Chapter 10

Network Flow

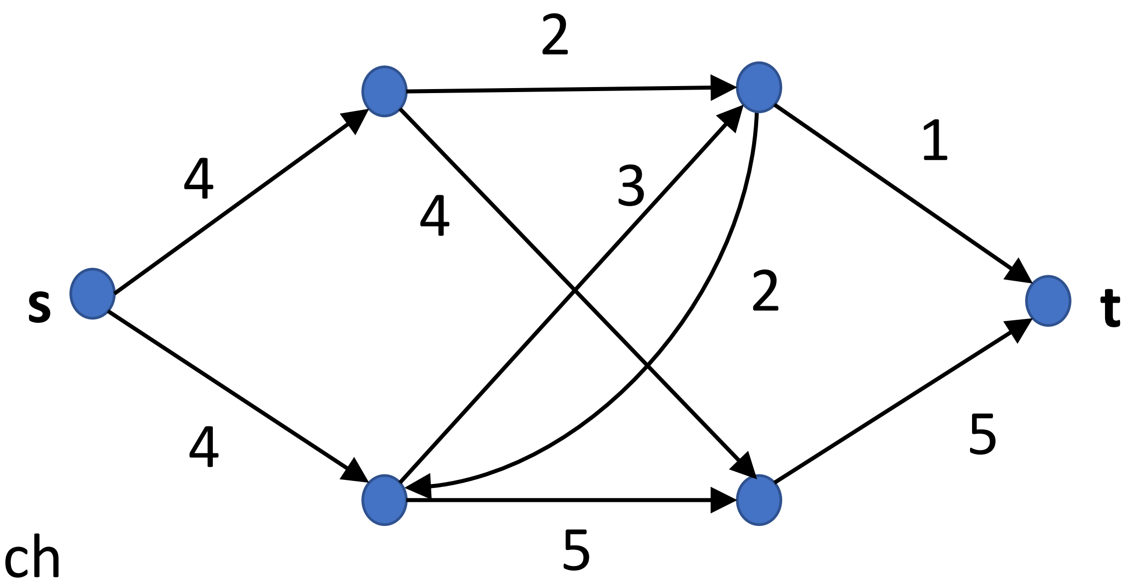
Flow networks

Use directed graphs to model **flow**.

Examples:

- road traffic
- water pipes
- evacuation plans

Each edge in the graph has **capacity**: how much "traffic" can flow through the edge in a unit of time.



Two special vertices

- **source** vertex **s** where all flows originate
- **sink** vertex **t** with only incoming flows

Question: How much traffic can flow from **s** to **t** without overflowing?

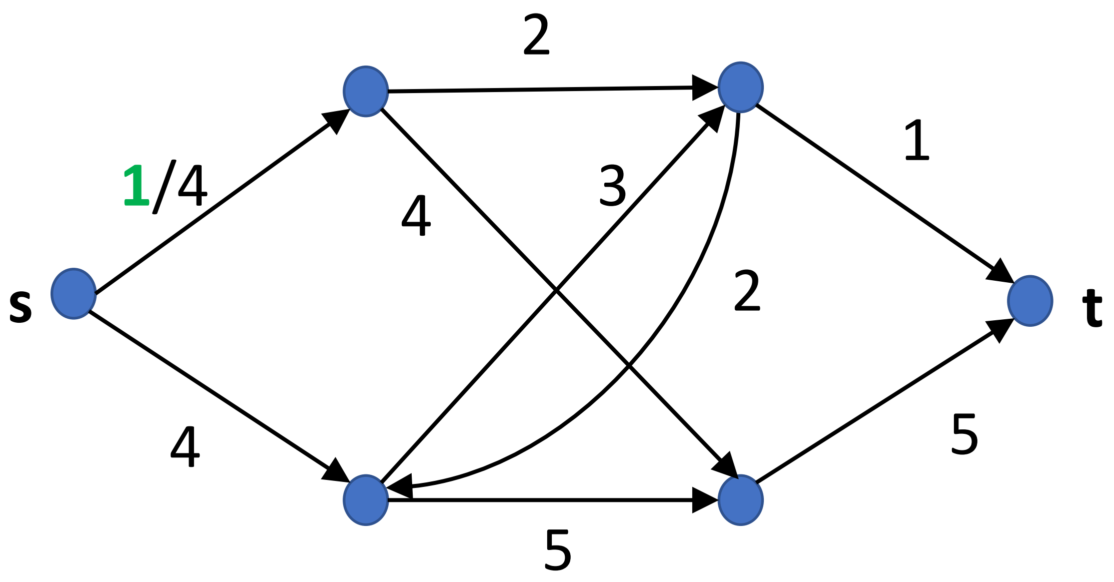
Flow networks and flows

Directed graph $G = (V, E)$ where
each edge $e = (u, v)$ has **capacity** $c_e \geq 0$.

Flow maps each edge e to a real number f_e.

Capacity constraints:
each edge e can admit a flow f_e where

$$f_e \leq c_e.$$



Flow networks and flows

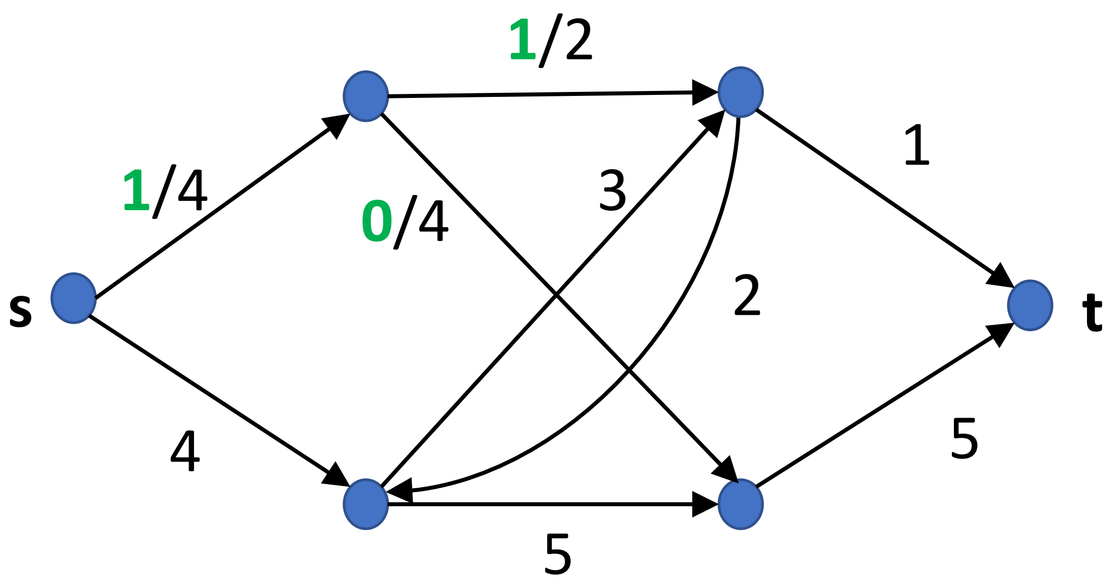
Directed graph $G = (V, E)$ where
each edge $e = (u, v)$ has **capacity** c_e.

Flow maps each edge e to a real number f_e.

Capacity constraints:
each edge e can admit a flow f_e where

$$f_e \leq c_e.$$

Conservation constraints:
for each vertex v such that $v \neq$ **s** and $v \neq$ **t**
total incoming flow = total outgoing flow



Flow networks and flows

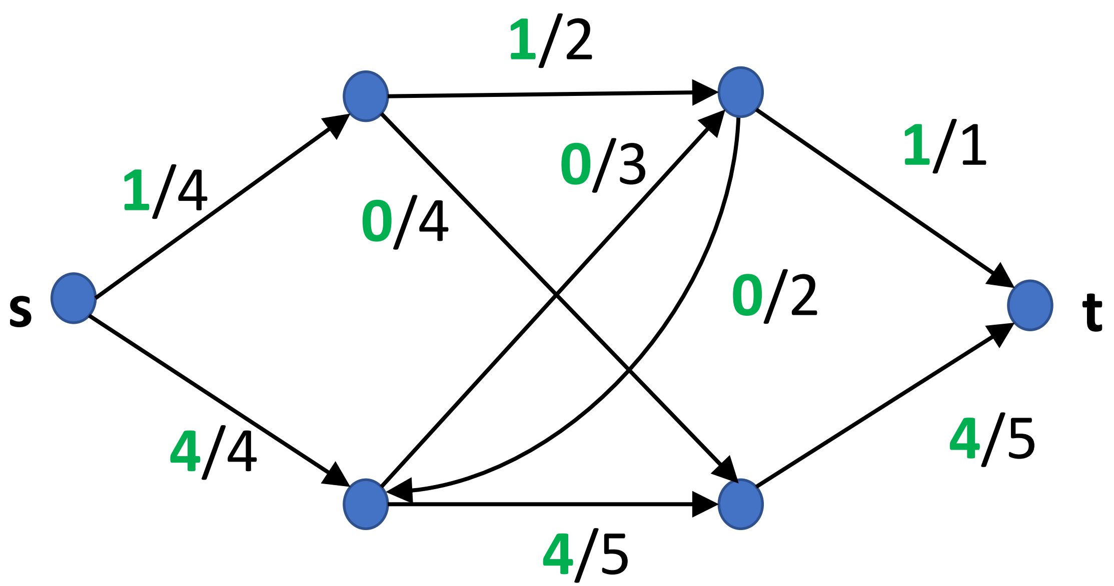
Directed graph $G = (V, E)$ where
each edge $e = (u, v)$ has **capacity** $c_e \geq 0$.

Flow maps each edge e to a real number f_e.

Capacity constraints:
each edge e can admit a flow f_e where

$$f_e \leq c_e.$$

Conservation constraints:
for each vertex v such that $v \neq$ **s** and $v \neq$ **t**
total incoming flow = total outgoing flow

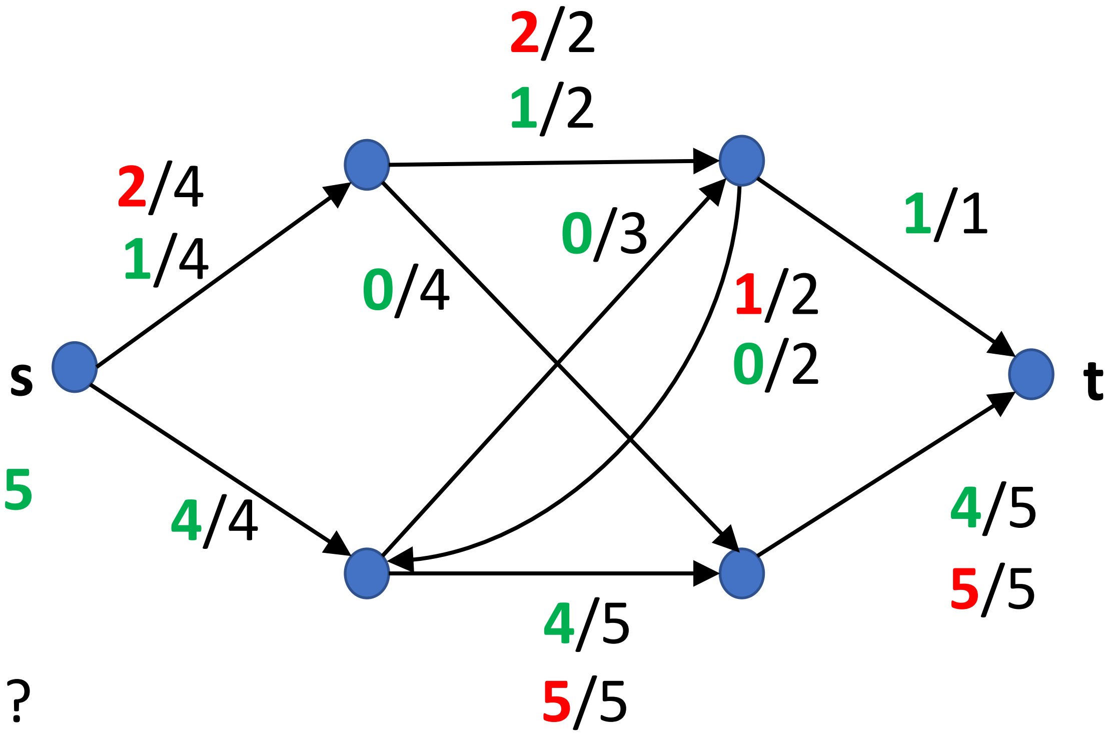


Flow: satisfies all capacity and conservation constraints in G

Flow value

Value of a flow: the total amount of flow that goes from **s** to **t**.

flow value = 5



flow value = 6

Is this value maximal?

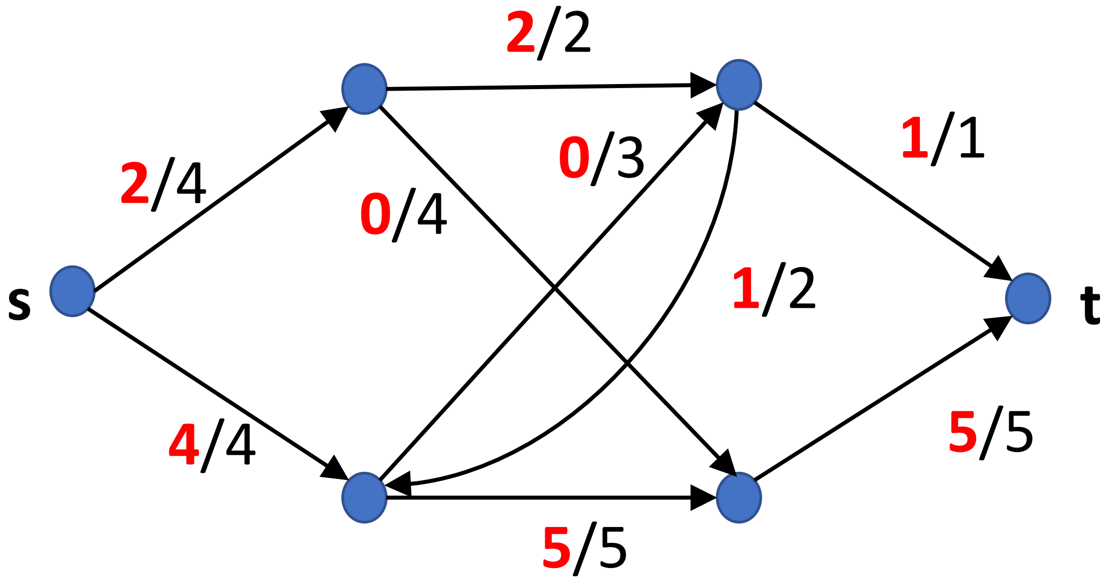
Is 6 the maximal flow value?

Maximum flow problem

Given a flow network, find the **maximum flow**.

We want to find:

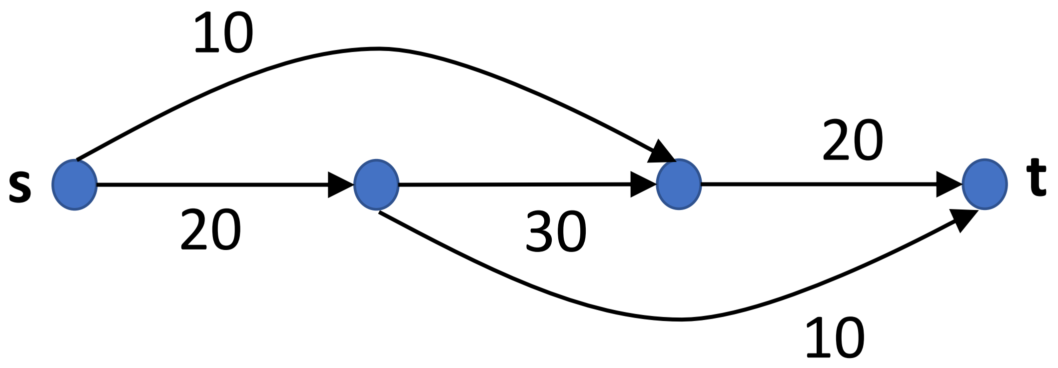
- the maximum flow value and
- the actual flow for each edge



First attempt: a greedy algorithm

Idea: Increase the flow on the edges incrementally

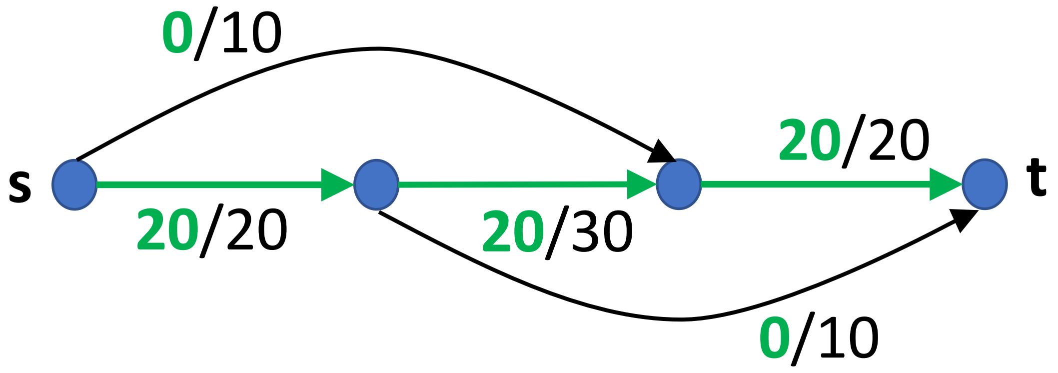
1. Start with flow 0 everywhere
2. Find a path from **s** to **t** with positive remaining capacity
3. Assign some flow on this path and go to 2



First attempt: a greedy algorithm

Idea: increase the flow on the edges incrementally

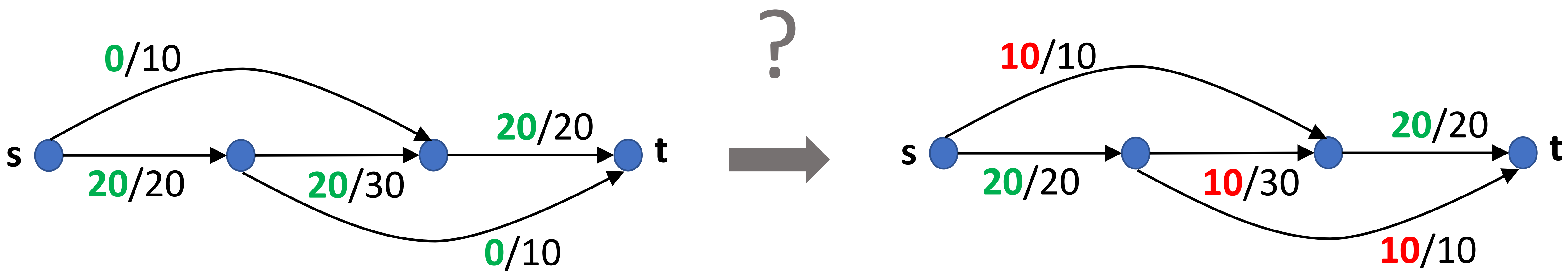
1. Start with flow 0 everywhere
2. Find a path from **s** to **t** with positive remaining capacity
3. Assign some flow on this path and go to 2



no path from **s** to **t** where the flow can be improved

Is this flow maximal?

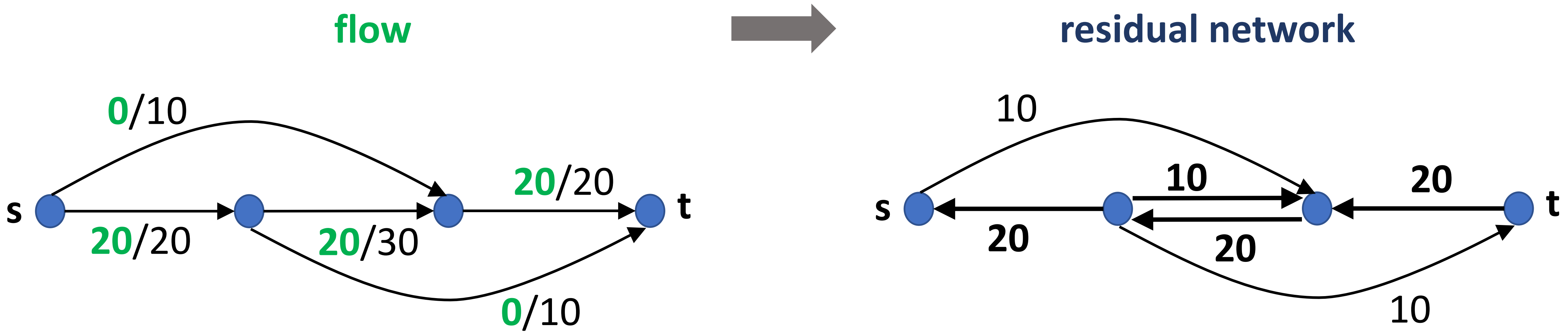
First attempt: a greedy algorithm



We need to **reduce** the flow on the middle edge.

Idea: Push flow in **reverse direction** to ”undo” assigned flow.

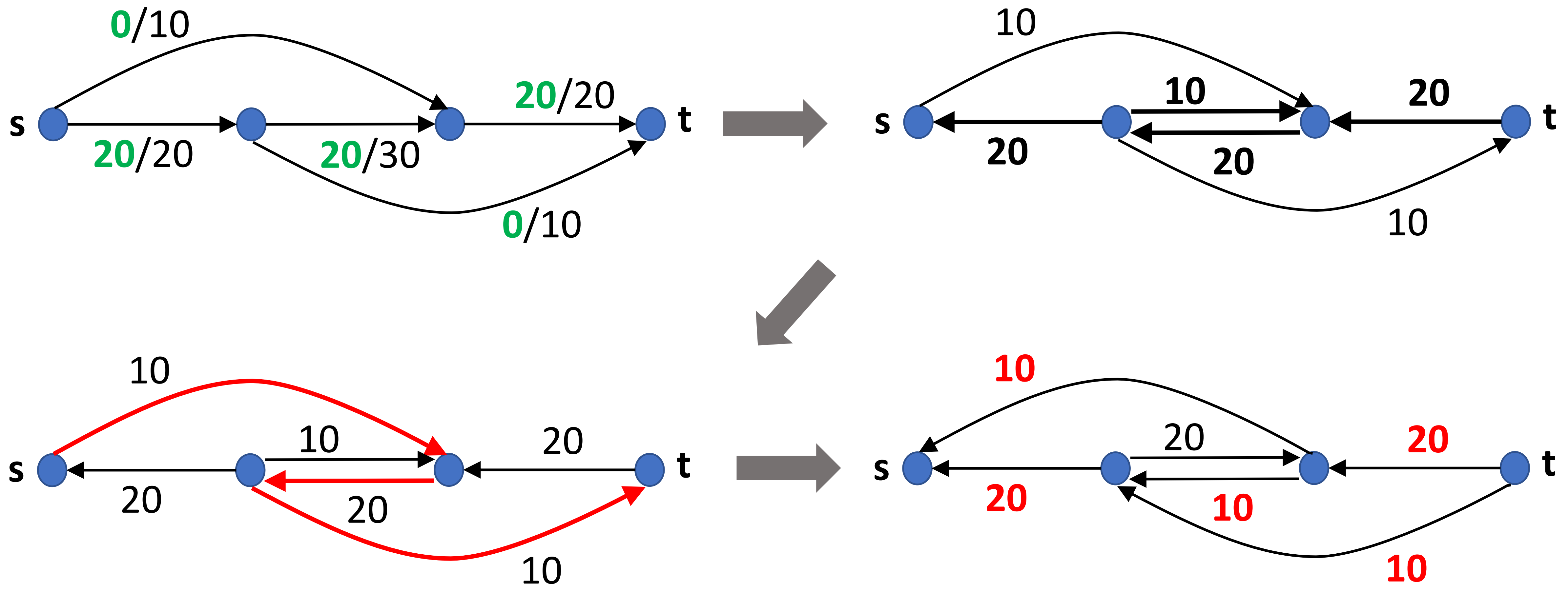
How to “not get stuck”? Residual networks



Same vertices as original network. For each edge (u, v) with flow f and capacity c add:

- edge (u, v) with capacity $c - f$ (the remaining, i.e., **residual capacity**),
- edge (v, u) with capacity f (the already assigned flow)

The example revisited



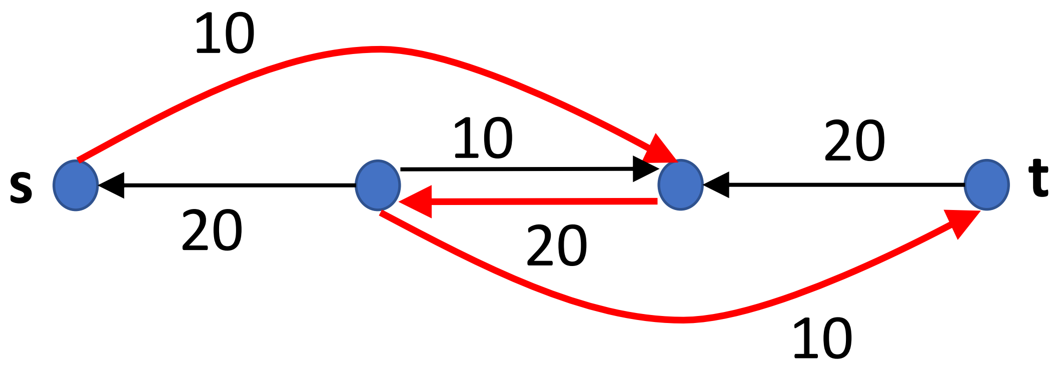
path from s to t with positive remaining capacity

Augmenting paths

Augmenting path p: a path from **s** to **t** in the residual network for some flow.

Can increase the flow by adding flow to the edges on the augmenting path.

The amount of increase is constrained by the minimum capacity of edge in p.



augmenting path, flow = 10

Iteratively increase the flow until no more augmenting paths can be found.

Ford-Fulkerson algorithm

set flow f_e to 0 for each edge e

construct the residual network

while there is an augmenting path $p = (e_1, e_2, \dots)$ **do**

  increase the flow on each edge in p by $\min\{c(e_1), c(e_2), \dots\}$

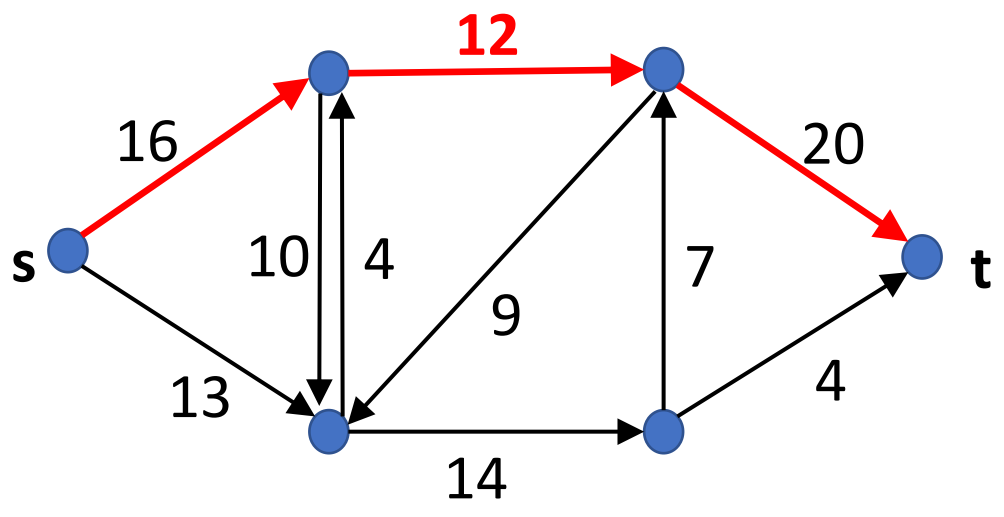
  update the residual network

end while

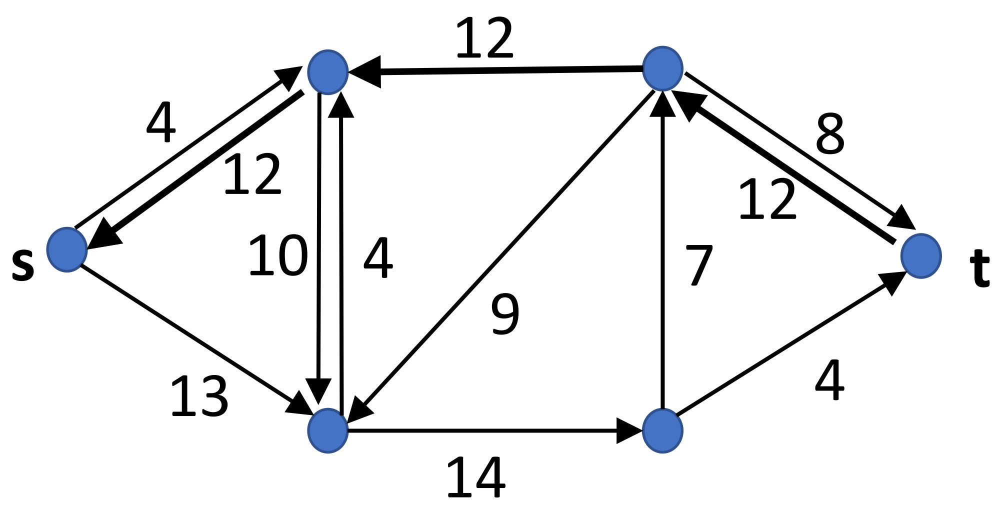
- Not specified which augmenting path to use.
- Search for augmenting path (for example BFS).

Correctness: When there is no augmenting path, the flow is maximal.

Example: Ford-Fulkerson algorithm

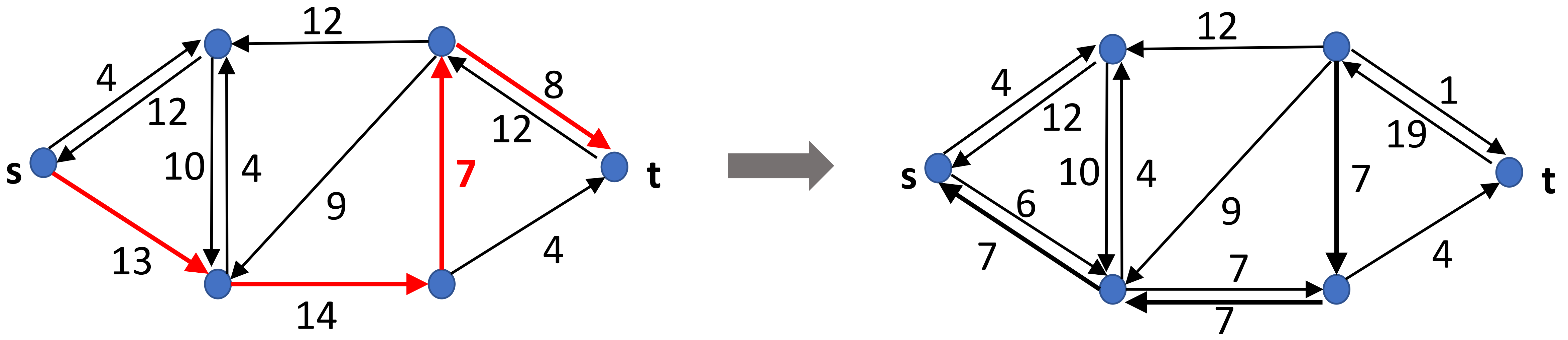


augmenting path, flow = 12



updated residual network

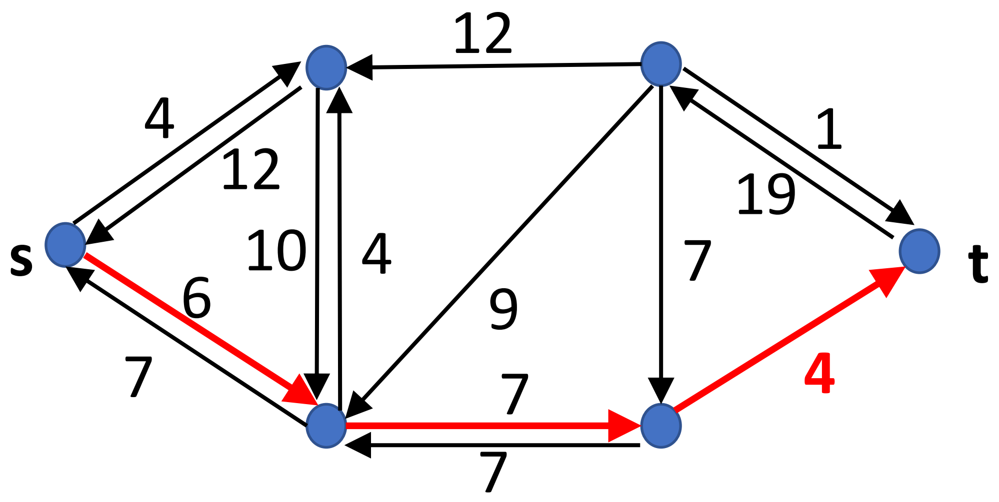
Example: Ford-Fulkerson algorithm



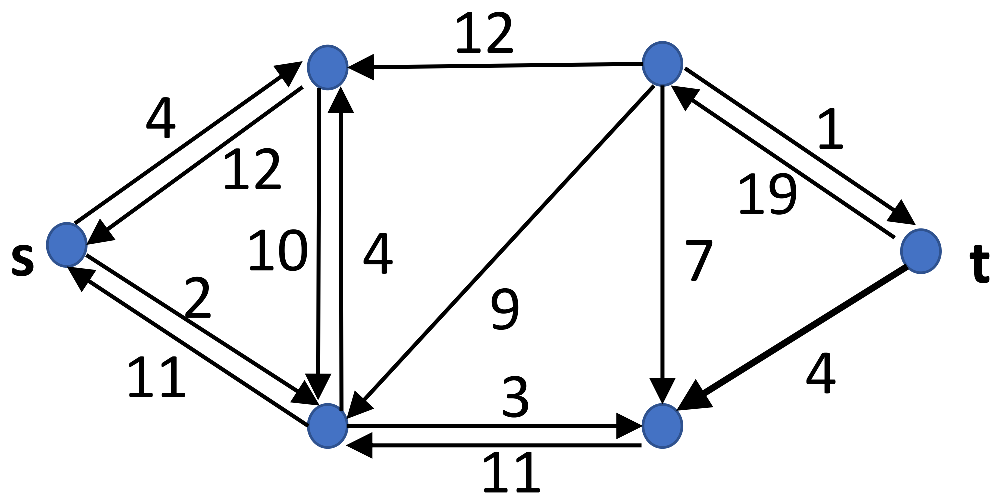
augmenting path, flow = 7

updated residual network

Example: Ford-Fulkerson algorithm

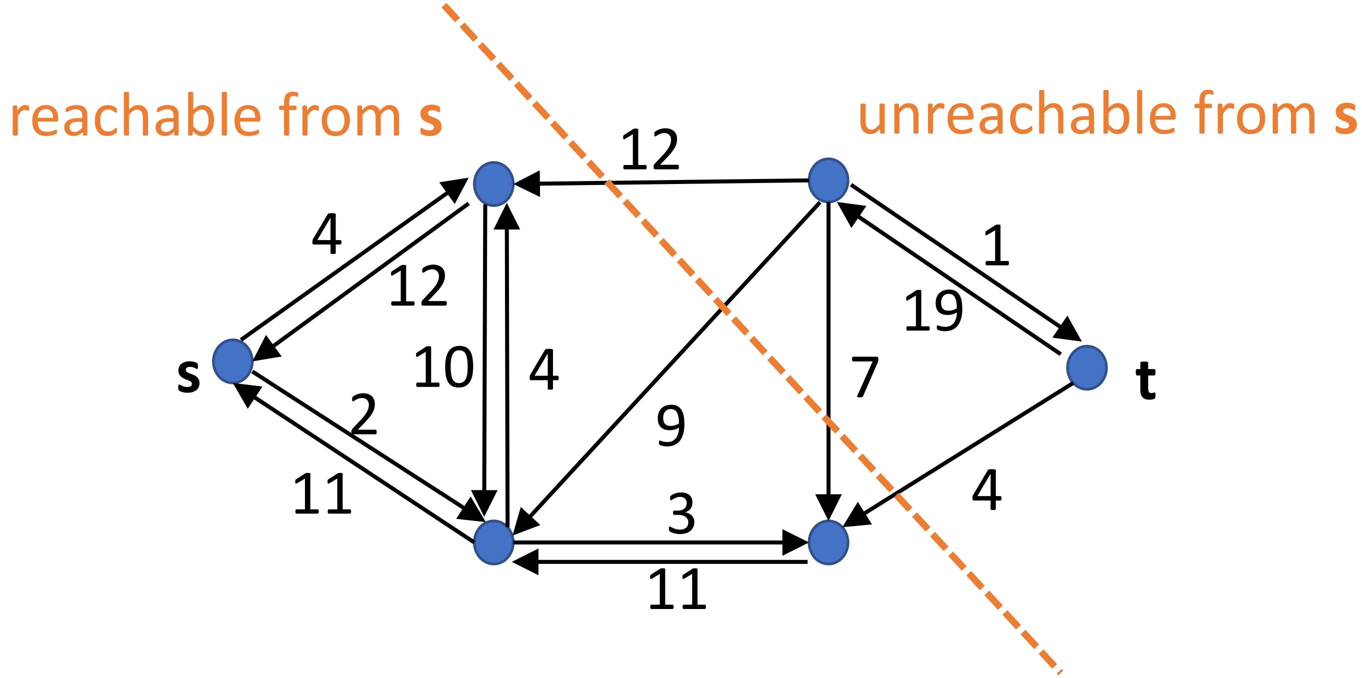


augmenting path, flow = 4



updated residual network

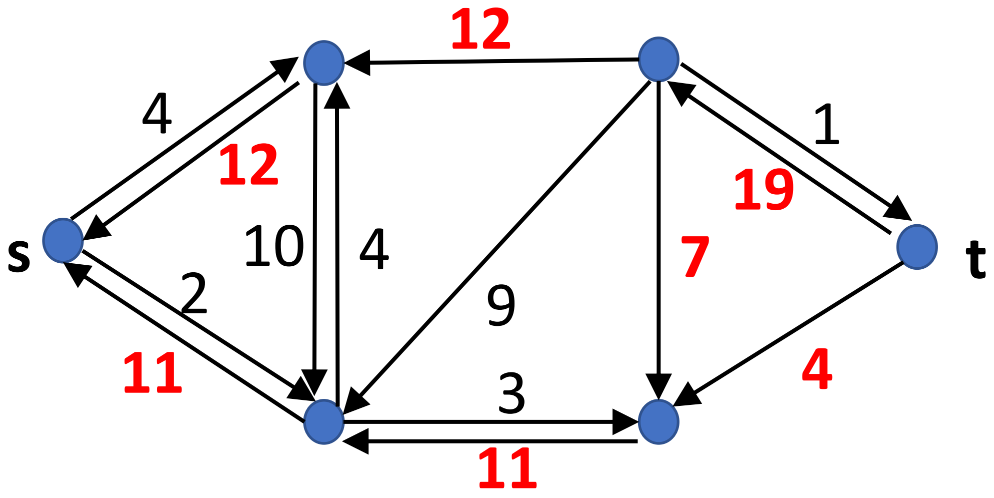
Example: Ford-Fulkerson algorithm



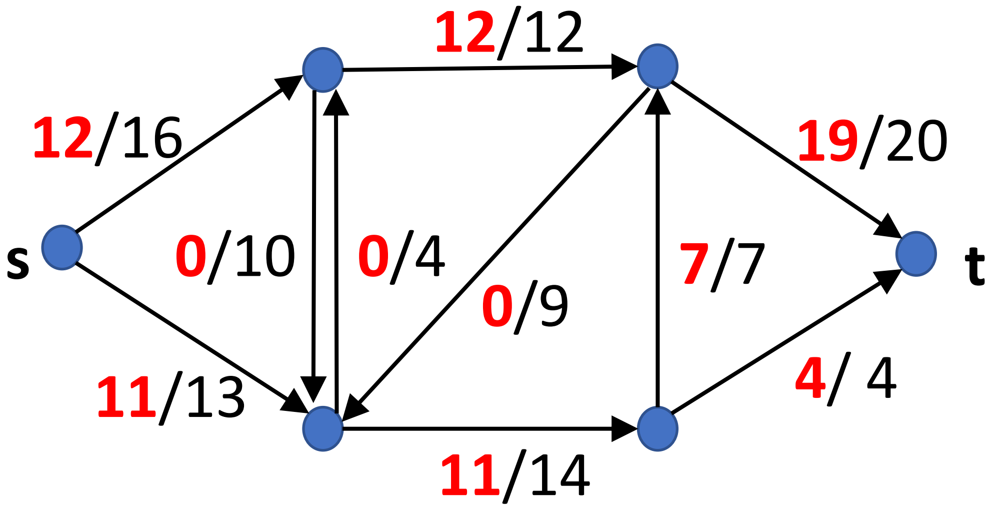
no more augmenting paths

Example: Ford-Fulkerson algorithm

We obtain the final flow from the final residual network.



final flow value = 23



Running time of the Ford-Fulkerson algorithm

Assumption: All edge capacities are integers.

Integrality Theorem: If all edge capacities are integers, then there is a maximum flow in which the flow assigned to each edge is also an integer.

- If all $c(e_1), c(e_2), \ldots$ are integers, then so is $\min\{c(e_1), c(e_2), \ldots\}$.
- The flow increase and the residual capacities are then also integers.

Idea: Use the integrality to bound the number of iterations of the algorithm.

Running time of the Ford-Fulkerson algorithm

set flow f_e to 0 for each edge e, construct the residual network $\boldsymbol{O(m)}$ **time**

while there is an augmenting path $p = (e_1, e_2, \dots)$ **do**

  finding one augmenting path takes $\boldsymbol{O(m+n) = O(m)}$ time

  increase the flow on each edge in p by $\min\{c(e_1), c(e_2), \dots\}$ $\boldsymbol{O(m)}$ **time**

  update the residual network $\boldsymbol{O(m)}$ **time**

end while

For an augmenting path $p = (e_1, e_2, \dots)$, we have $\min\{c(e_1), c(e_2), \dots\} \geq 1$.

At each iteration, the flow value increases by at least 1.

This means **at most $\boldsymbol{F}$ iterations**, where F is the maximum flow.

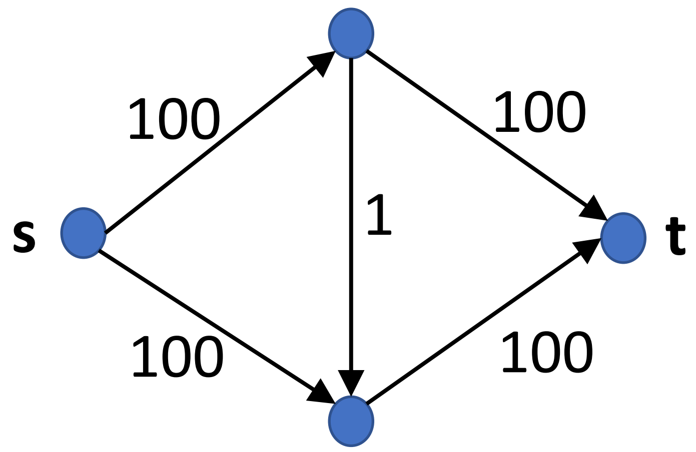
Running time of the Ford-Fulkerson algorithm

The running time is $\boldsymbol{O(mF)}$, where F is the computed maximum flow value.

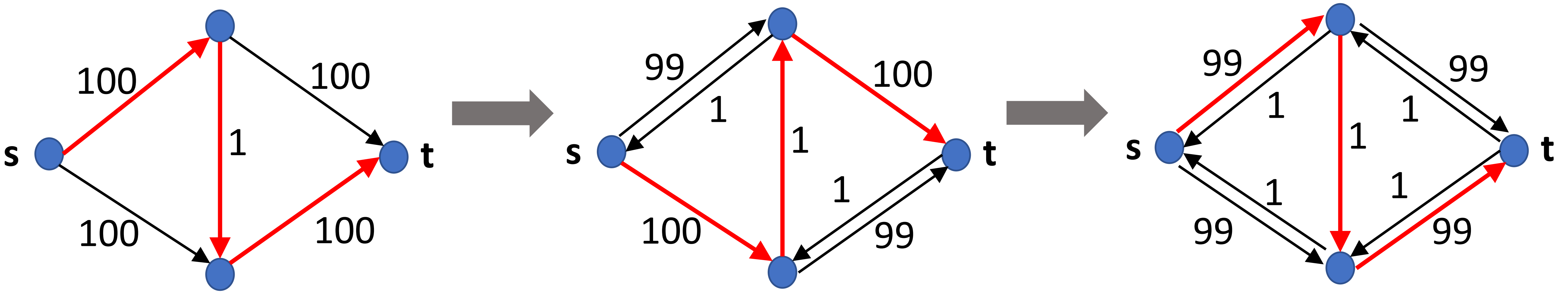
- **output-sensitive**: depends on the output value

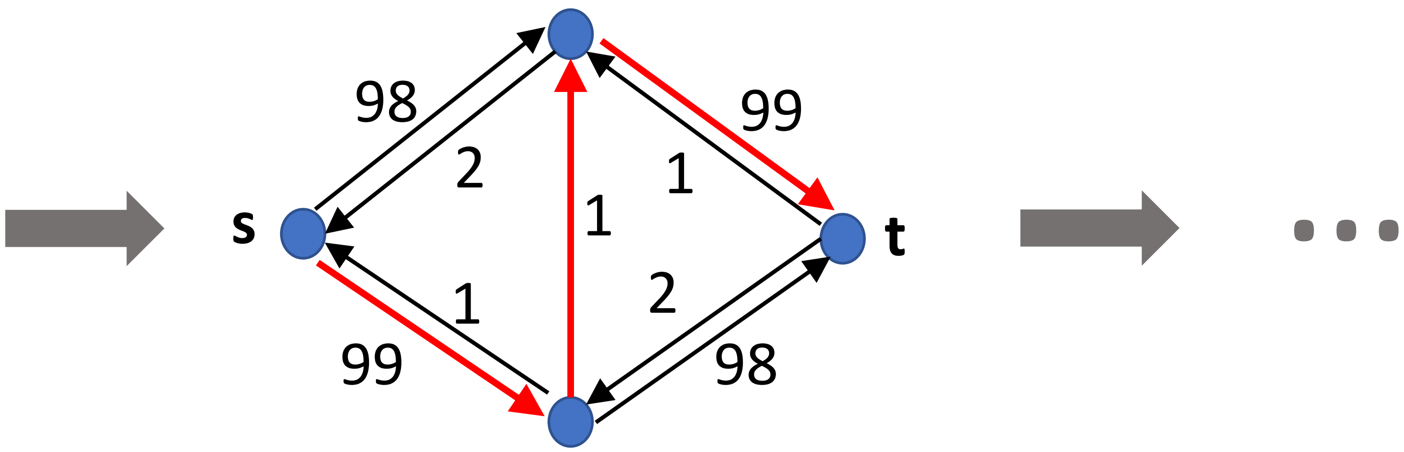
If we multiply all capacities by 100, the time complexity increases 100 times.

The time complexity can actually be that bad!



A particularly bad choice of augmenting paths





200 iterations to complete!

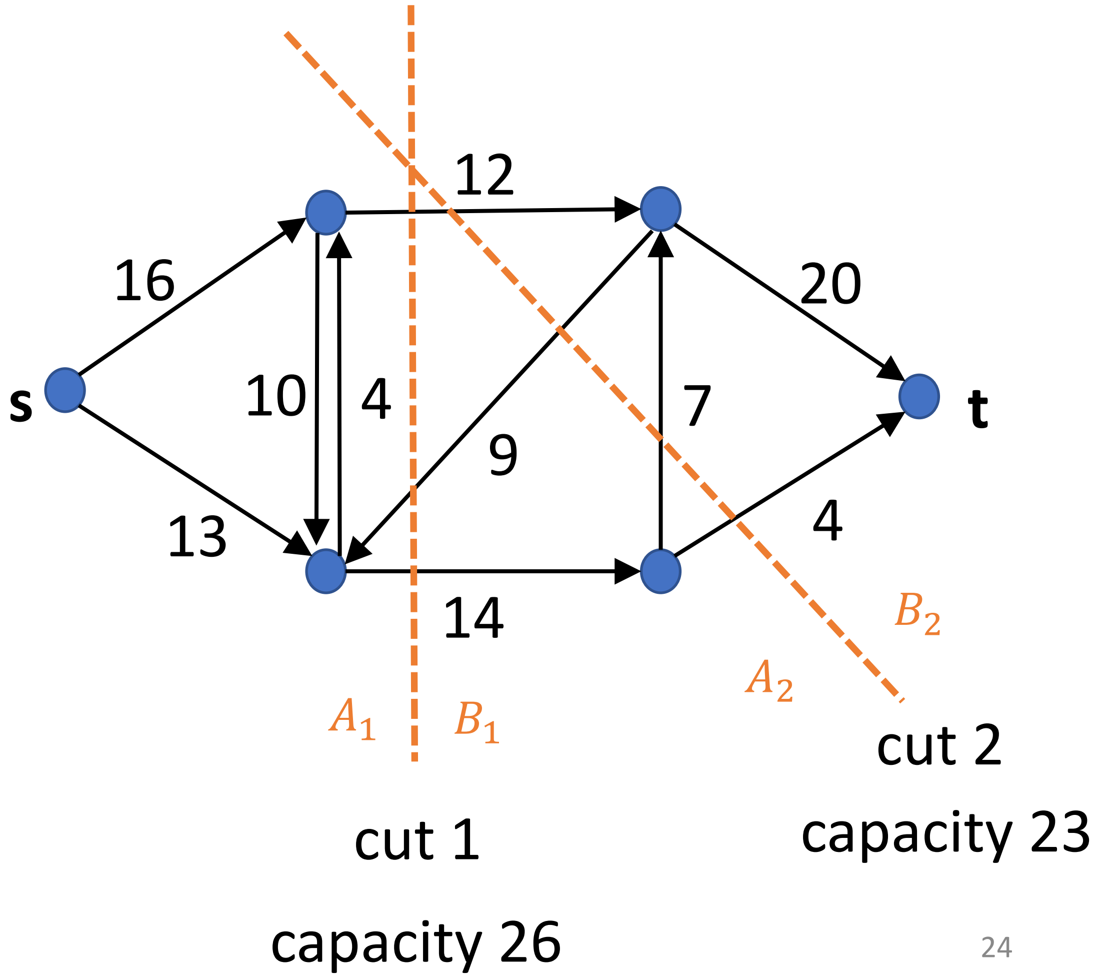
s-t cuts

We did not prove that the algorithm computes the maximum flow. Can be proven using a characterization of the maximum flow value in terms of **cuts** in the flow network.

An **s-t cut** is a partition (A, B) of the set of vertices such that

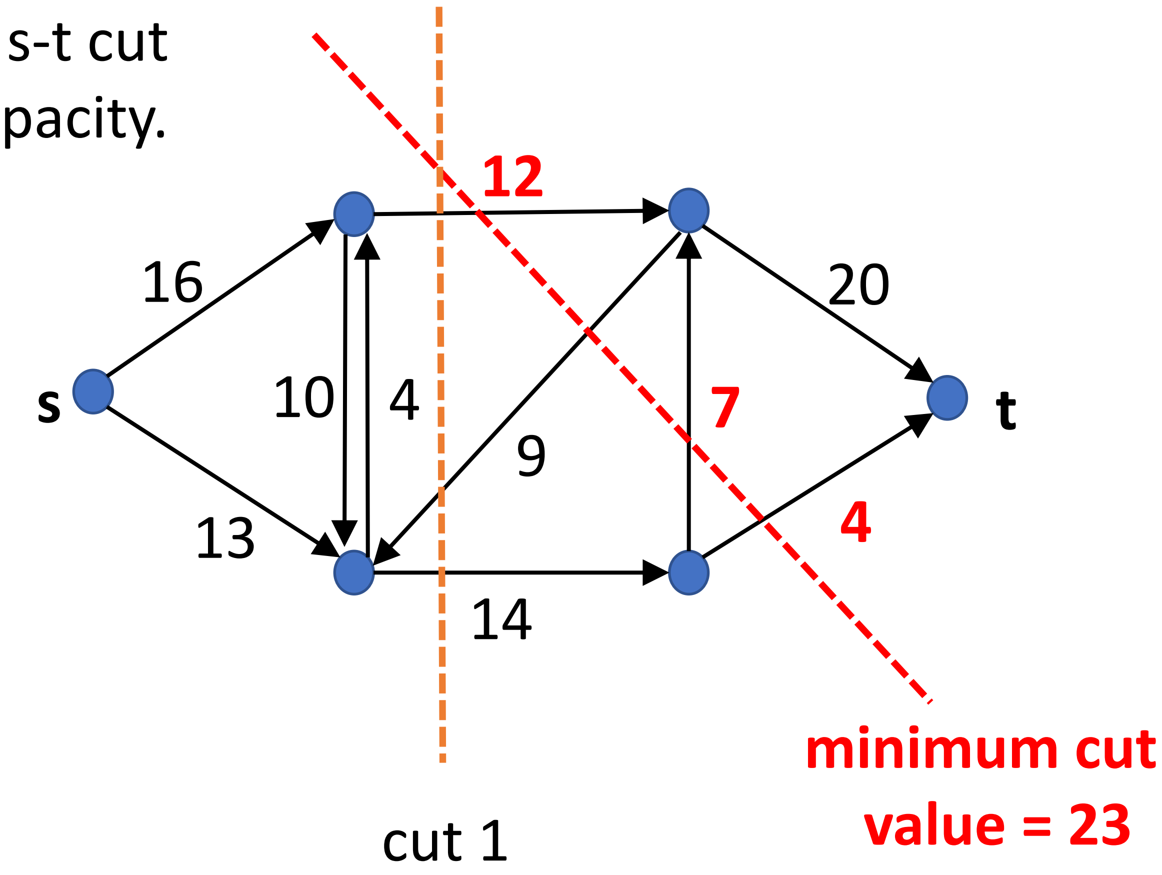
- s $\in A$
- t $\in B$

The **capacity** of a cut is the total capacity of the edges crossing the cut from A to B.



Minimum cuts

A **minimum cut** is an s-t cut that has minimum capacity.



maximum flow value = 23

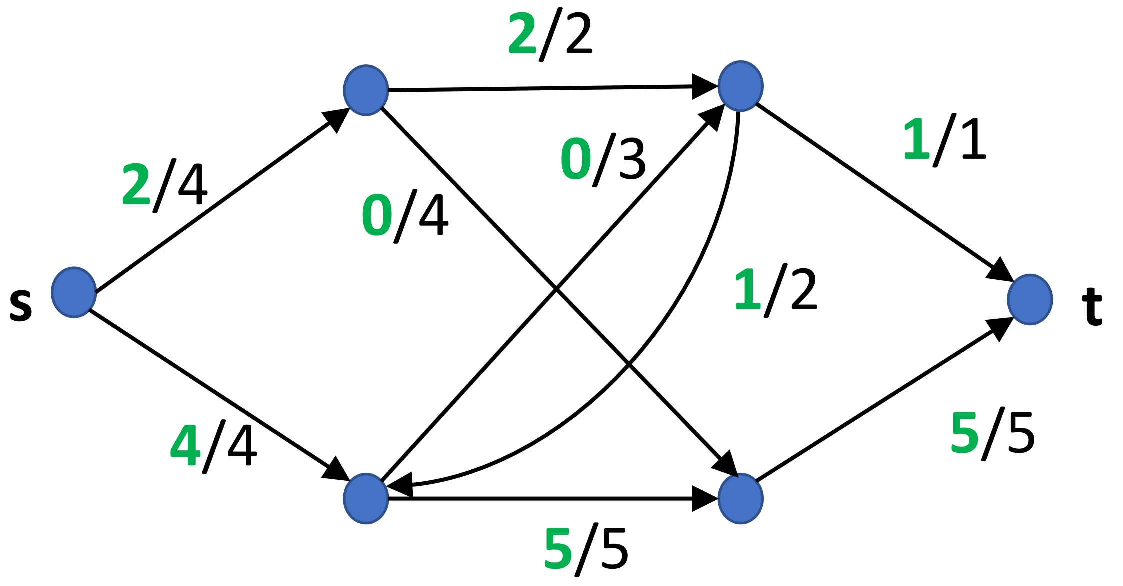
Chapter 10

Network Flow

Part 2

Recap: Flow networks and flows

Directed graph $G = (V, E)$ where
each edge $e = (u, v)$ has **capacity** $c_e \geq 0$.



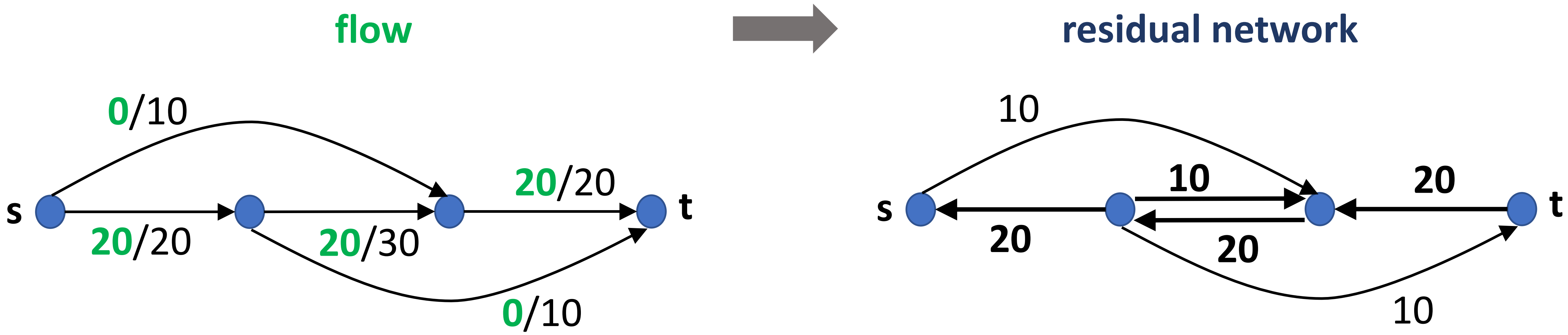
Flow: maps each edge e to a real number f_e
and satisfies the following constraints:
Capacity constraints: each edge e can admit
a flow f_e where $f_e \leq c_e$.
Conservation constraints:
for each vertex v such that $v \neq$ **s** and $v \neq$ **t**
total incoming flow = total outgoing flow

flow value = 6

Value of a flow: the total amount of flow that goes from **s** to **t**.

Maximum flow problem: Given a flow network, find
the maximum flow value, and the actual flow for each edge.

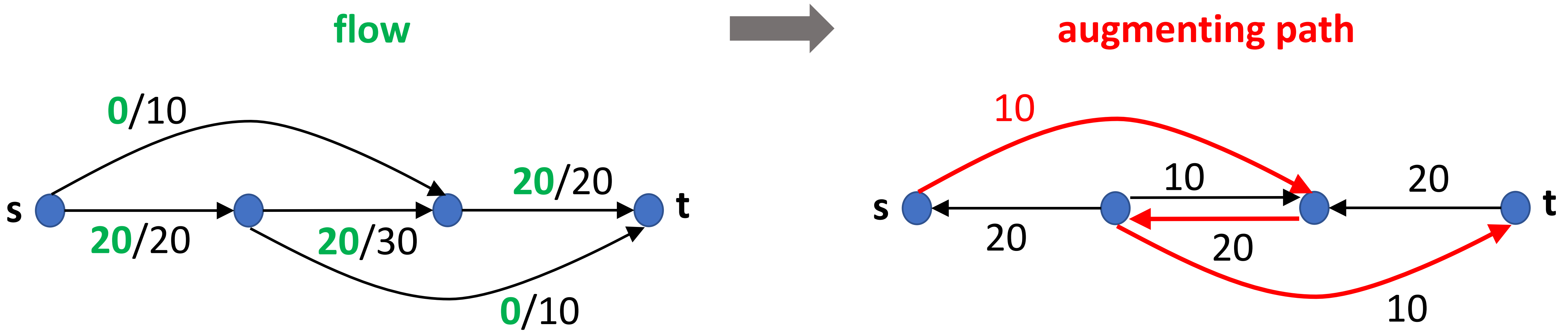
Recap: Residual networks



Same vertices as original network. For each edge (u, v) with flow f and capacity c add:

- edge (u, v) with capacity $c - f$ (the remaining, i.e., **residual capacity**),
- edge (v, u) with capacity f (the already assigned flow)

Recap: Augmenting paths



Augmenting path p: a path from **s** to **t** in the residual network for some flow.

Can increase the flow by adding flow to the edges on the augmenting path.

Recap: Ford-Fulkerson algorithm

set flow f_e to 0 for each edge e

construct the residual network

while there is an augmenting path $p = (e_1, e_2, \dots)$ **do**

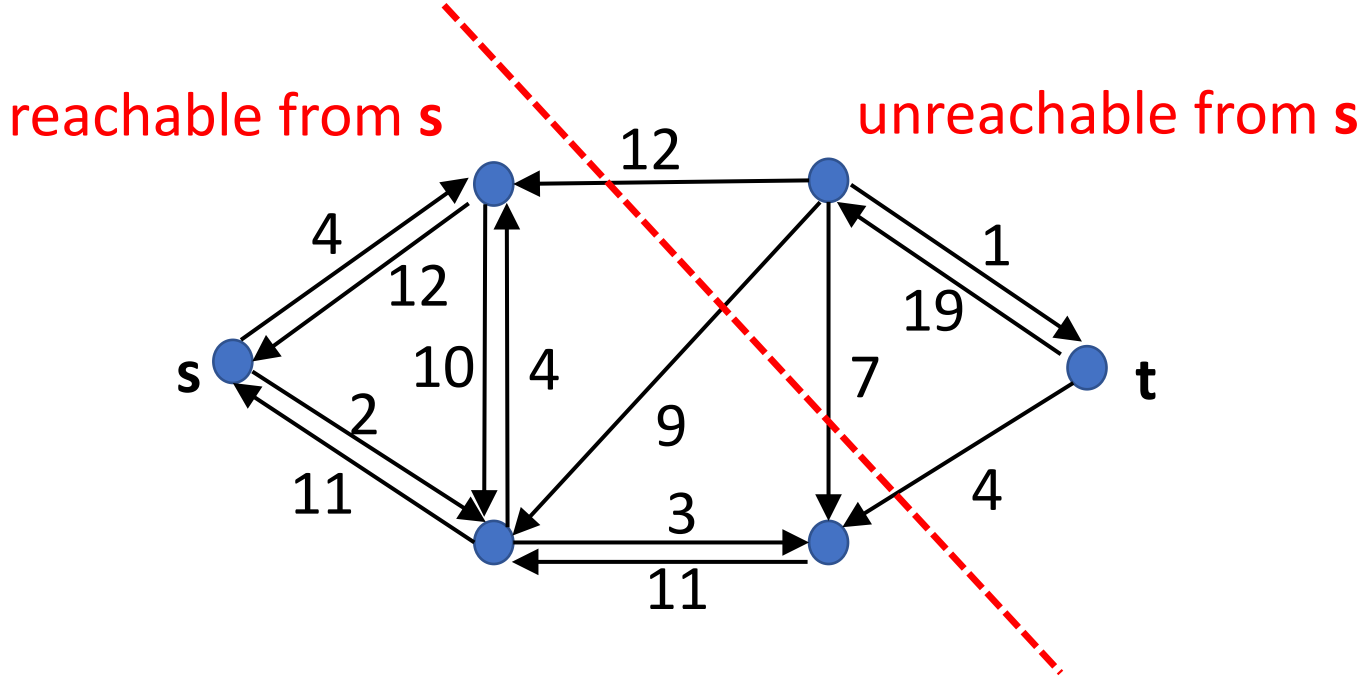
  increase the flow on each edge in p by $\min\{c(e_1), c(e_2), \dots\}$

  update the residual network

end while

When there is no augmenting path, the flow is maximal.

Recap: Ford-Fulkerson algorithm



no more augmenting paths

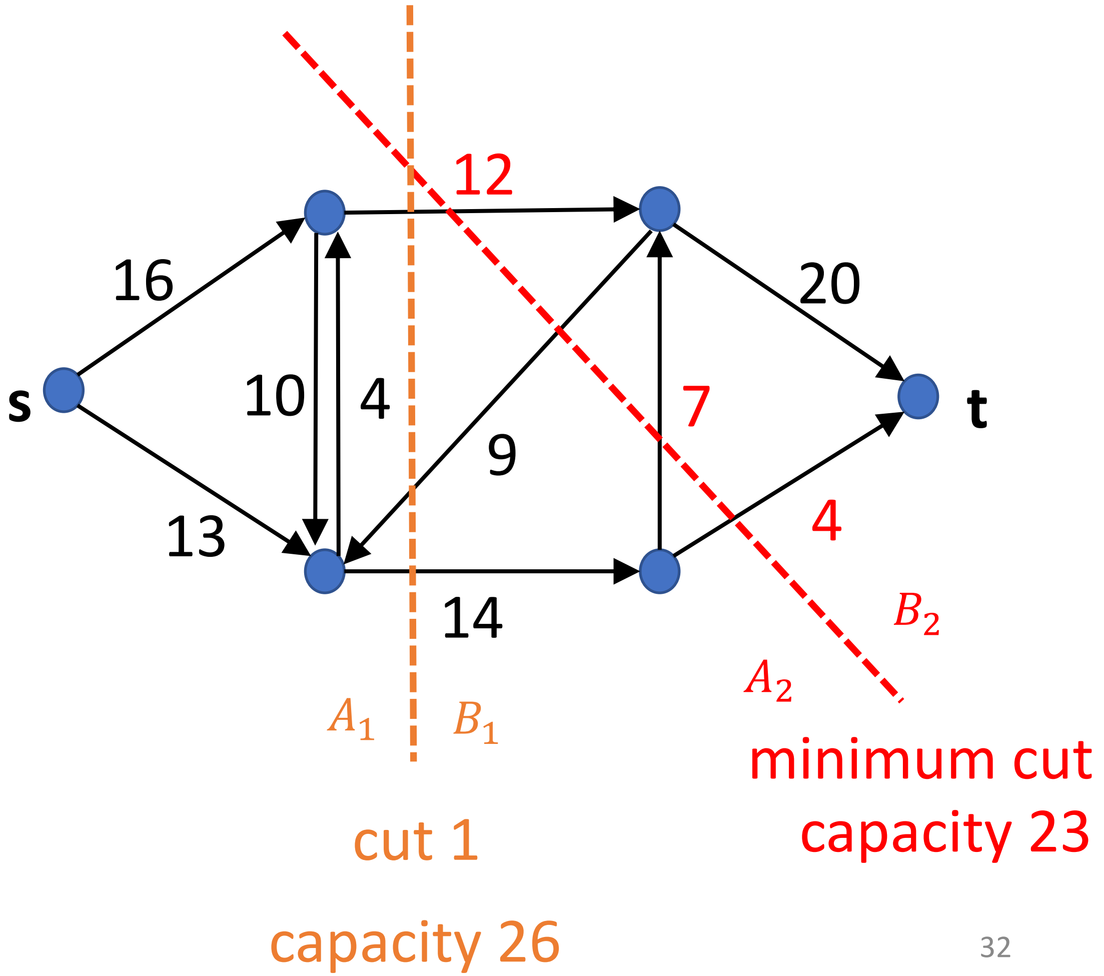
Recap: minimum cuts

An **s-t cut** is a partition (A, B) of the set of vertices such that

- $\mathbf{s} \in A$
- $\mathbf{t} \in B$

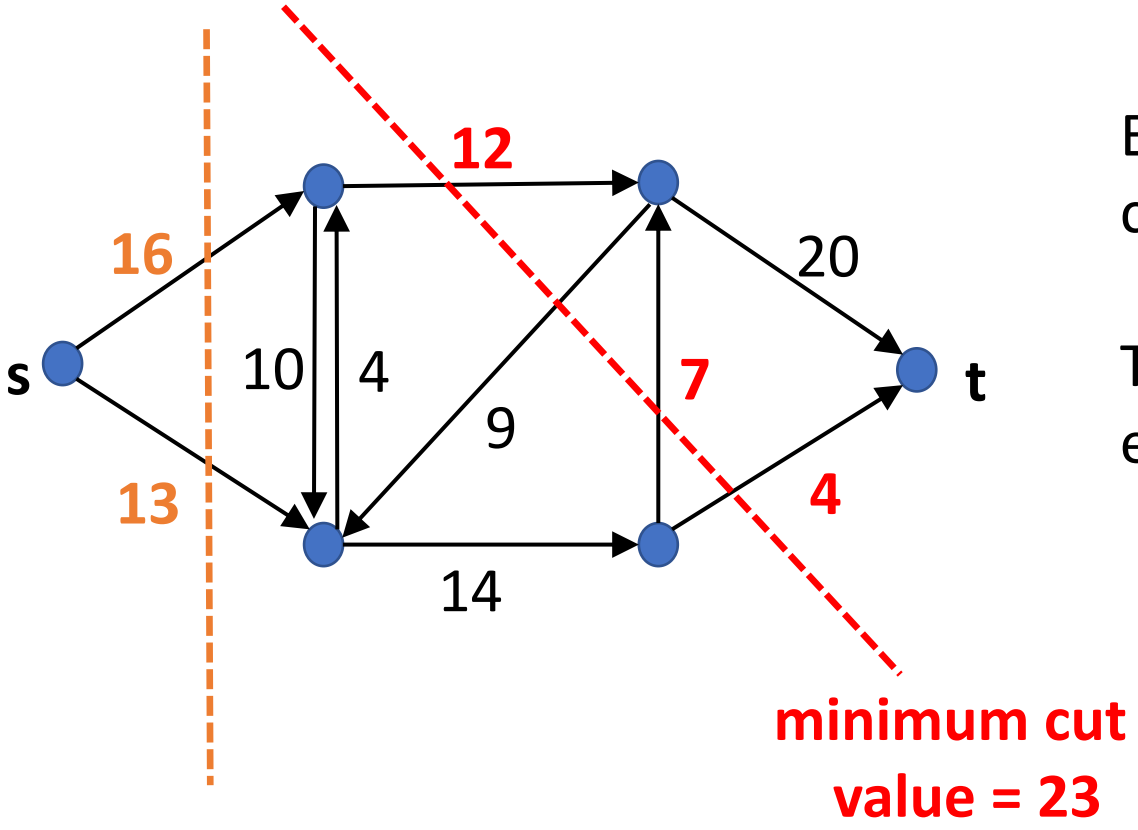
The **capacity** of a cut is the total capacity of the edges crossing the cut from A to B.

A **minimum cut** is an s-t cut that has minimum capacity.



Max flow = Min cut

Max-flow-min-cut Theorem:
The maximum flow value of a network is the same as the value of the minimum cut.



Every flow must pass through each cut, from the **s** side to the **t** side.

$\Longrightarrow$

The maximum flow is less than or equal to the value of any cut.

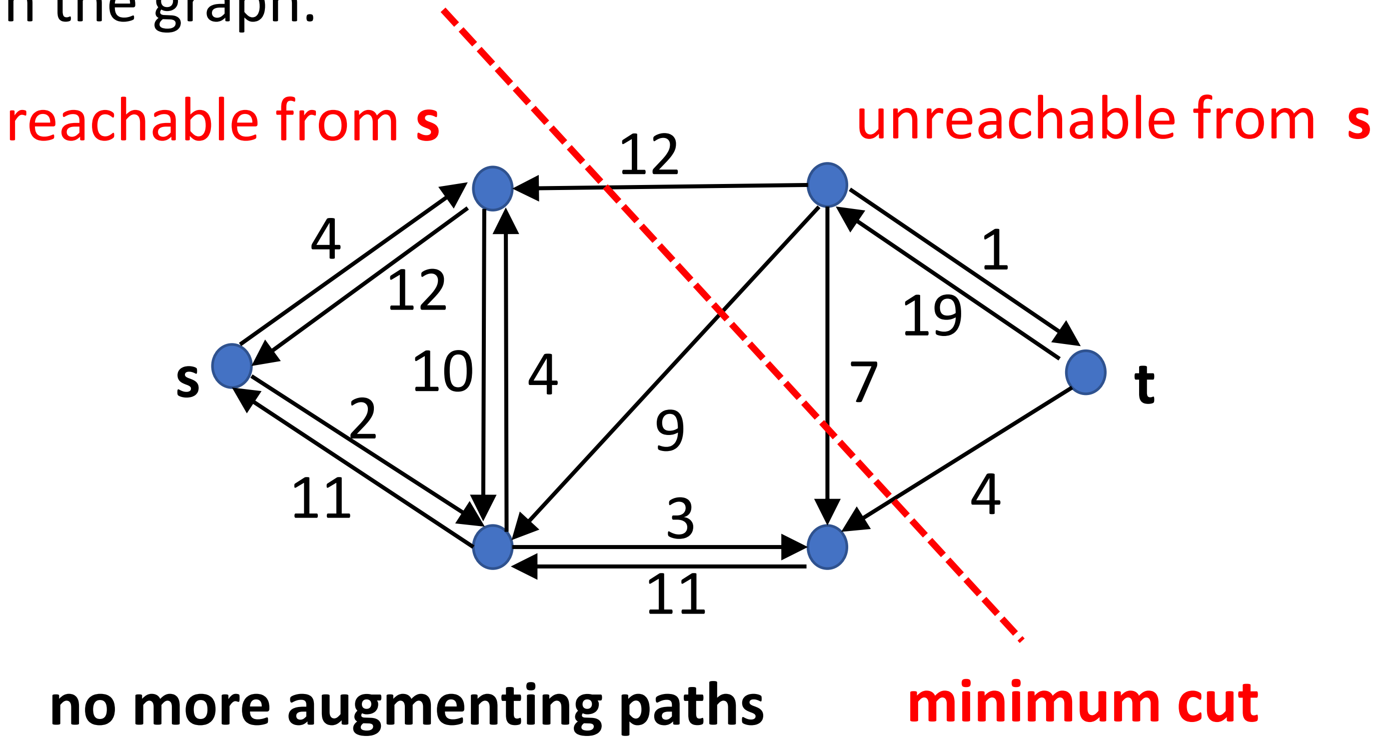
$\Longrightarrow$

max flow value $\leq$ min cut value

Computing a minimum cut

Using the Ford-Fulkerson algorithm we can compute a minimum cut:

- its numerical value, and
- its location in the graph.

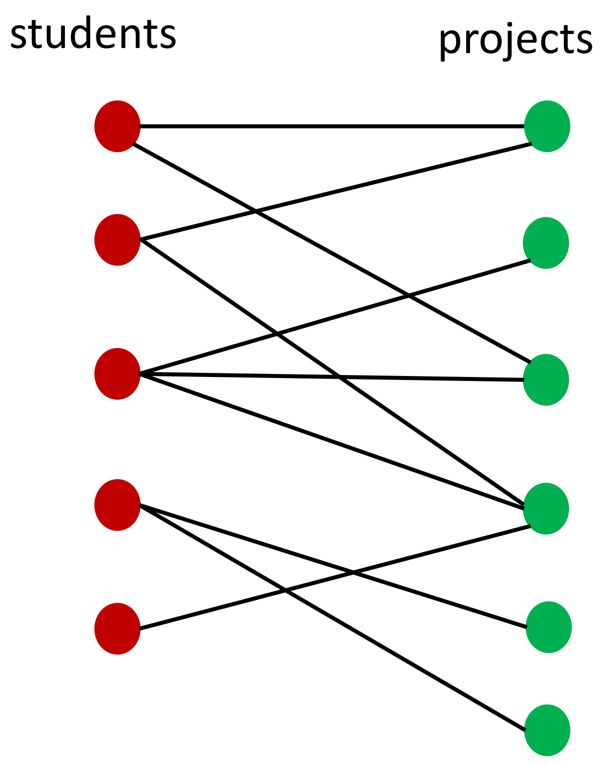


Note: there may be other minimum cuts.

Bipartite Matching

Matching problems

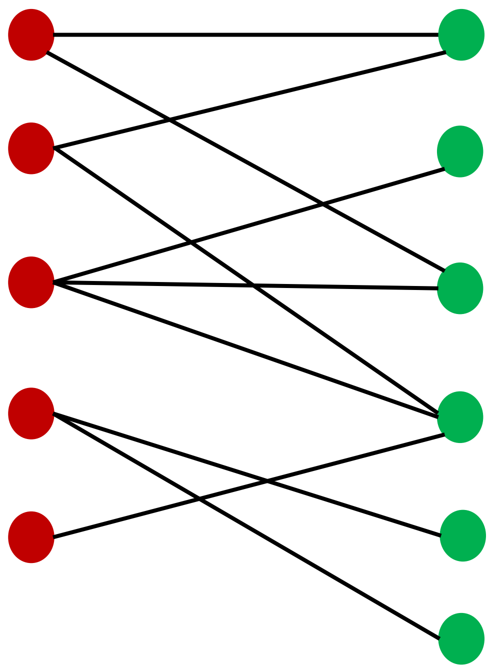
- Two "types" of objects
  - students, projects
- Each object of one type has preference about the objects of the other type.
  - students prefer some projects
- One-to-one mapping between some objects of different types
  - no student assigned two projects
  - no project assigned to two students



Question: Can we match students to projects while respecting their preferences? How?

Bipartite matching

disjoint sets of vertices



edge: objects allowed to match

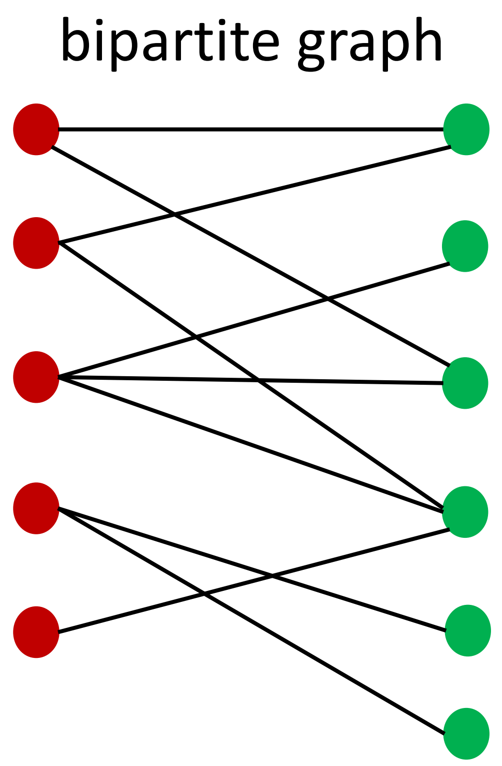
Bipartite graph

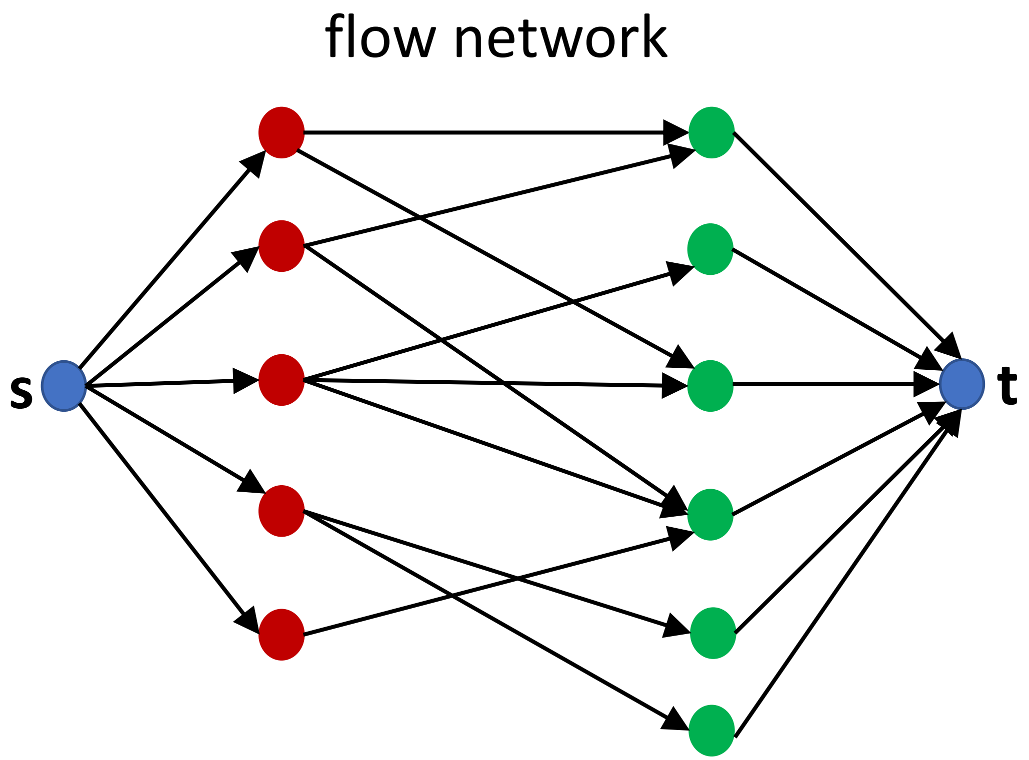
- set of vertices partitioned in two disjoint sets
- all edges of the graph only connect vertices from one of the sets with vertices from the other set

Matching: a set of edges, such that each vertex belongs to at most one edge in the set

Maximum matching: a matching between connected vertices of different types such that the number of matched pairs of vertices is as large as possible

Reducing bipartite matching to network flow



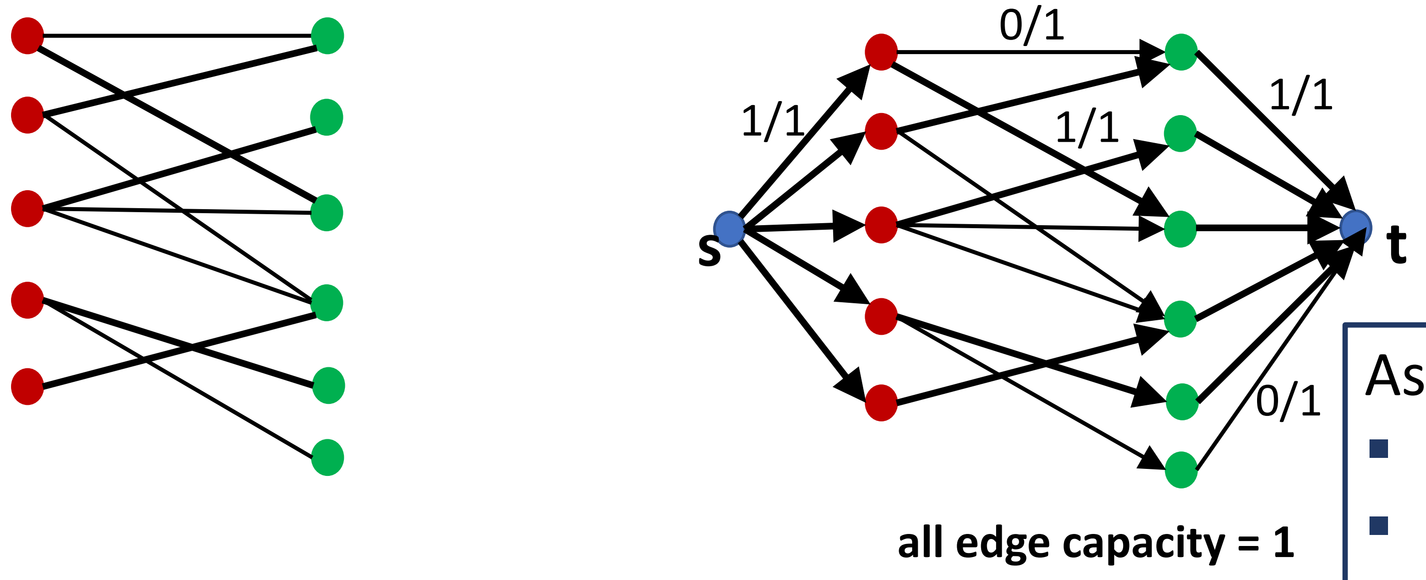


all edge capacity = 1

maximum matching

maximum flow

Reducing bipartite matching to network flow



Assign to edge flow
- 1 if in the matching
- 1 if connected to **s** or **t** and vertex with edge in the matching
- 0 otherwise

A matching of size x always gives a flow of value x

None of the edges in the matching share any vertices.

$\Longrightarrow$

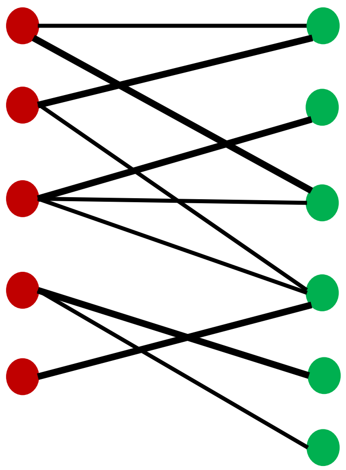
The paths in the flow are disjoint (except at **s** and **t**).

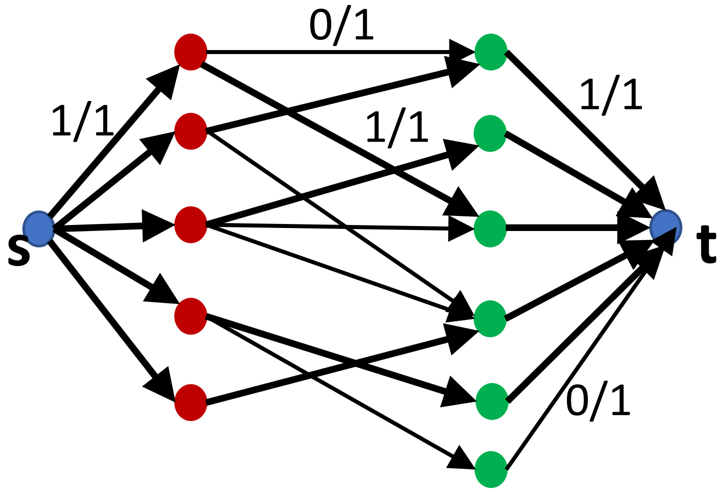
$\Longrightarrow$

The value of the flow is equal to the sum of the path flows (= size of the matching).

Reducing bipartite matching to network flow

Add edge to matching if and only if flow is 1.





all edge capacity = 1

A flow of value x always gives a matching of size x

All edge capacities are 1. Apply **Integrality Theorem**

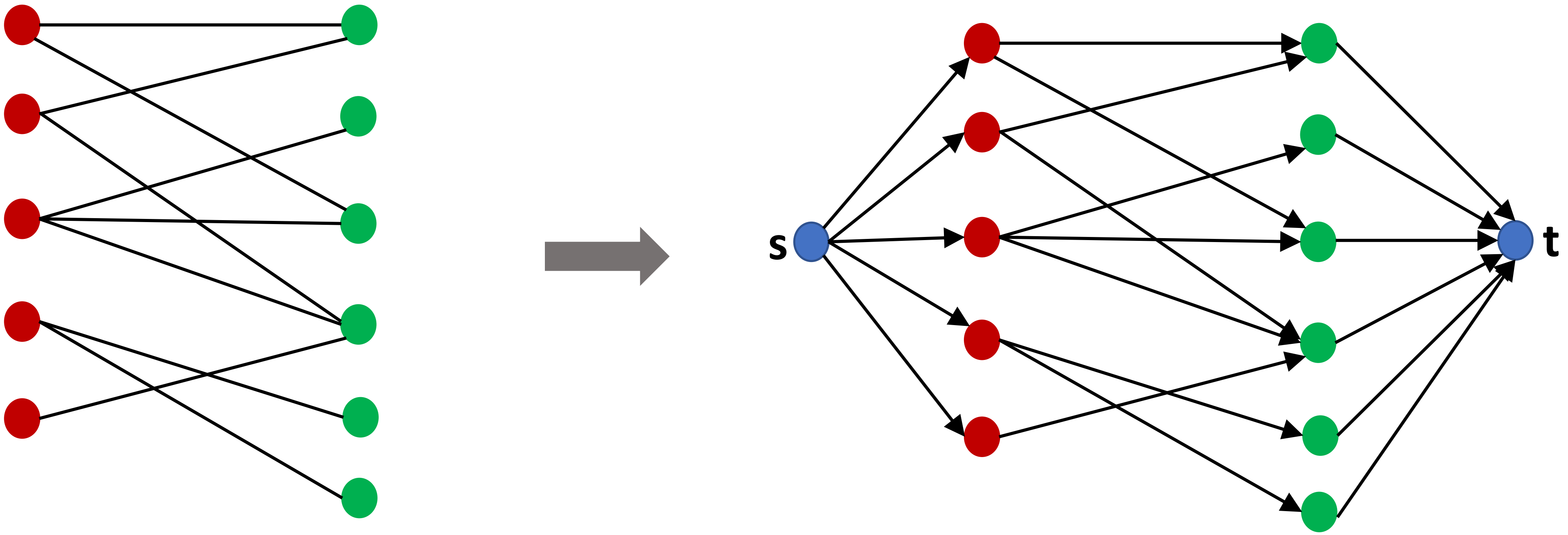
$$\Longrightarrow$$

There exists a maximum flow where flow values are 0 or 1.
Pick the edges with flow 1 and use them to define a matching.
The conservation constraints imply that this is a valid matching.

Reducing bipartite matching to network flow

bipartite graph

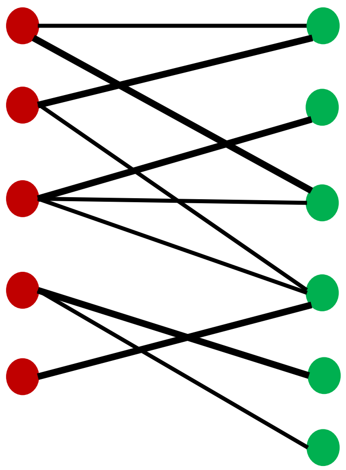
flow network

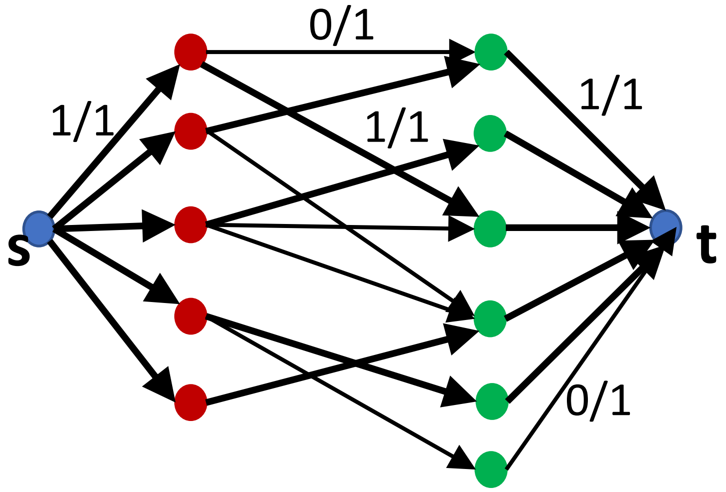


all edge capacity = 1

maximum matching size = maximum flow value

Reducing bipartite matching to network flow



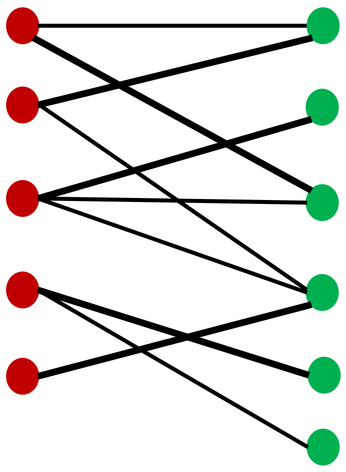


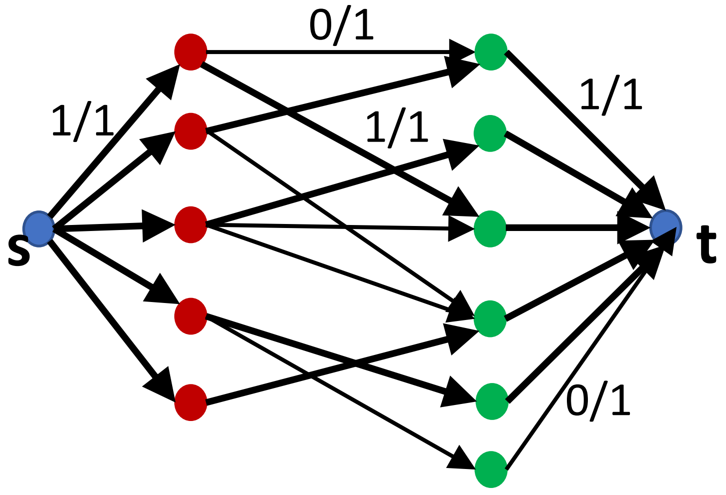
all edge capacity = 1

maximum matching size $\leq$ maximum flow value

Let v be the value of the maximum flow.
Assume there is matching of size $x > v$.
Then, there is a flow with value x.
Since $x > v$ this contradicts v being maximum.
Hence, all matchings have size $\leq v$.

Reducing bipartite matching to network flow





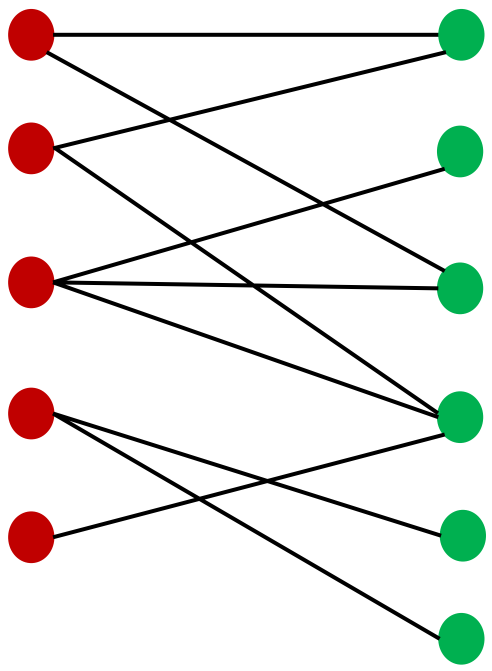
all edge capacity = 1

maximum flow value $\leq$ maximum matching size

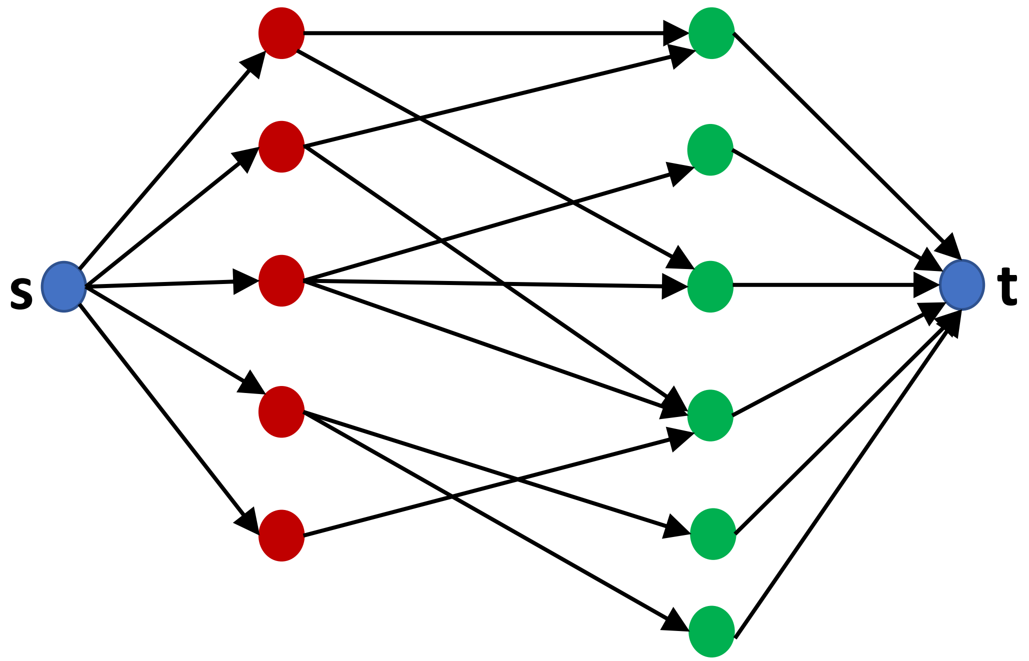
Let v be the size of the maximum matching.
Assume there is flow with value $x > v$.
Then, there is a matching with size x.
Since $x > v$ this contradicts v being maximum.
Hence, all flows have value $\leq v$.

Reducing bipartite matching to network flow

bipartite graph



flow network

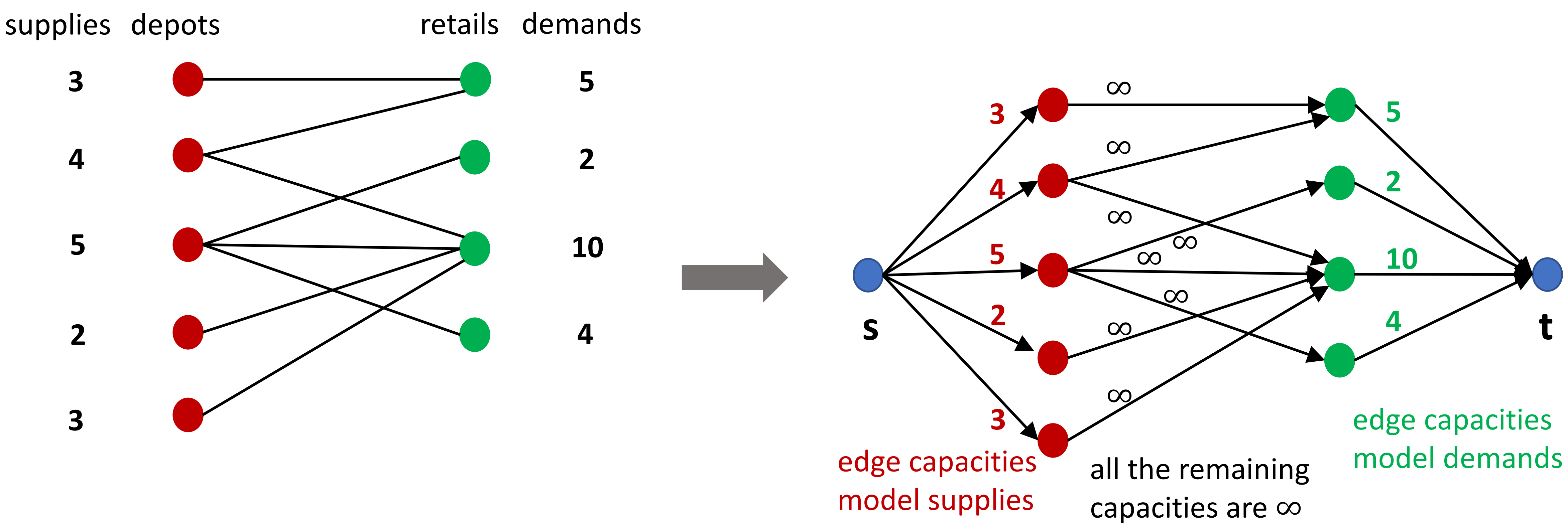


all edge capacity = 1

Using the Ford-Fulkerson algorithm to compute maximum matching:

- all flows assigned to edges by the algorithm are either 0 or 1.

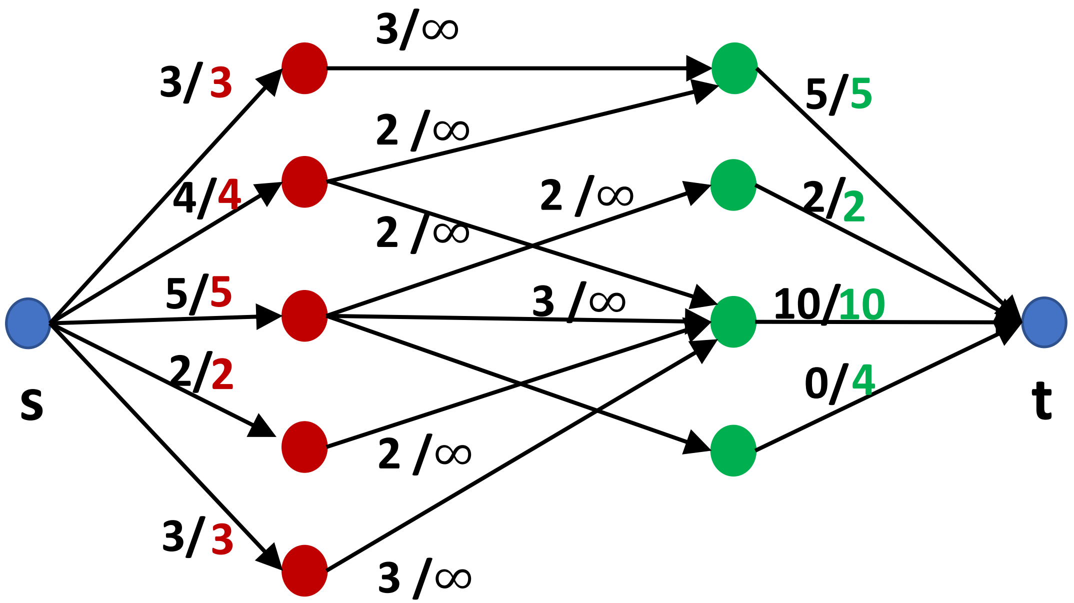
Extension: using different edge capacities



Goal: match supplies to demands such that the maximum amount is matched.

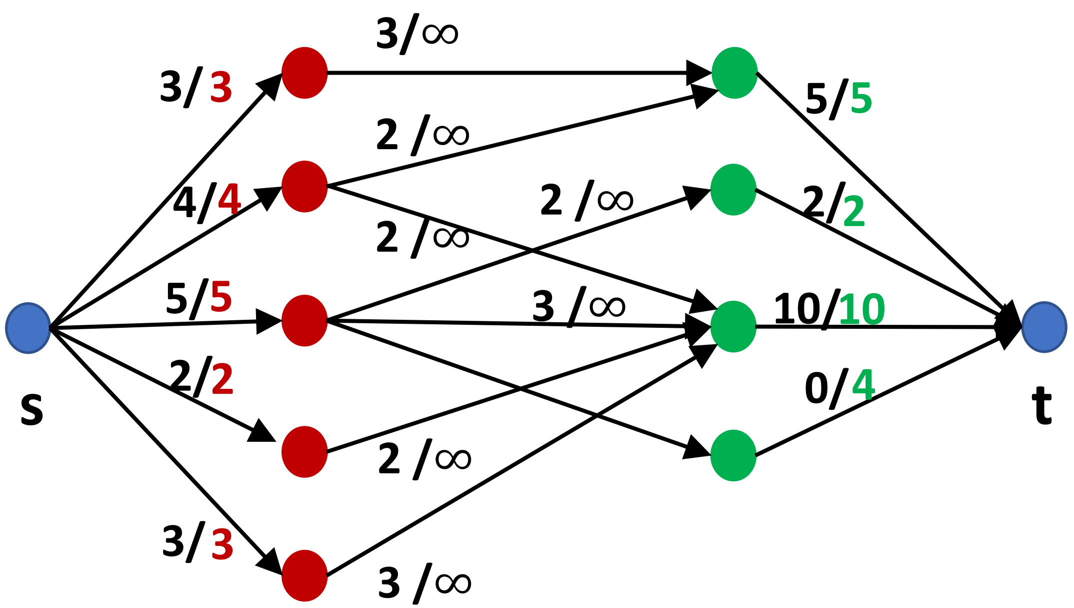
Note: the incoming flow may be split to multiple outgoing edges

Extension: using different edge capacities



Can all the demands be satisfied?

Extension: using different edge capacities



Is this flow maximal?