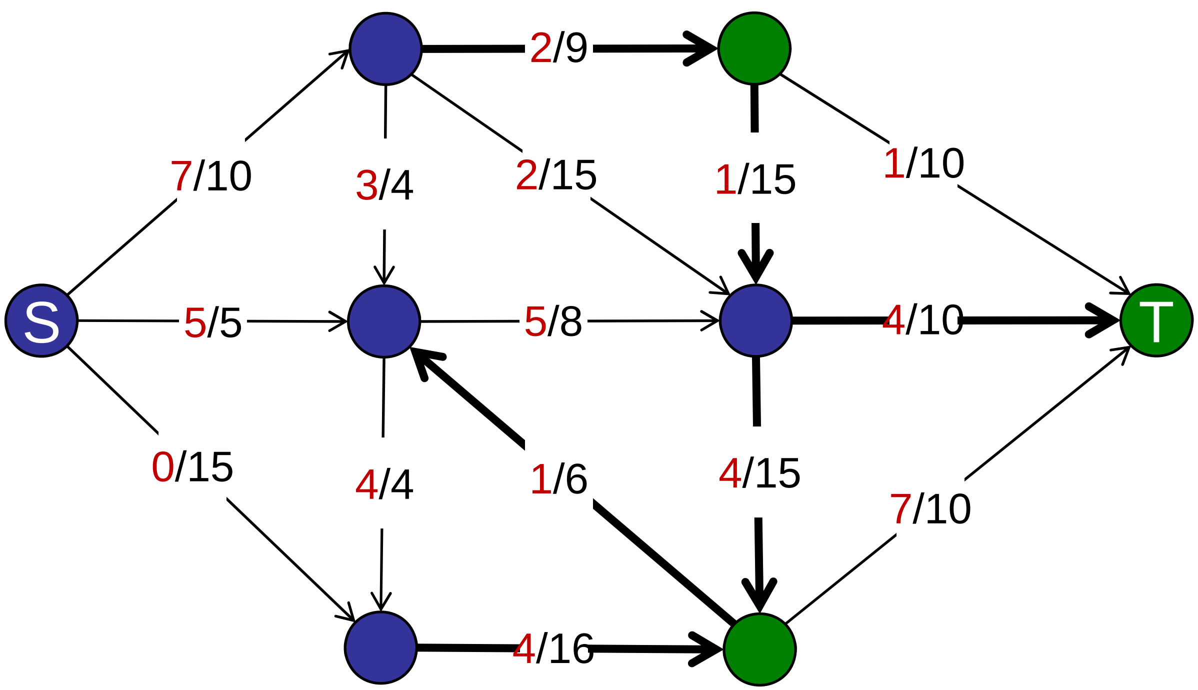
Then value(f) = capacity(S,T).



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.

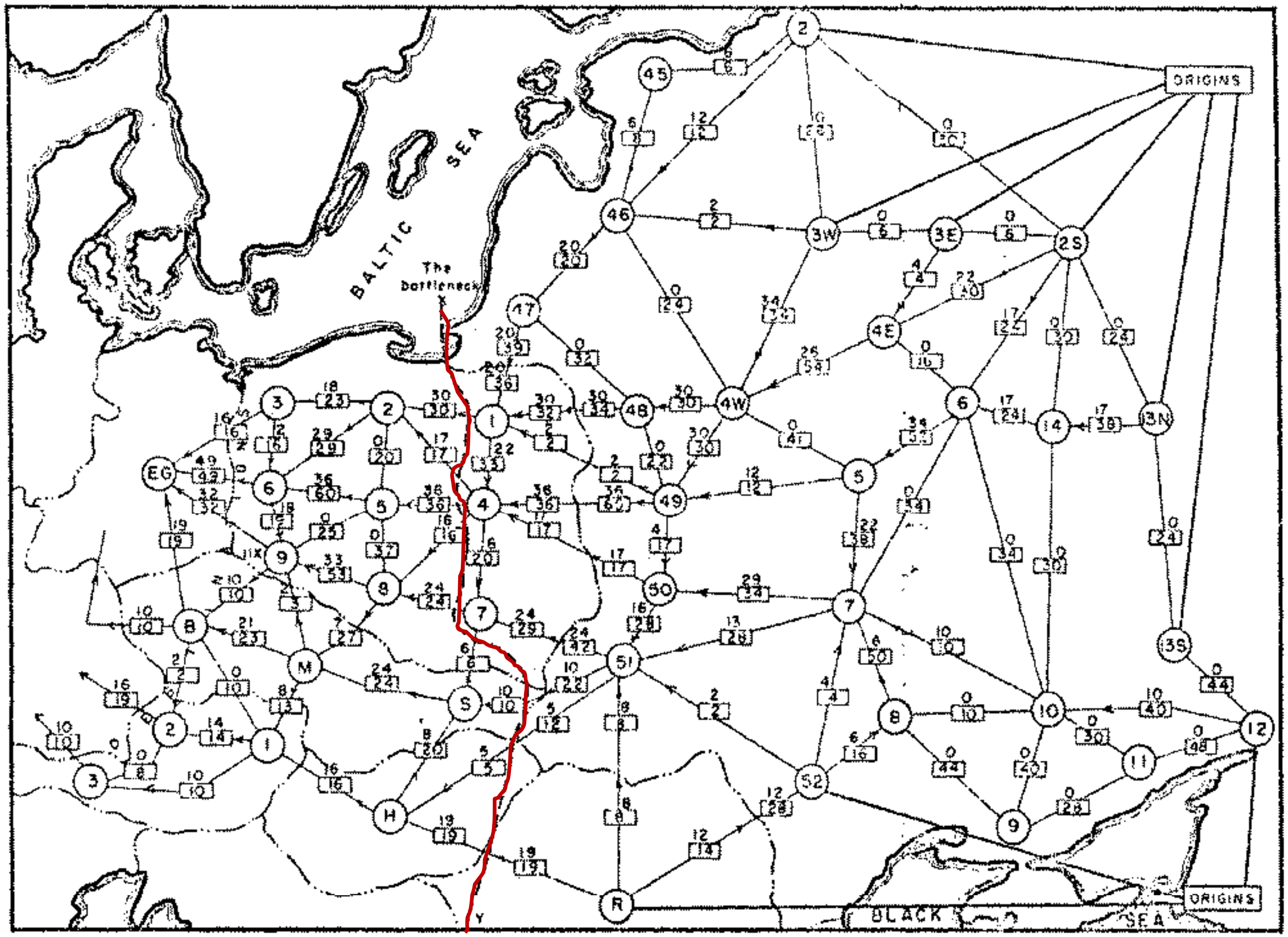


Cuts and Flows

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.



Declassified US schematic of the railway network connecting Eastern Europe and the Soviet Union. Cut capacity = 163,000 tons.

From: Harris and Ross [1955], via

Schrijver “On the history of the transportation and maximum flow problems.”

Cuts and Flows

Max-Flow / Min-Cut

For a graph G = (V,E), the maximum st-flow is *equal* to the value of the minimum st-cut.

Cuts and Flows

Augmenting Path Theorem:

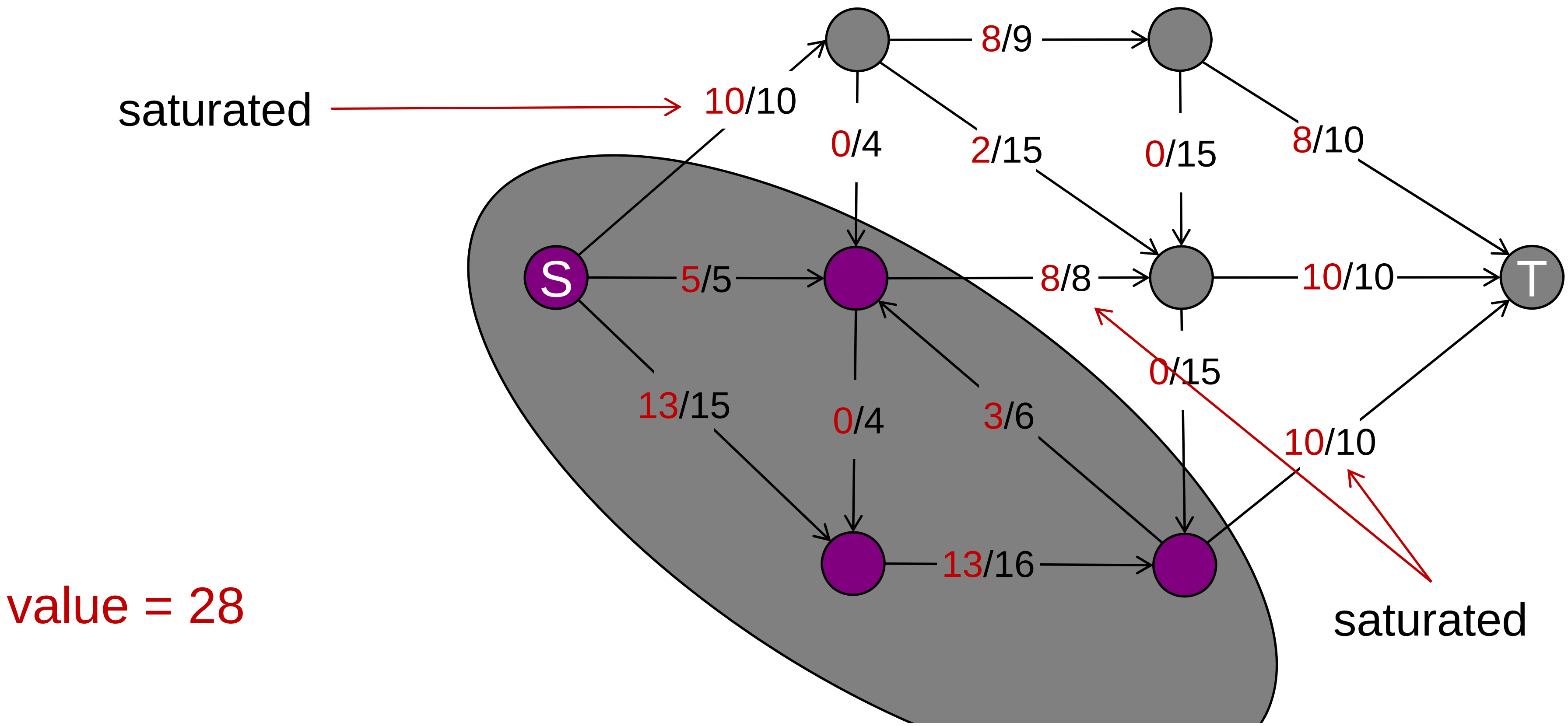
Flow f is a maximum flow if and only if there are no augmenting paths in the residual graph.

➔ If Ford-Fulkerson terminates, then there is no augmenting path. Thus, the resulting flow is maximum.

Ford-Fulkerson

Augmenting path: Undirected path from s → t

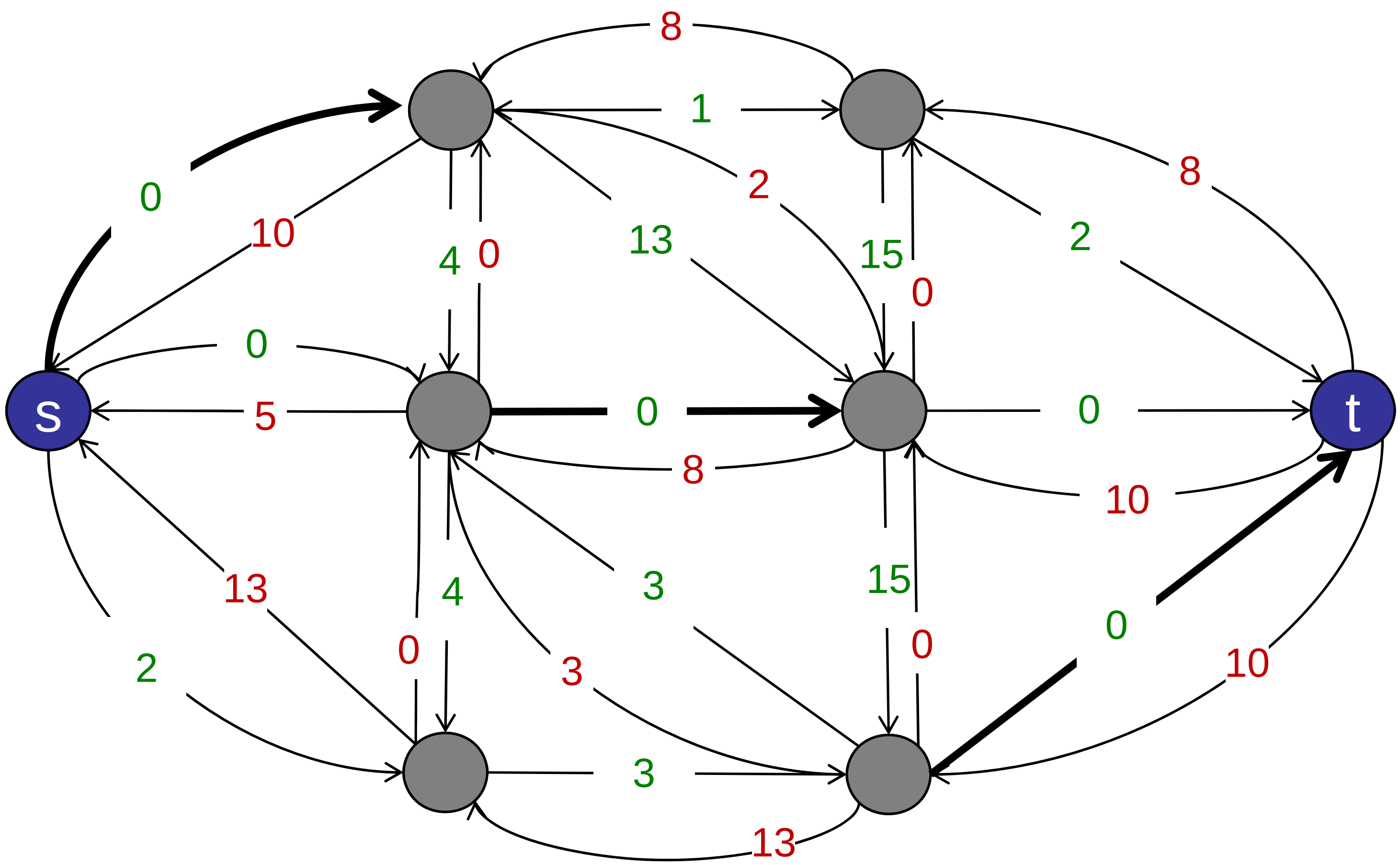
- Can increase flow on all forward edges OR
- Can decrease flow on backward edges



Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)

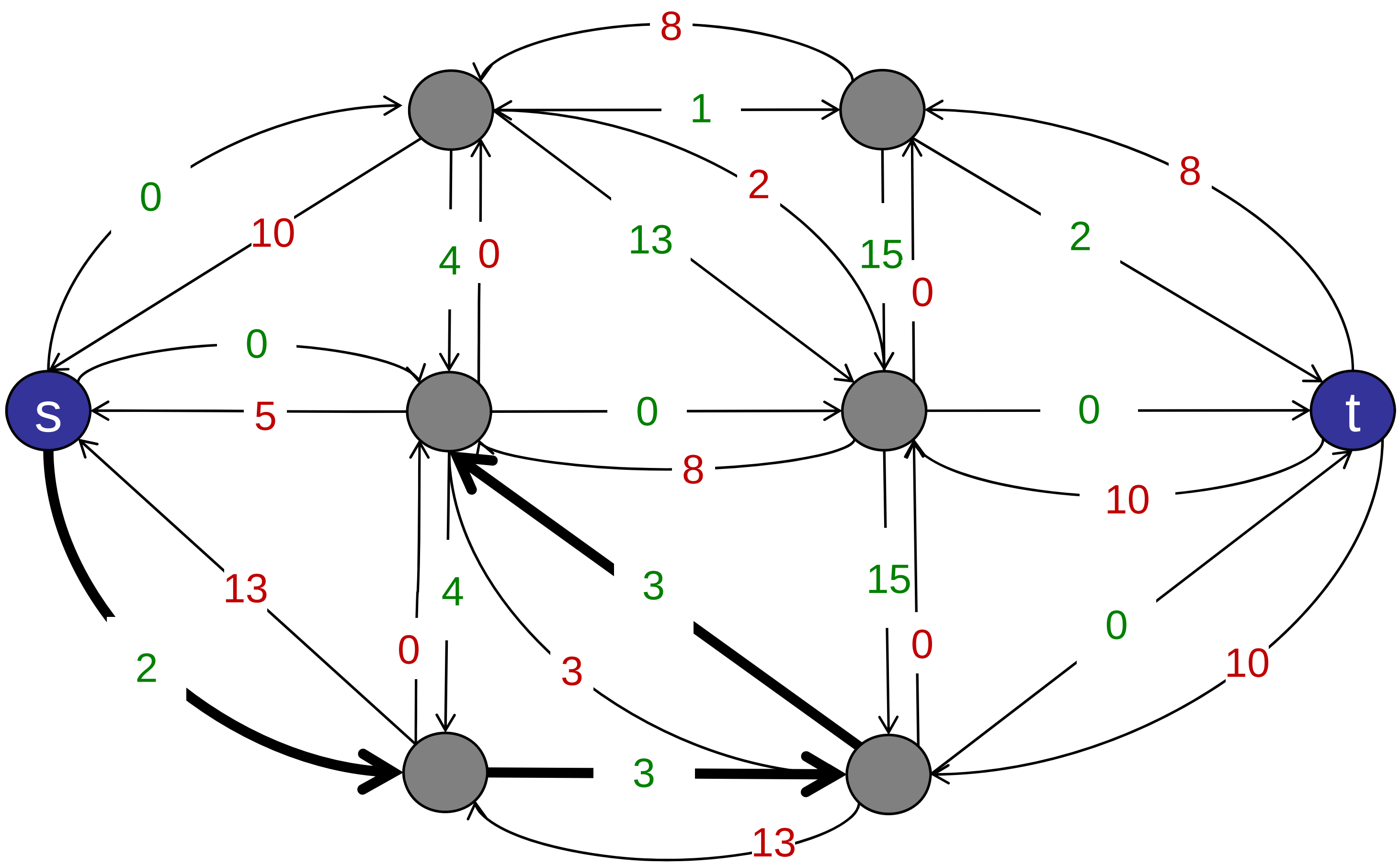


Forward flow = reverse residual flow.

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)



Forward flow = reverse residual flow.

Cuts and Flows

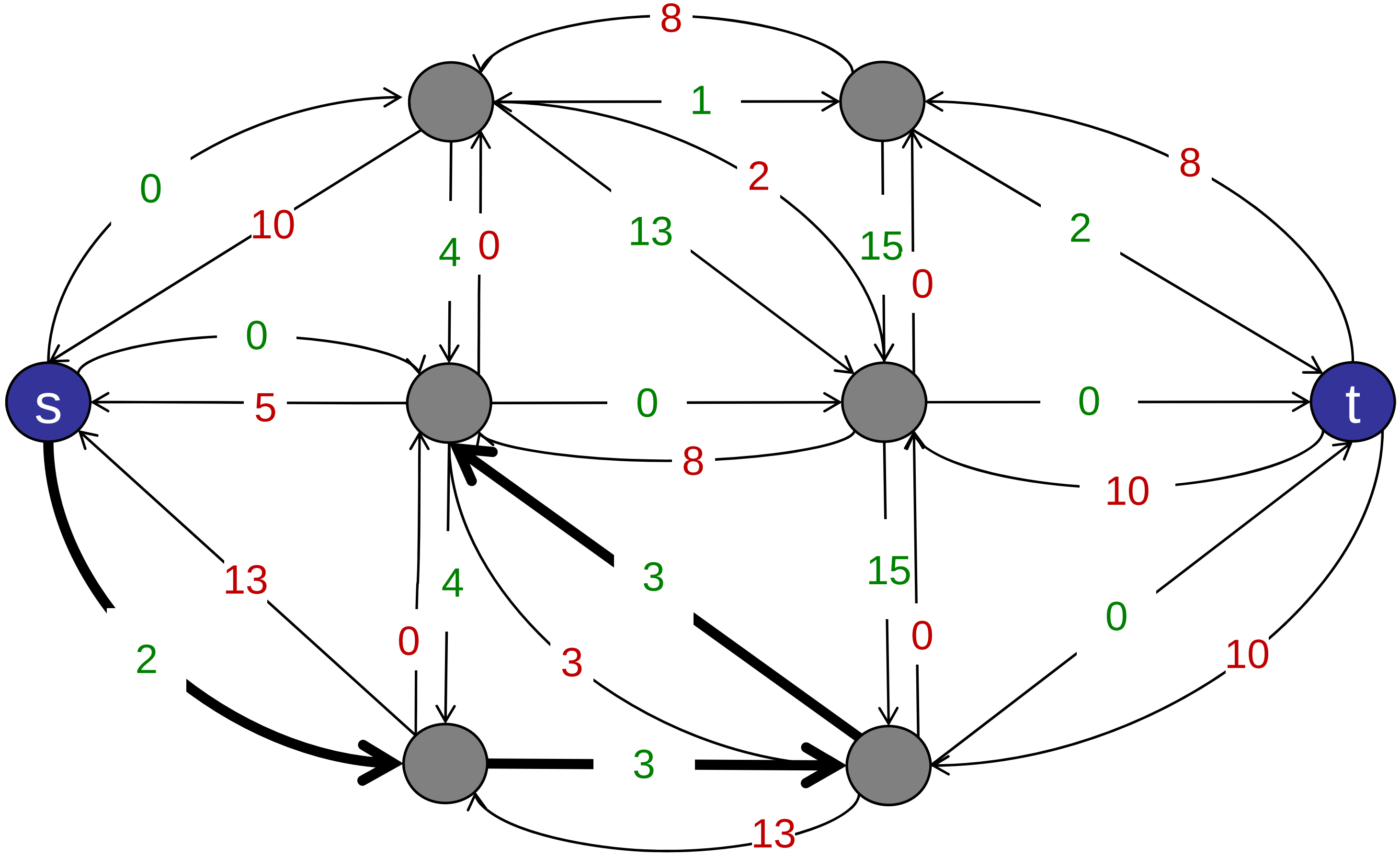
How to find a min-cut:

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.

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)

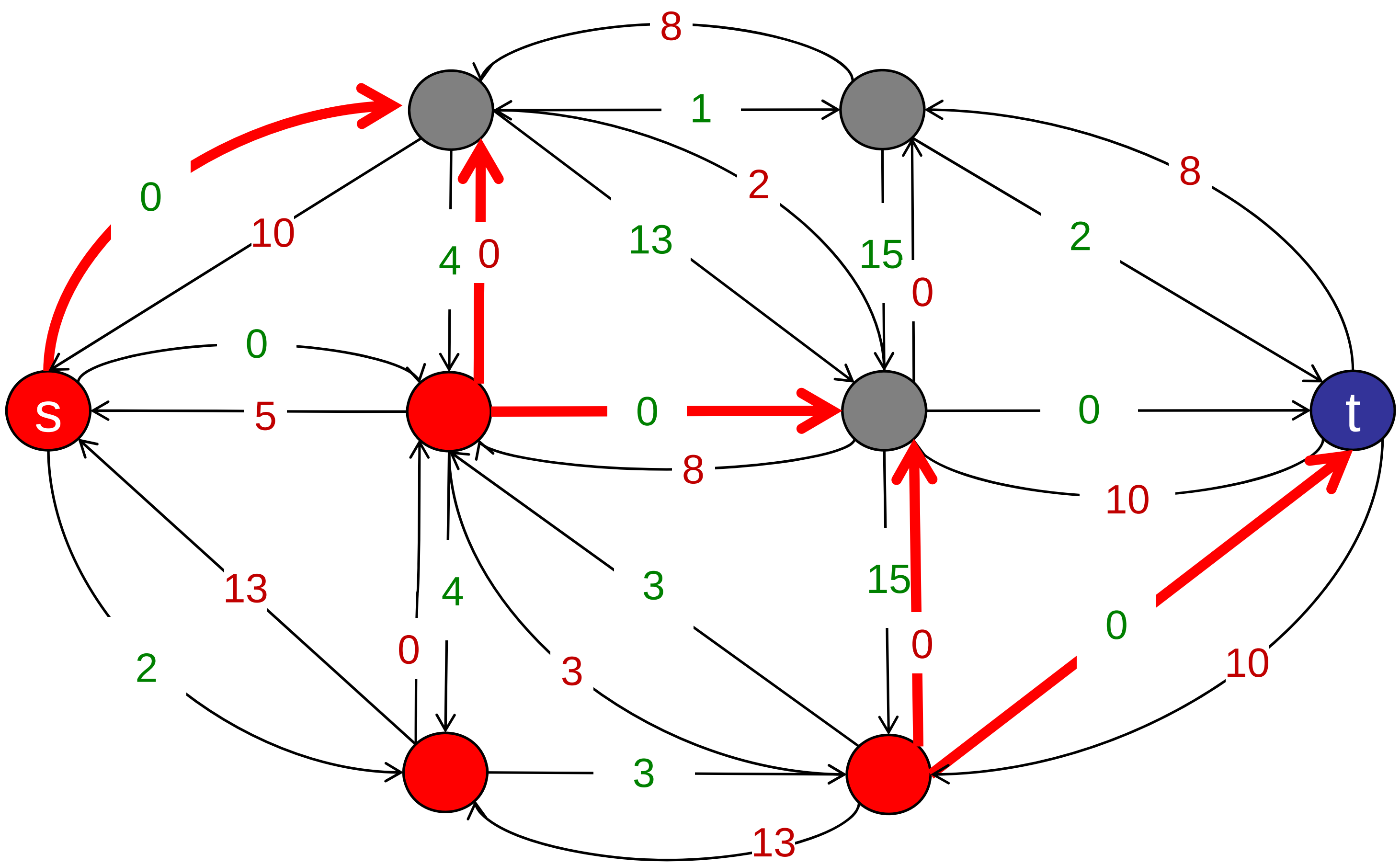


Forward flow = reverse residual flow.

Finding an Augmenting Path

Residual Graph: amount that flow can be increased

residual(e) = capacity(e) – flow(e)



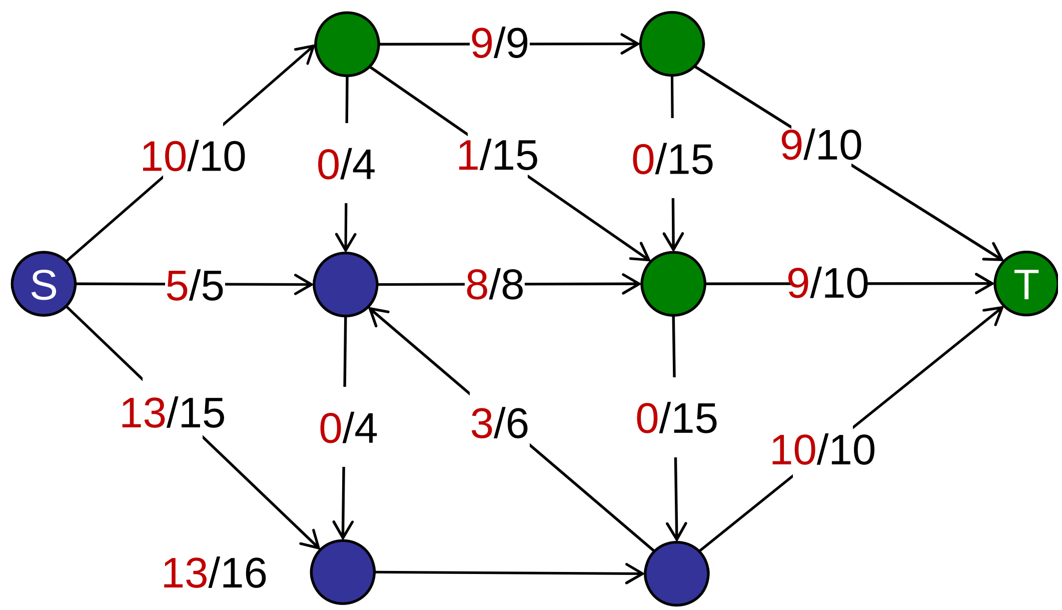
Forward flow = reverse residual flow.

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.

Summary:

- ✓ How to find an augmenting path? The bottleneck capacity?
- ✓ If it terminates, does it always find a max-flow?
- ✓ How fast is Ford-Fulkerson? Can we do better?

A brief history of max flow…

year	method	performance	discovered
1951	simplex	$O(E^3F)$	Dantzig
1955	augmenting path	$O(EF)$	Ford-Fulkerson
1970	shortest augmenting path	$O(VE^2)$	Dinitz, Edmonds-Karp
1972	fattest augmenting path	$O(E^2 \log V \log (F))$	Dinitz, Edmonds-Karp
1986	push-relabel	$O(V^2E)$, $O(V^3)$, $O(VE \log(V))$	
1994		$O(VE \log_{E/V\log V} V)$	King-Rao-Tarjan
2006	$O(\min\{V^{2/3},E^{1/2}\}E \log ((V^2/E + 2)\log F)$		Goldberg-Rao
2012	Combo	$O(VE)$	Orlin, KRT

Clickers

Friday

- Return clickers at the end of class!
- Don’t forget!
- Missing / unreturned clickers cost $105!!

Roadmap

Network Flows

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