CS2010: Data Structures and Algorithms II

Maximum Flow

Ivana.Dusparic@scss.tcd.ie

Outline

› Mincut

› Maxflow

› Ford-Fulkerson

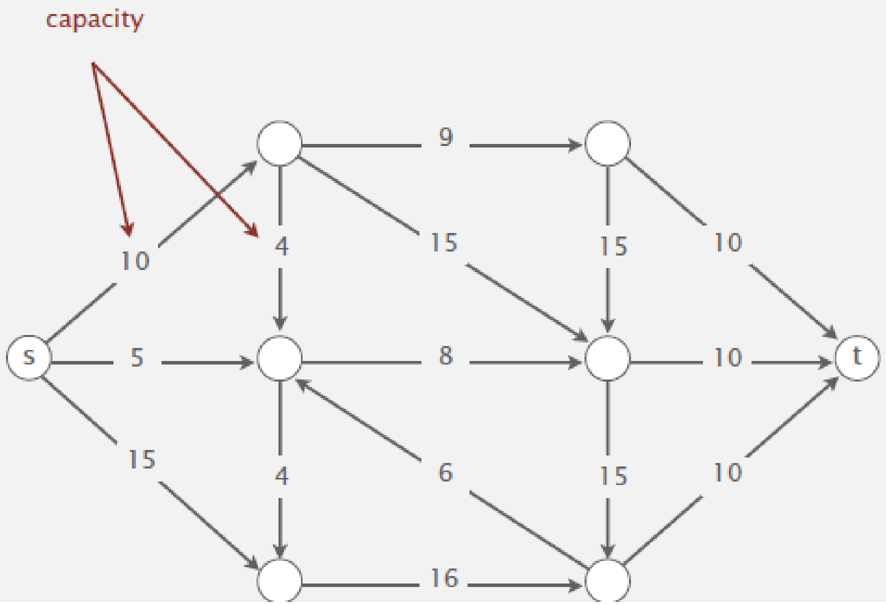
› Edmonds-Karp

› Applications

Mincut Problem

Input. An edge-weighted digraph, source vertex s, and target vertex t.

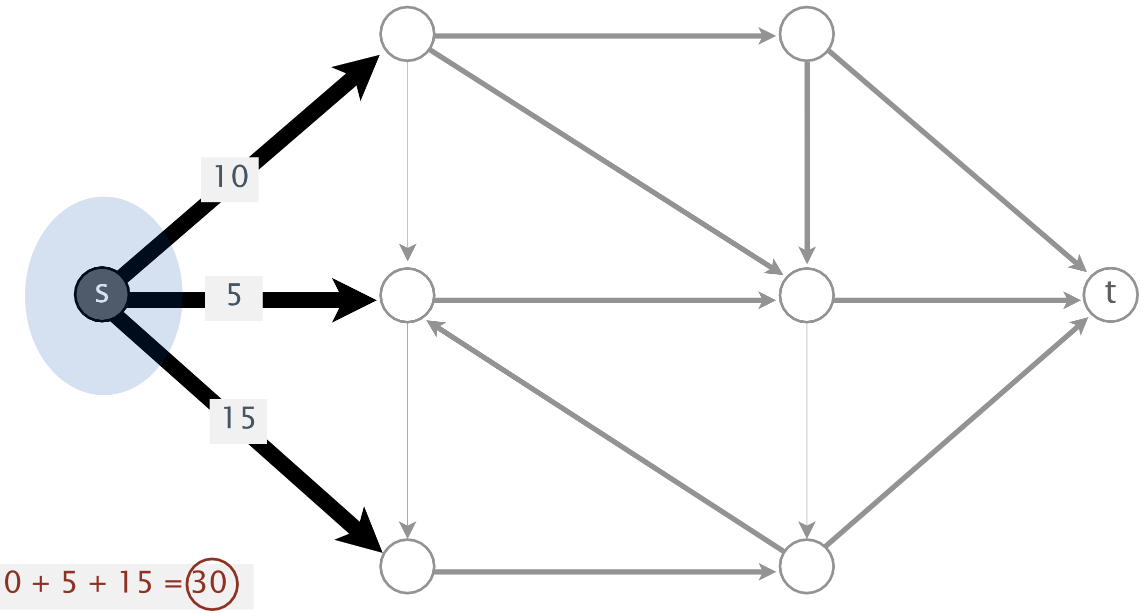
each edge has a positive capacity



Mincut problem

Def. A *st*-cut (cut) is a partition of the vertices into two disjoint sets, with s in one set A and t in the other set B.

Def. Its capacity is the sum of the capacities of the edges from A to B.

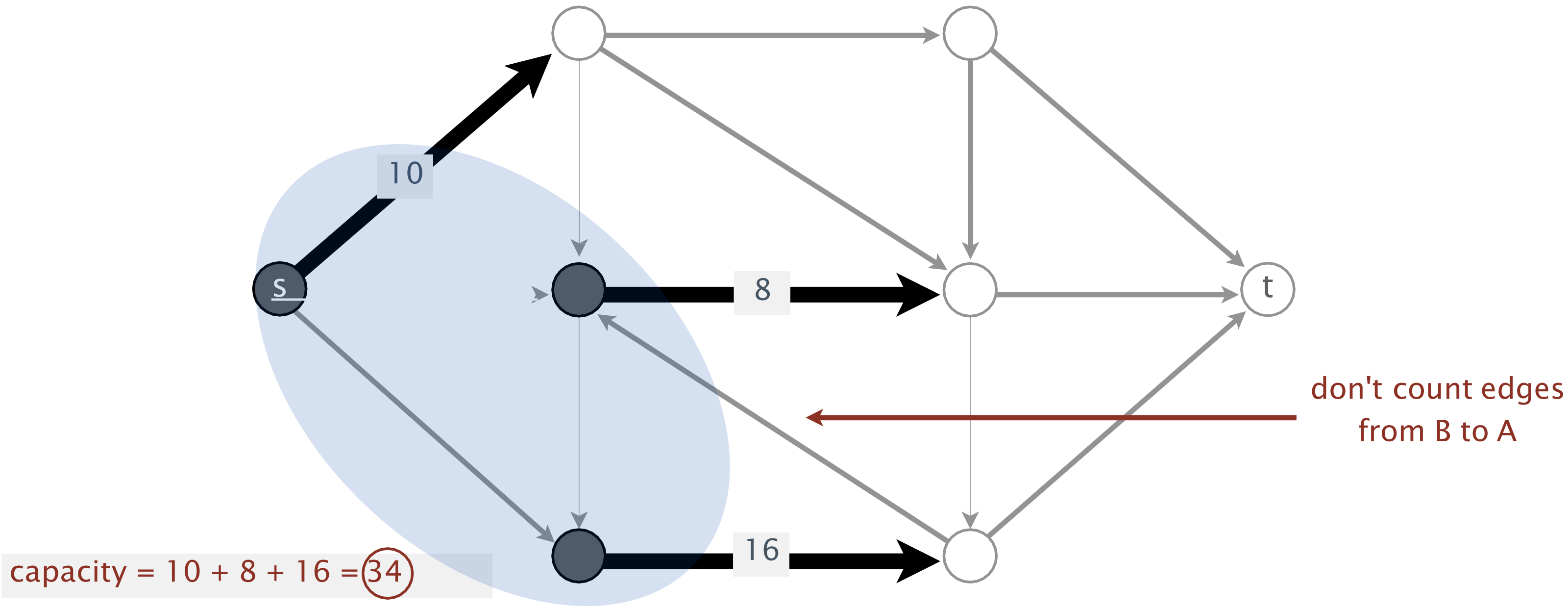


capacity = 10 + 5 + 15 = 30

Mincut problem

Def. A *st*-**cut (cut)** is a partition of the vertices into two disjoint sets, with s in one set A and t in the other set B.

Def. Its **capacity** is the sum of the capacities of the edges from A to B.

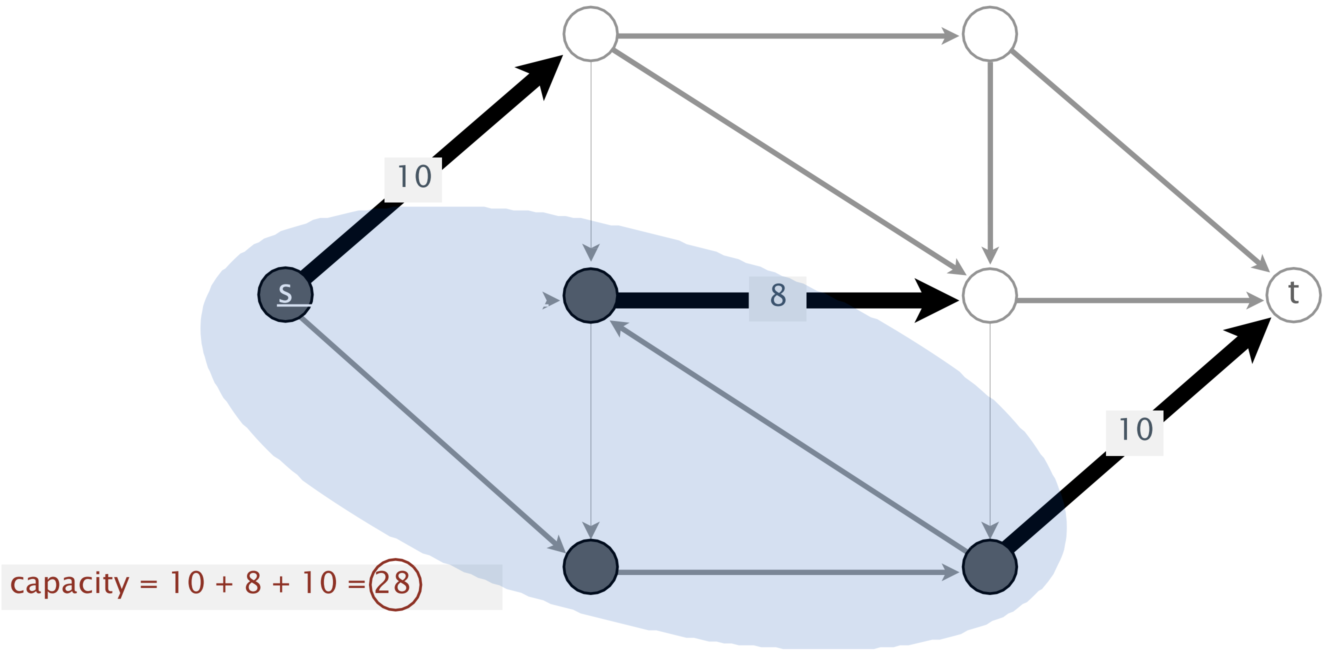


Mincut problem

Def. A *st*-cut (cut) is a partition of the vertices into two disjoint sets, with s in one set A and t in the other set B.

Def. Its capacity is the sum of the capacities of the edges from A to B.

Minimum st-cut (mincut) problem. Find a cut of minimum capacity.

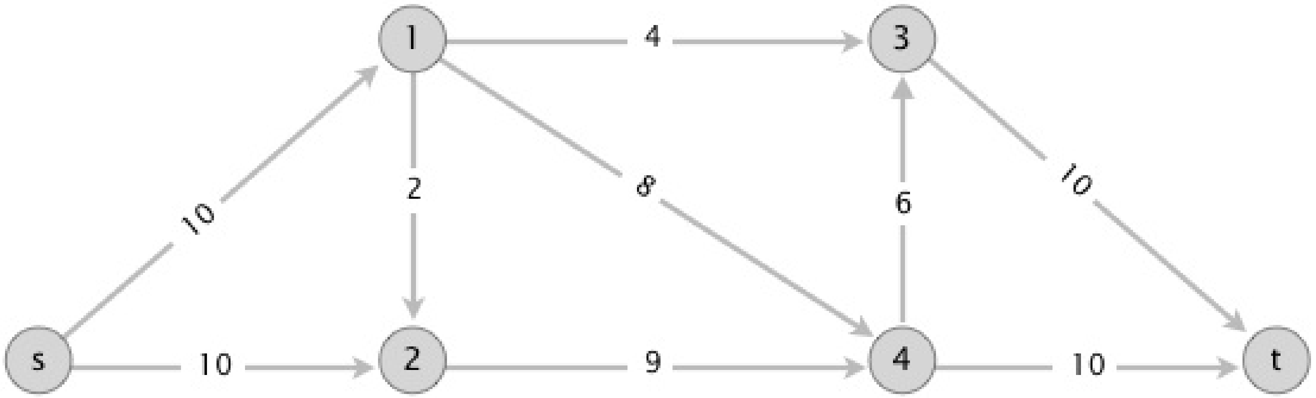


Mincut applications

- Image segmentation – background and foreground
- Cutting of road/rail/information network
- Splitting large-graphs/sharding – if too large to compute
- Community detection in social media – cut along least interactions, or common interests etc
- Etc

Mincut exercise

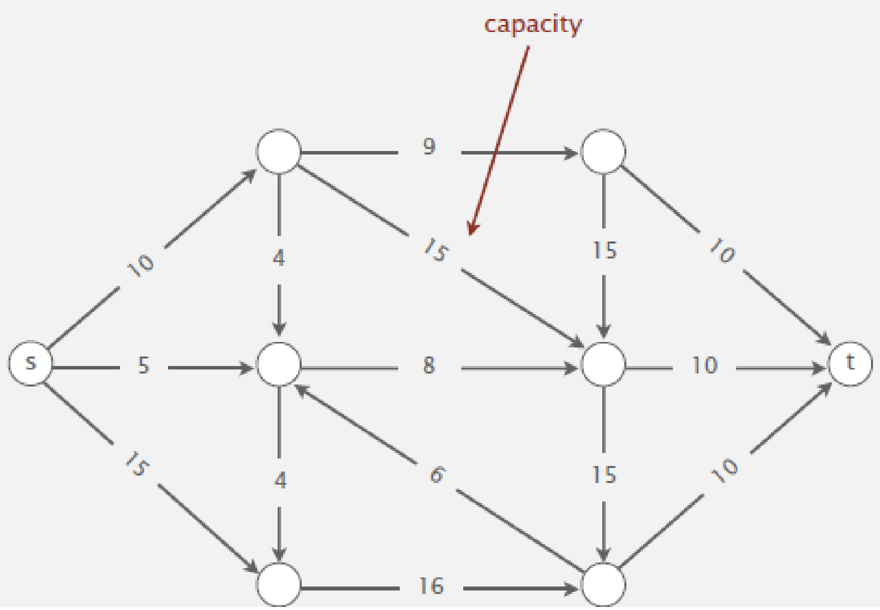
In the flow network below, what is the capacity of the cut with $A = \{s, 2, 4\}$ and $B = \{1, 3, t\}$?



Maxflow problem

Input. An edge-weighted digraph, source vertex s, and target vertex t.

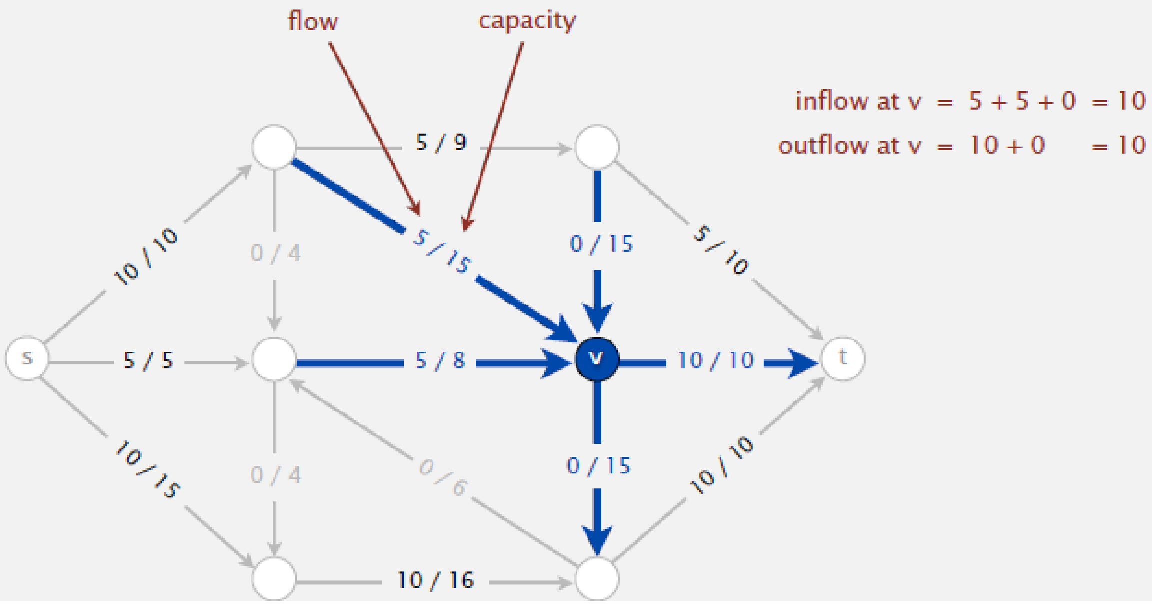
each edge has a positive capacity



Maxflow

Def. An *st*-flow (flow) is an assignment of values to the edges such that:

- Capacity constraint: $0 \leq$ edge's flow $\leq$ edge's capacity.
- Local equilibrium: inflow = outflow at every vertex (except s and t).



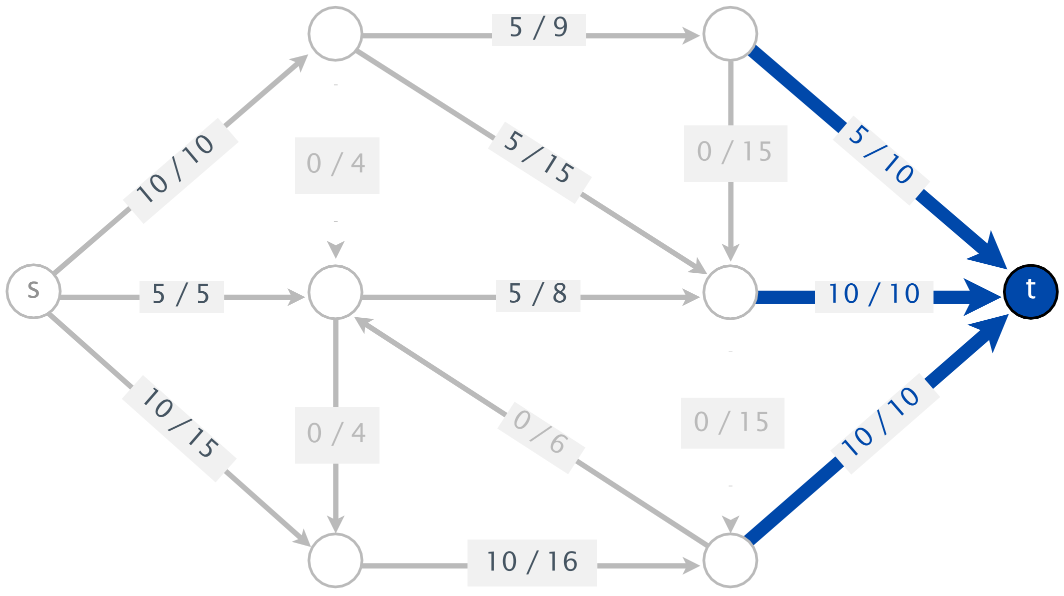
Maxflow problem

Def. An *st*-flow (flow) is an assignment of values to the edges such that:

- Capacity constraint: $0 \le$ edge's flow $\le$ edge's capacity.
- Local equilibrium: inflow = outflow at every vertex (except s and t).

Def. The value of a flow is the inflow at t.

we assume no edges point to s or from t



value = 5 + 10 + 10 = 25

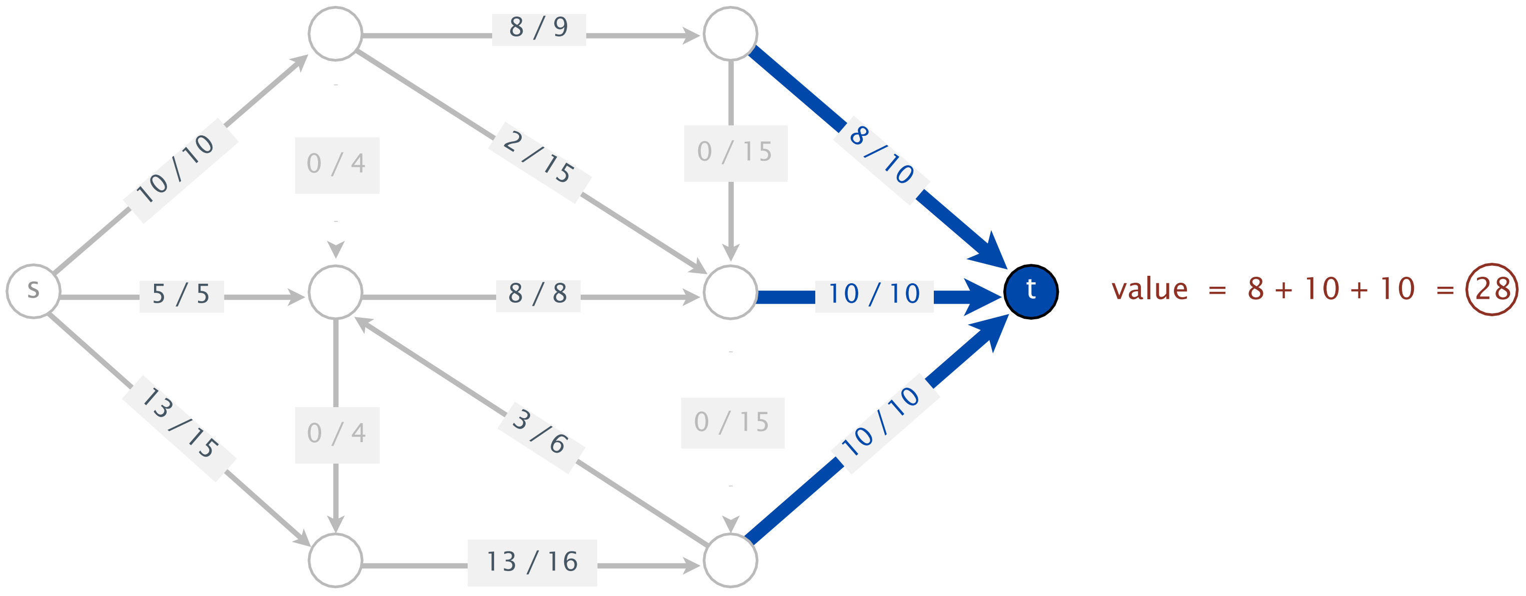
Maxflow problem

Def. An *st*-**flow (flow)** is an assignment of values to the edges such that:

- Capacity constraint: $0 \le$ edge's flow $\le$ edge's capacity.
- Local equilibrium: inflow = outflow at every vertex (except s and t).

Def. The **value** of a flow is the inflow at t.

Maximum st-flow (maxflow) problem. Find a flow of maximum value.

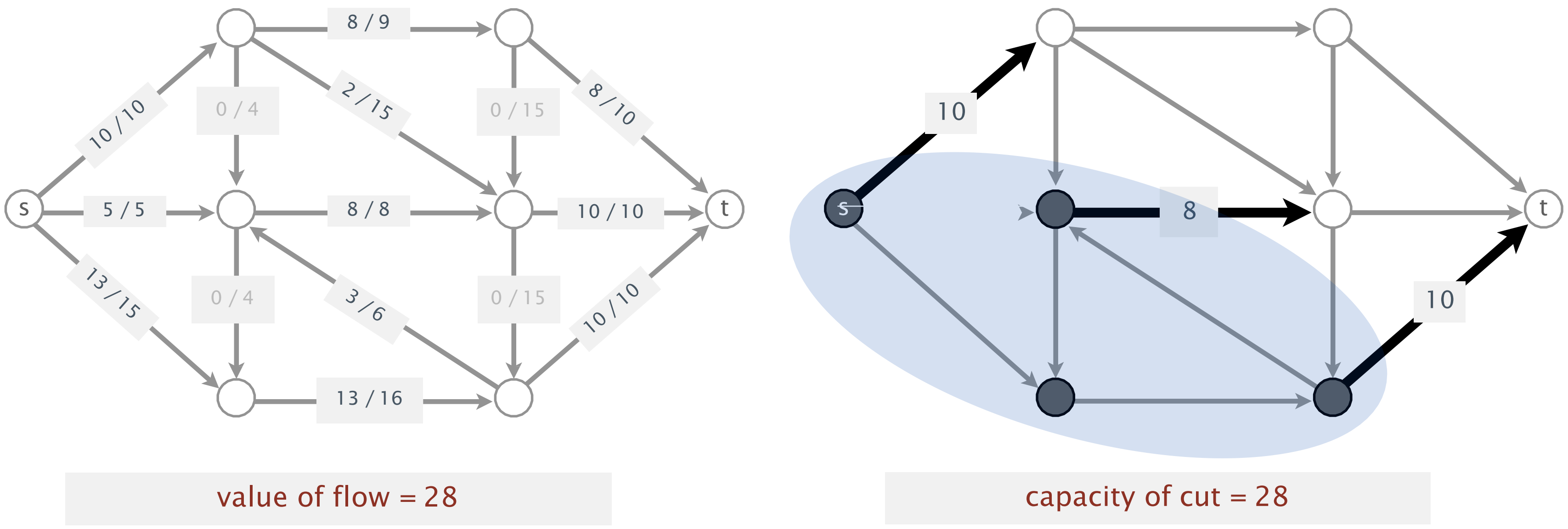


Summary

Input. A weighted digraph, source vertex s, and target vertex t.

Mincut problem. Find a cut of minimum capacity.

Maxflow problem. Find a flow of maximum value.



Remarkable fact. These two problems are dual!

Ford-Fulkerson method

Residual graph

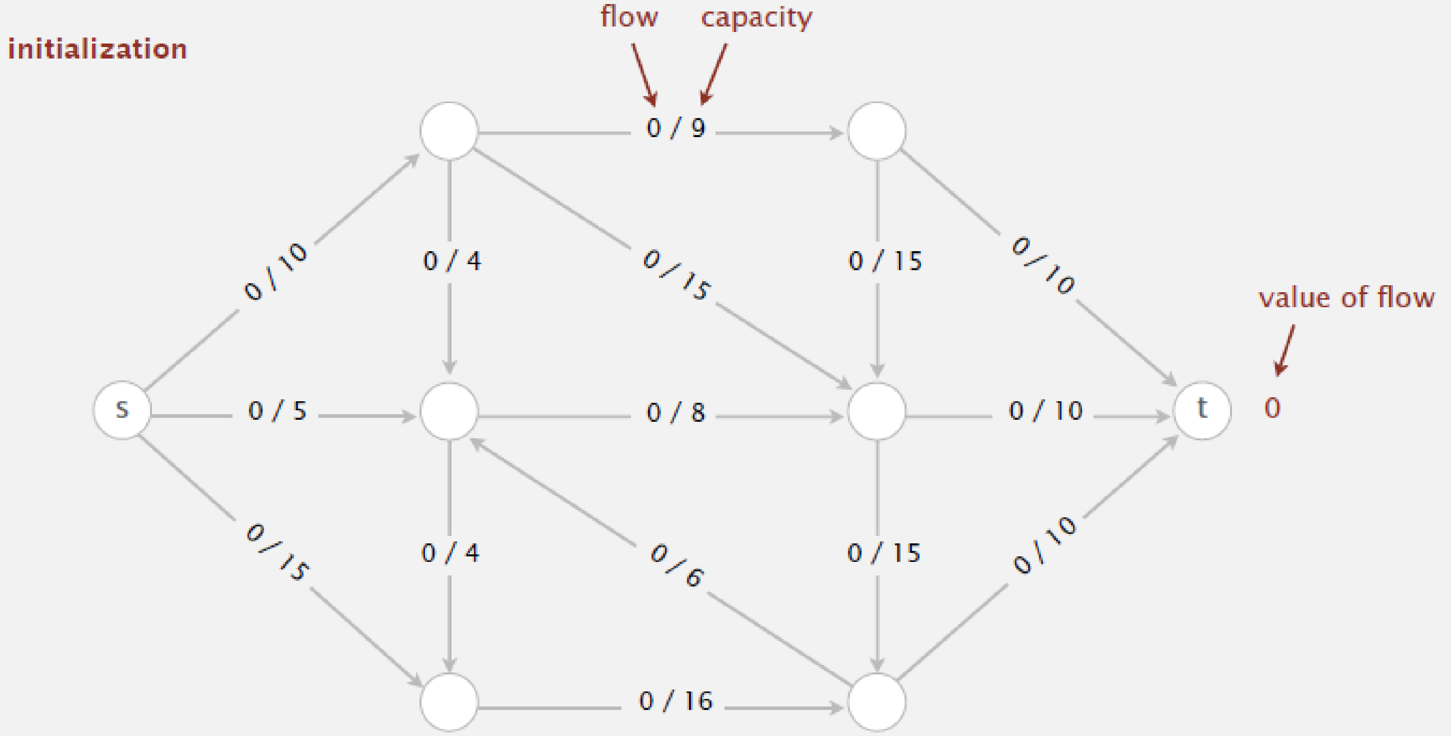
- Residual capacity of an edge – difference between capacity and flow
- Residual graph:
  - same vertices as the original network
  - one or two edges for each edge in the original
  - if the flow along the edge x-y is less than the capacity there is a forward edge x-y with a capacity equal to the difference between the capacity and the flow (this is called the residual capacity)
  - if the flow is positive there is a backward edge y-x with a capacity equal to the flow on x-y.

Augmenting path

- a path from the source to the sink in the residual network, whose purpose is to increase the flow in the original one
- The edges in this path can point the “wrong way” according to the original network.
- The path capacity of a path is the minimum capacity of an edge along that path.

Ford-Fulkerson

Initialization. Start with 0 flow.

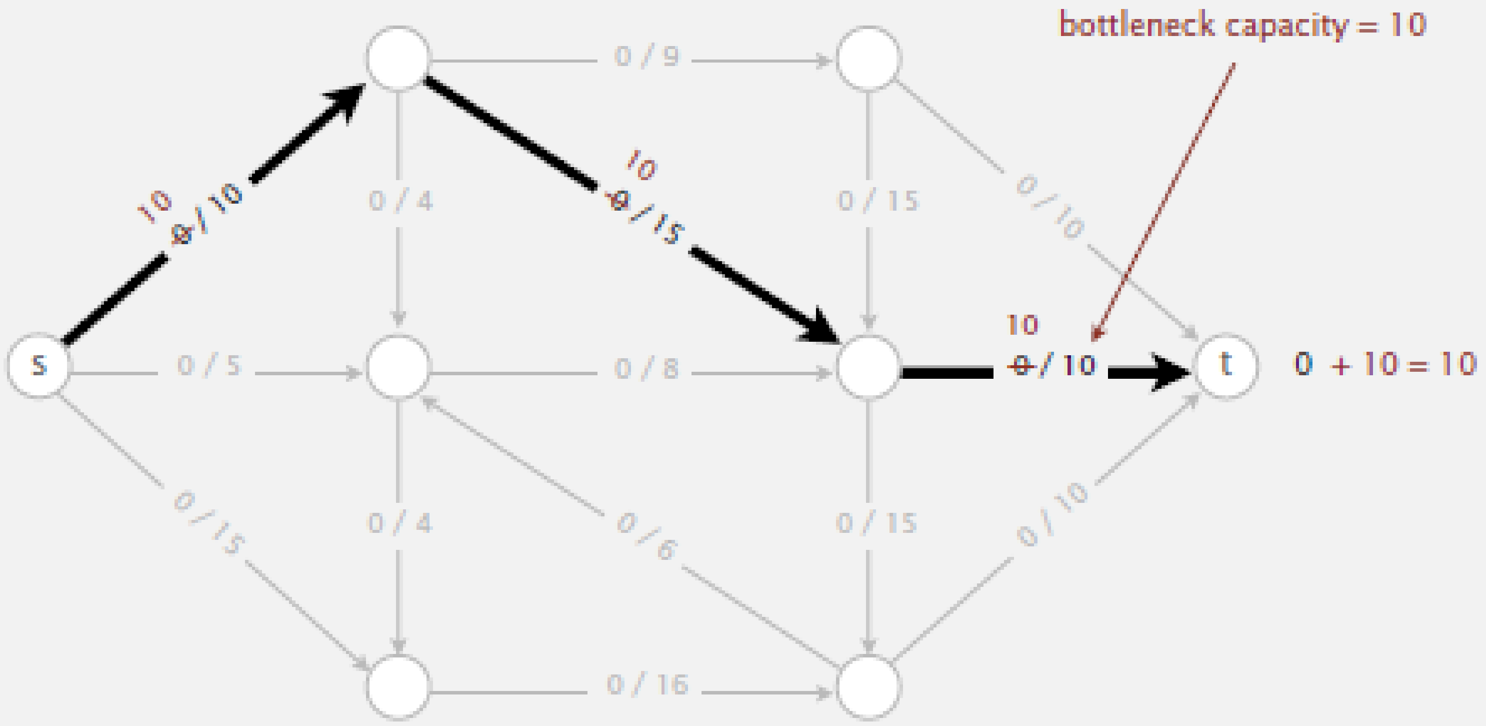


Idea: increase flow along augmenting paths

Augmenting path. Find an undirected path from s to t such that:

- Can increase flow on forward edges (not full).
- Can decrease flow on backward edge (not empty).

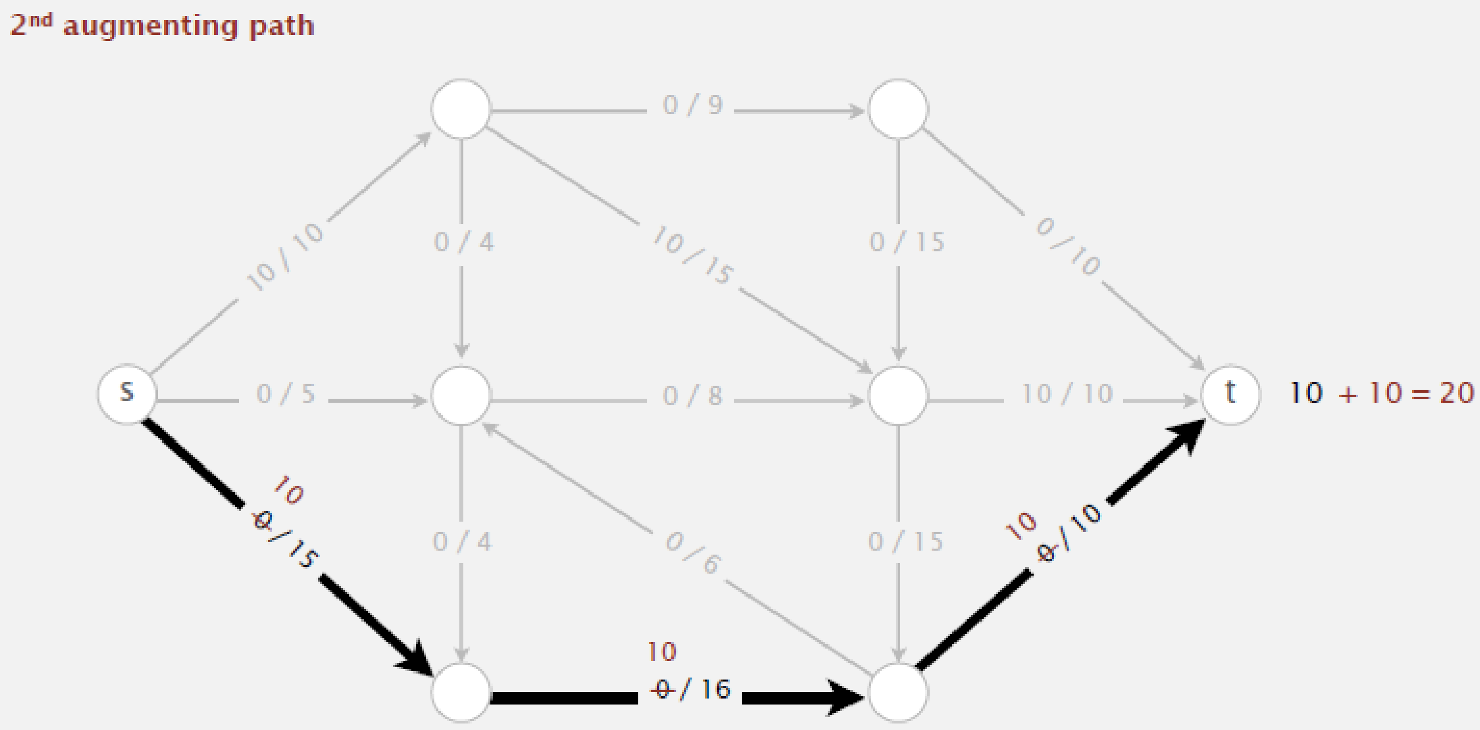
1st augmenting path



Idea: increase flow along augmenting paths

Augmenting path. Find an undirected path from s to t such that:

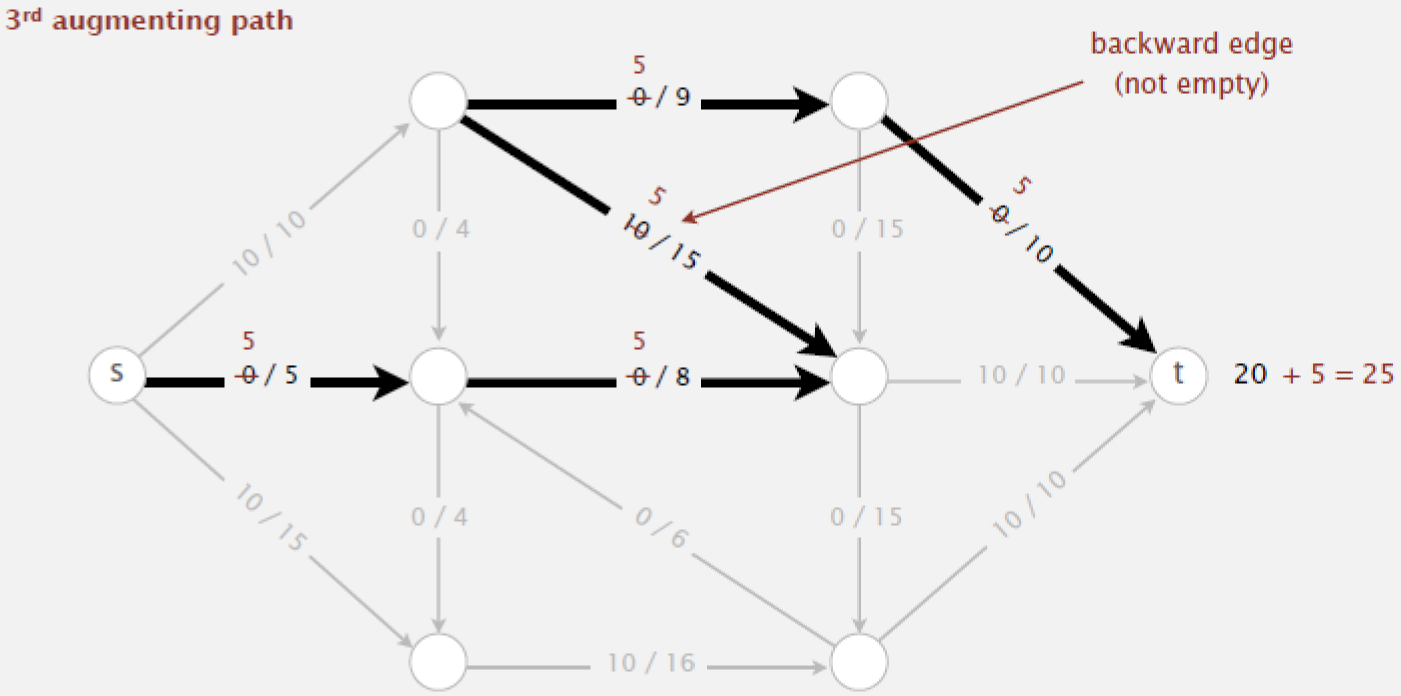
- Can increase flow on forward edges (not full).
- Can decrease flow on backward edge (not empty).



Idea: increase flow along augmenting paths

Augmenting path. Find an undirected path from s to t such that:

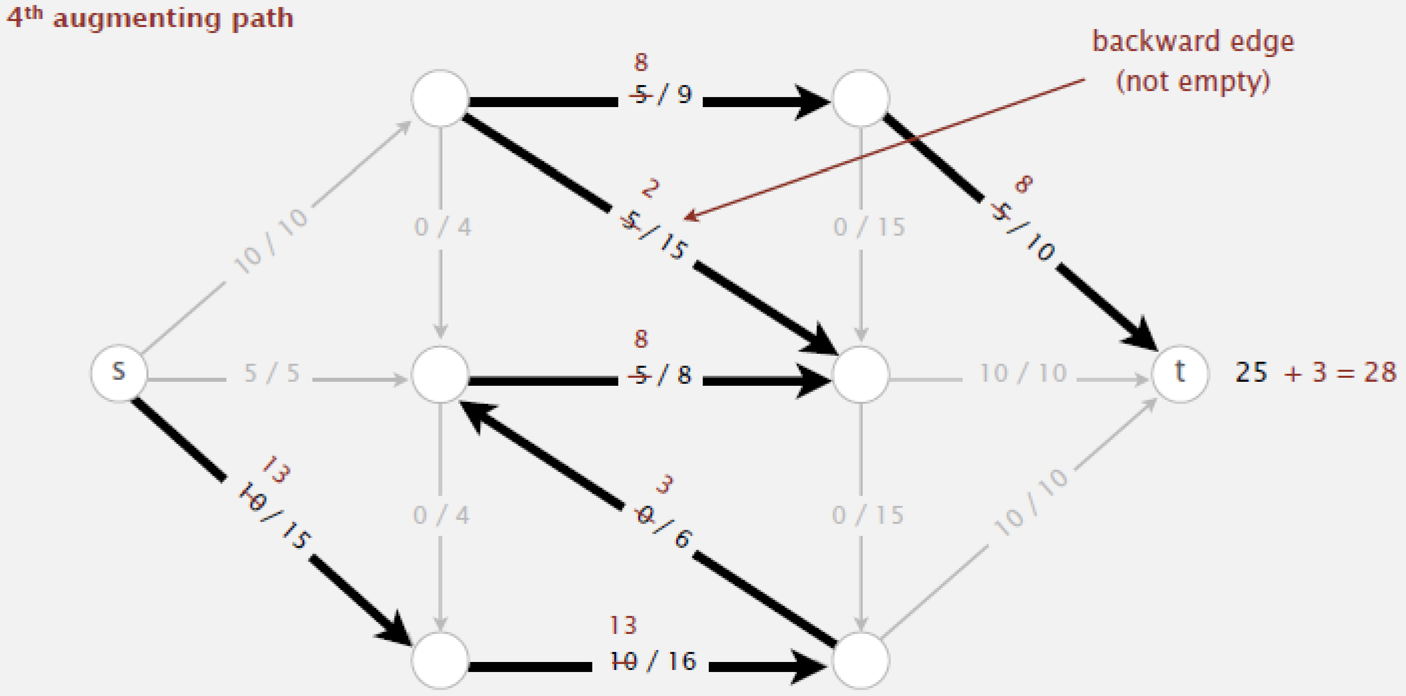
- Can increase flow on forward edges (not full).
- Can decrease flow on backward edge (not empty).



Idea: increase flow along augmenting paths

Augmenting path. Find an undirected path from s to t such that:

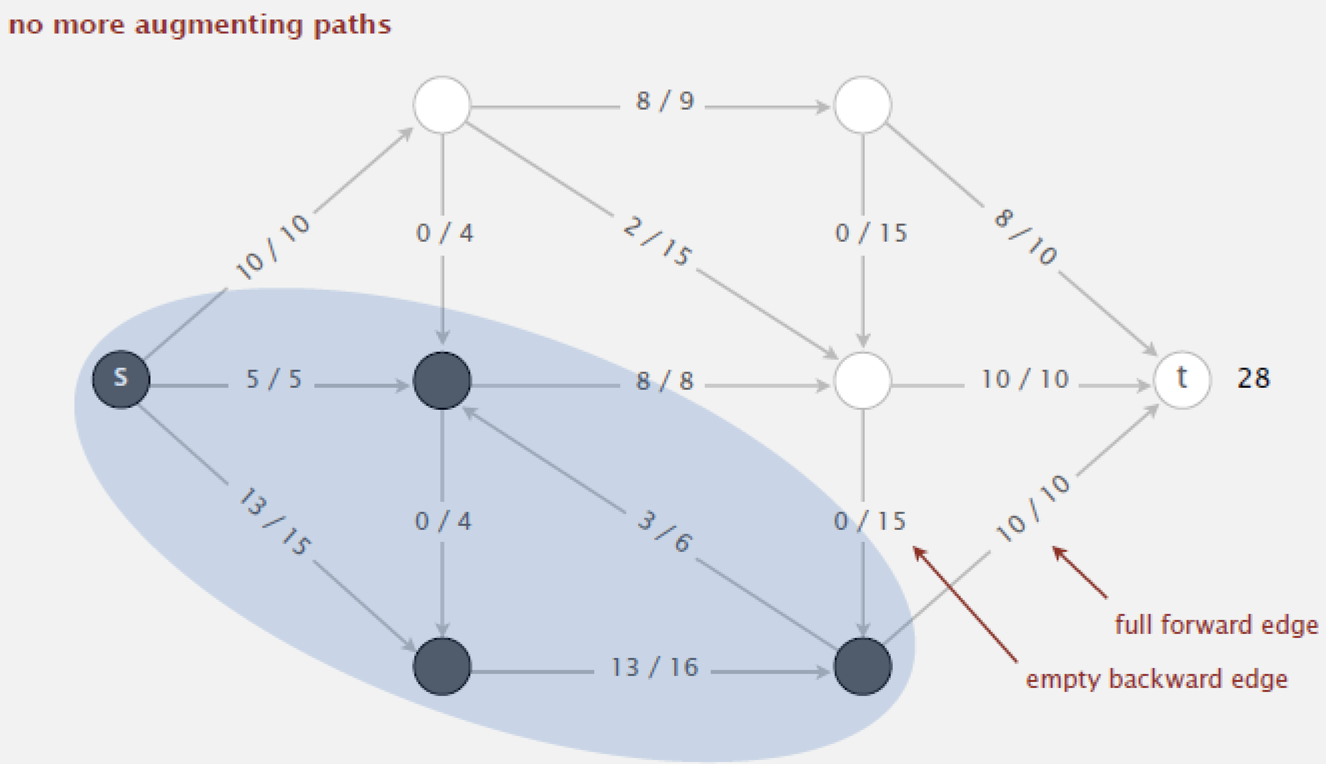
- Can increase flow on forward edges (not full).
- Can decrease flow on backward edge (not empty).



Idea: increase flow along augmenting paths

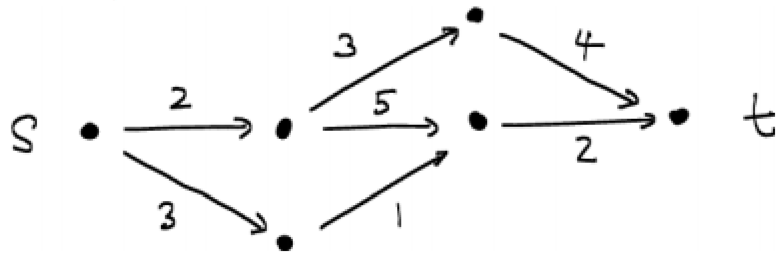
Termination. All paths from s to t are blocked by either a

- Full forward edge.
- Empty backward edge.

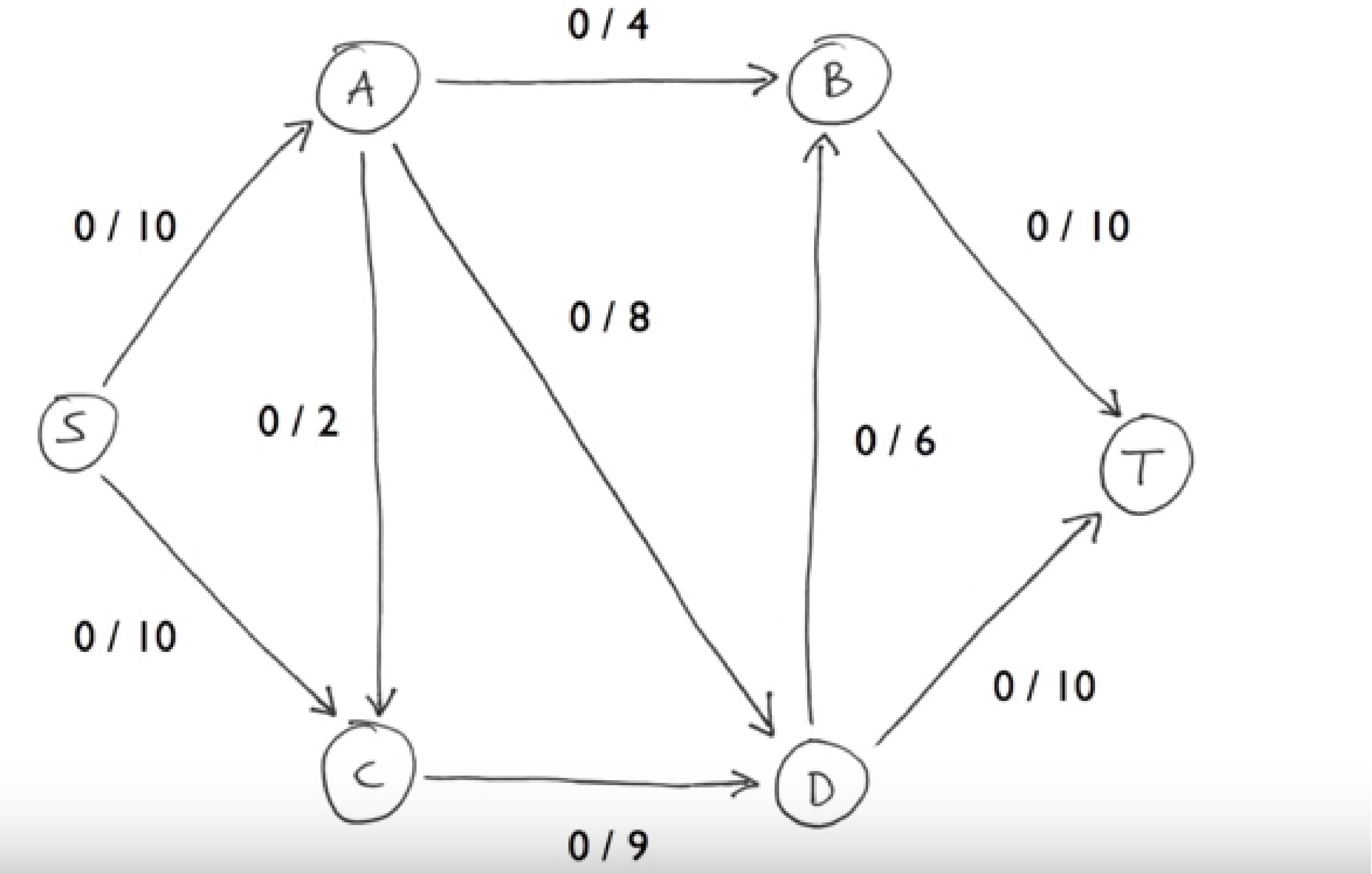


example

Consider a flow network G,



Ford Fulkerson exercise



Ford Fulkerson exercise solution

https://www.youtube.com/watch?v=Tl90tNtKvxs

Ford-Fulkerson algorithm

Ford-Fulkerson algorithm

Start with 0 flow.

While there exists an augmenting path:

- **find an augmenting path**
- **compute bottleneck capacity**
- **increase flow on that path by bottleneck capacity**

Questions.

- How to compute a mincut?
- How to find an augmenting path?
- If FF terminates, does it always compute a maxflow?
- Does FF always terminate? If so, after how many augmentations?

How to find mincut from maxflow?

- mincut – cut of minimum capacity
- flow – flow into source
- Maxflow – when no more augmenting paths

Maxflow-mincut theorem

Augmenting path theorem. A flow f is a maxflow iff no augmenting paths.

Maxflow-mincut theorem. Value of the maxflow = capacity of mincut.

Pf. The following three conditions are equivalent for any flow f:

i. There exists a cut whose capacity equals the value of the flow f.

ii. f is a maxflow.

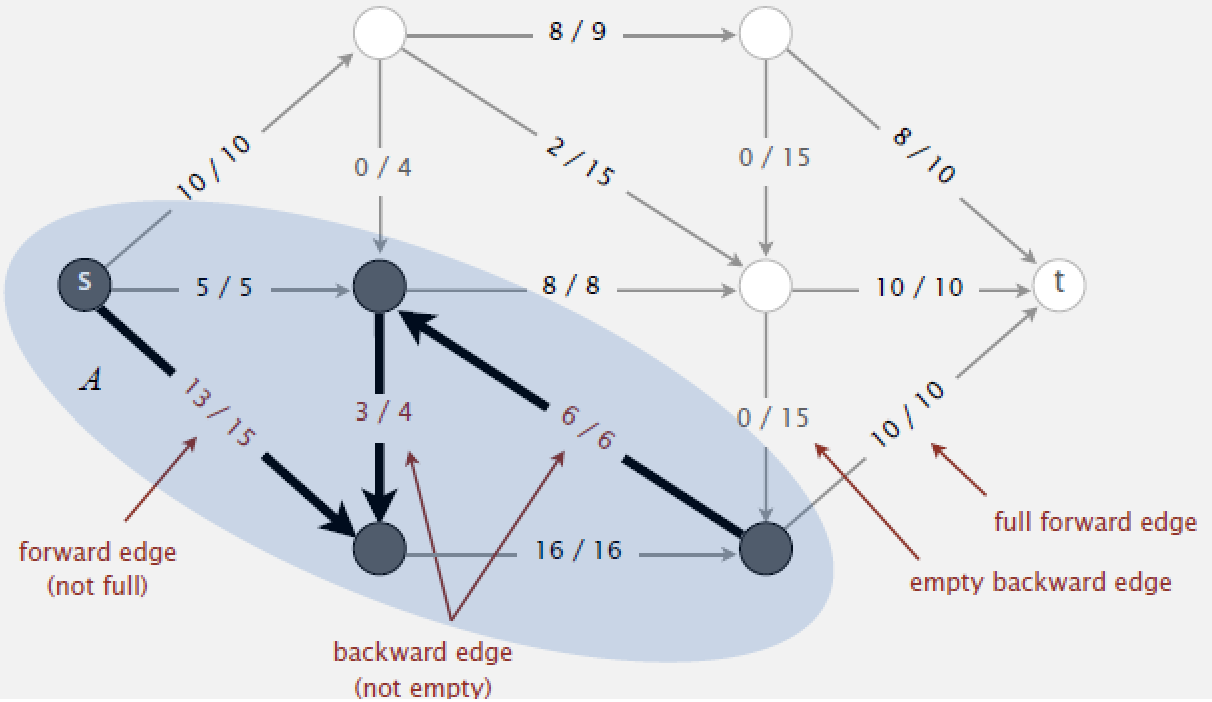
iii. There is no augmenting path with respect to f.

Proof and background in Sedgwick

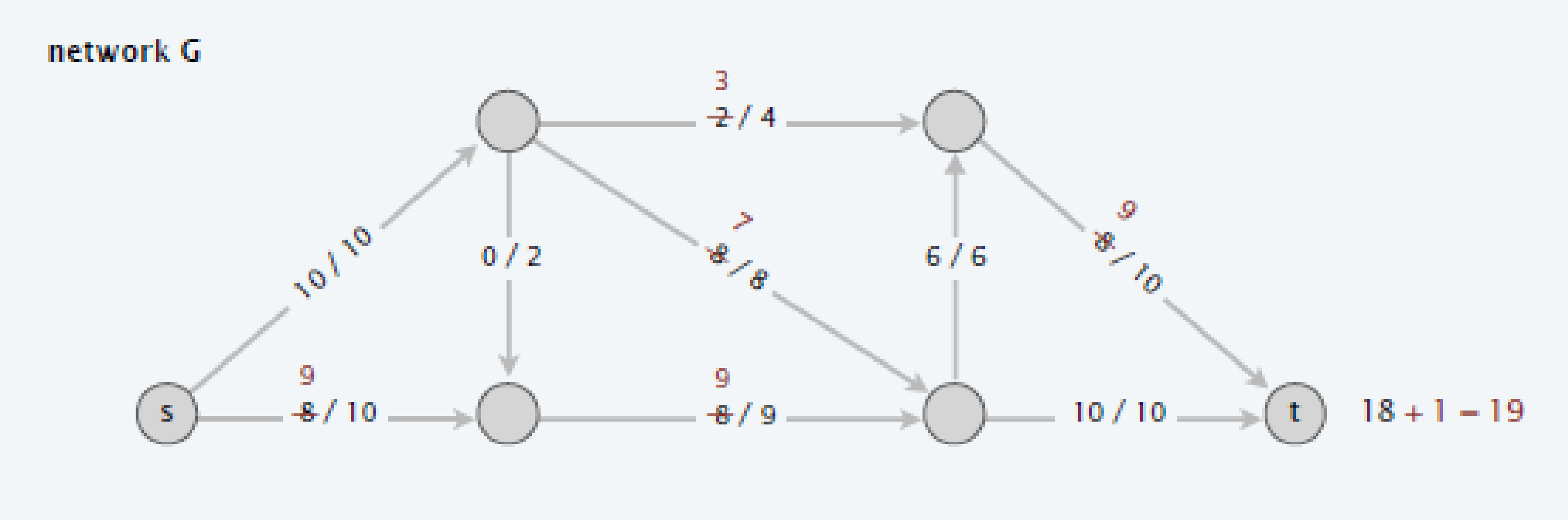
Computing a mincut from a maxflow

To compute mincut (A, B) from maxflow f :

- By augmenting path theorem, no augmenting paths with respect to f.
- Compute A = set of vertices connected to s by an undirected path with no full forward or empty backward edges.



Mincut in exercise



Ford-Fulkerson algorithm

Ford-Fulkerson algorithm

Start with 0 flow.

While there exists an augmenting path:

- **find an augmenting path**
- **compute bottleneck capacity**
- **increase flow on that path by bottleneck capacity**

Questions.

- How to compute a mincut? Easy. ✓
- How to find an augmenting path? BFS works well.
- If FF terminates, does it always compute a maxflow? Yes. ✓
- Does FF always terminate? If so, after how many augmentations?

yes, provided edge capacities are integers (or augmenting paths are chosen carefully)

requires clever analysis

Ford-Fulkerson algorithm with integer capacities

Important special case. Edge capacities are integers between 1 and U.

flow on each edge is an integer

Invariant. The flow is integer-valued throughout Ford-Fulkerson.

Pf. [by induction]

- Bottleneck capacity is an integer.
- Flow on an edge increases/decreases by bottleneck capacity.

Proposition. Number of augmentations $\leq$ the value of the maxflow.

Pf. Each augmentation increases the value by at least 1.

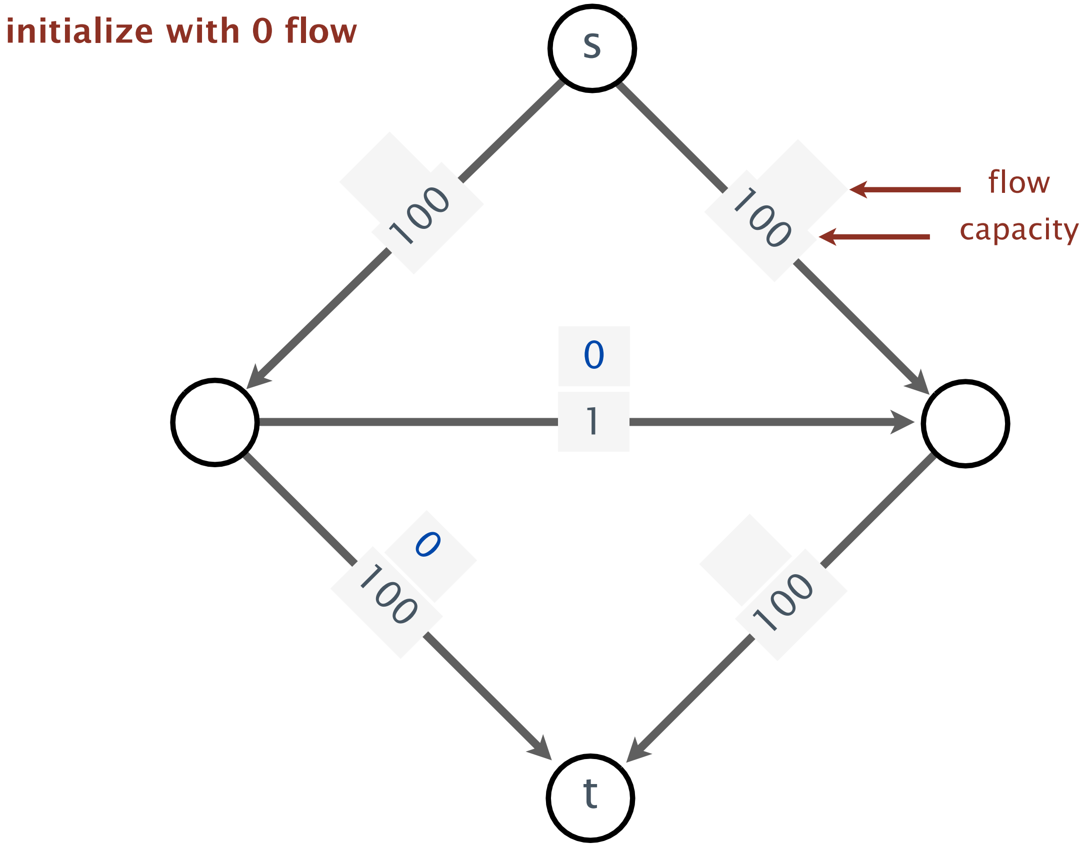
critical for some applications (stay tuned)

Integrality theorem. There exists an integer-valued maxflow.

Pf. Ford-Fulkerson terminates and maxflow that it finds is integer-valued.

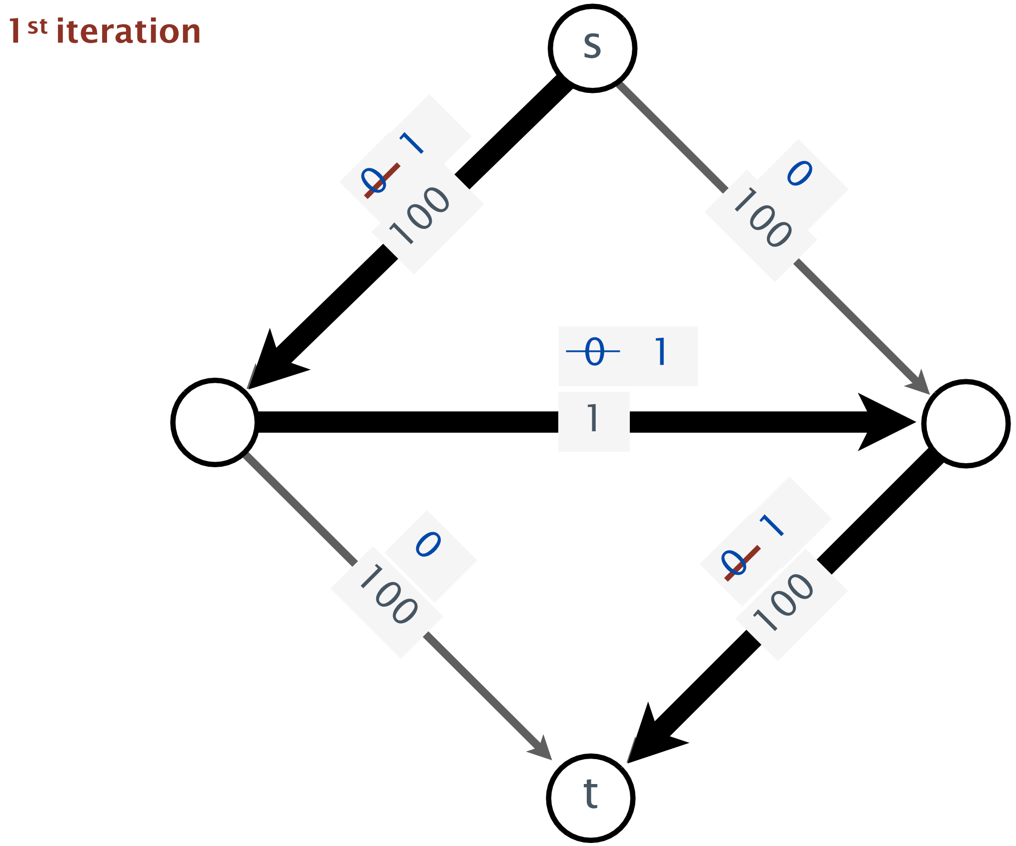
Bad case for Ford-Fulkerson

Bad news. Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maxflow.



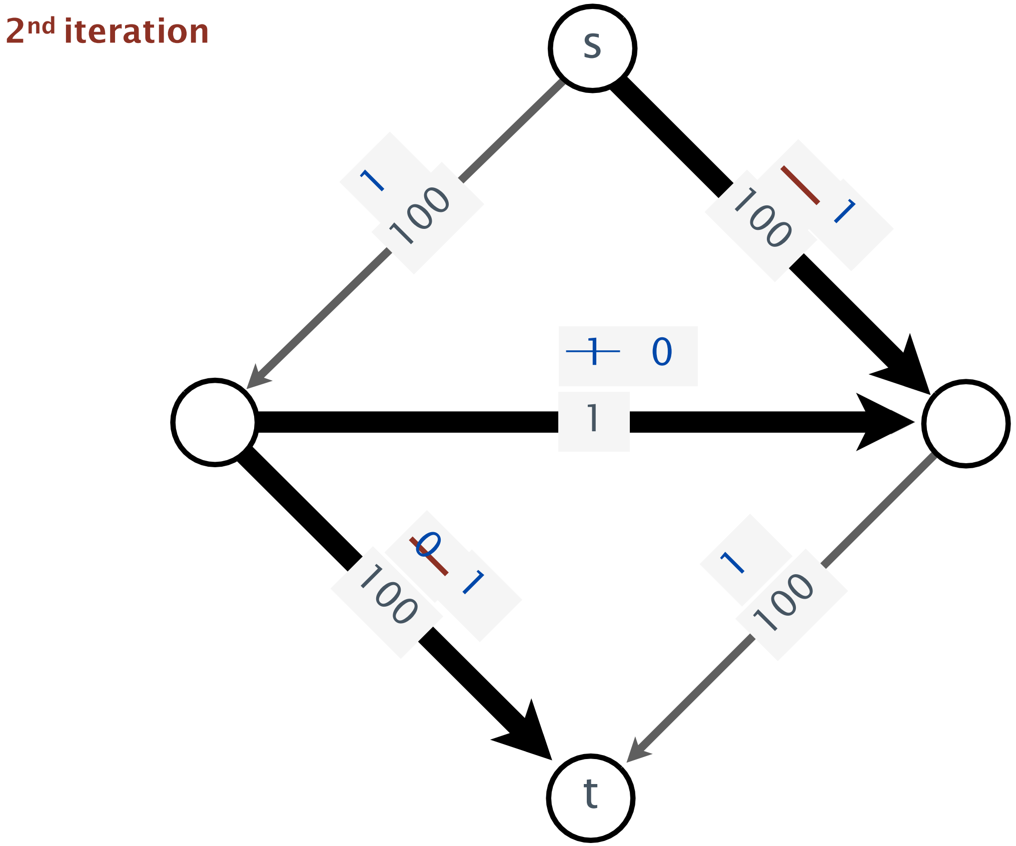
Bad case for Ford-Fulkerson

Bad news. Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maxflow.



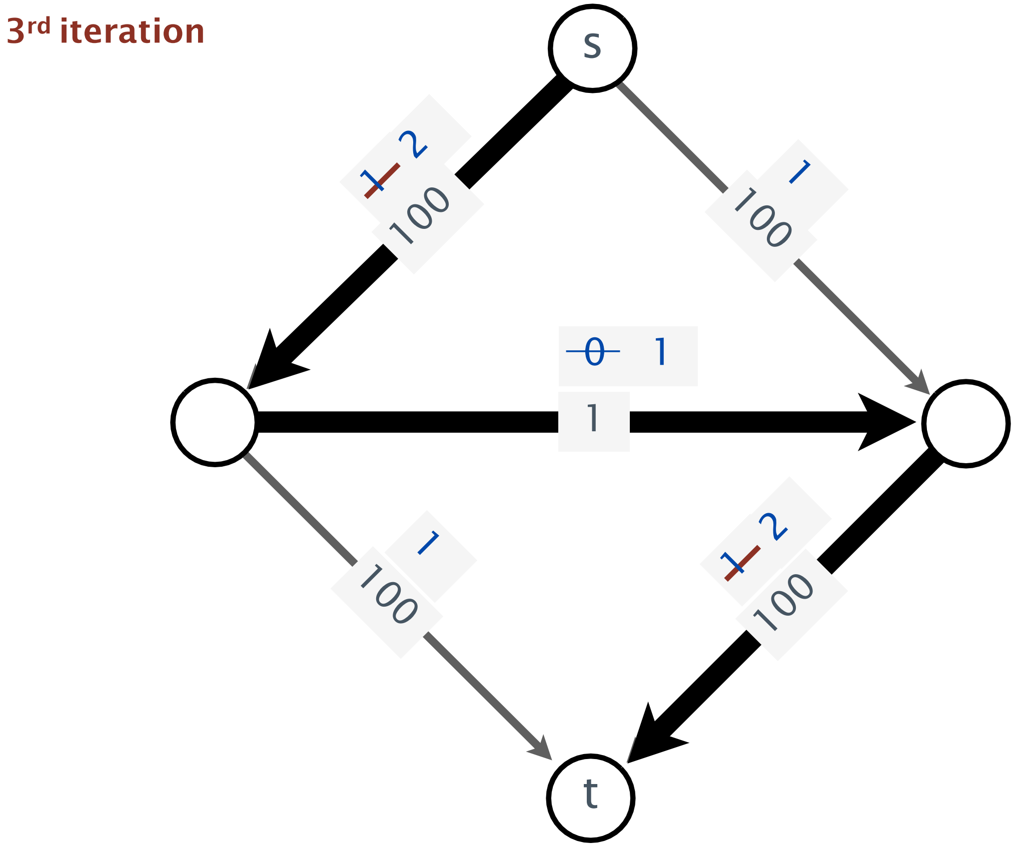
Bad case for Ford-Fulkerson

Bad news. Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maxflow.



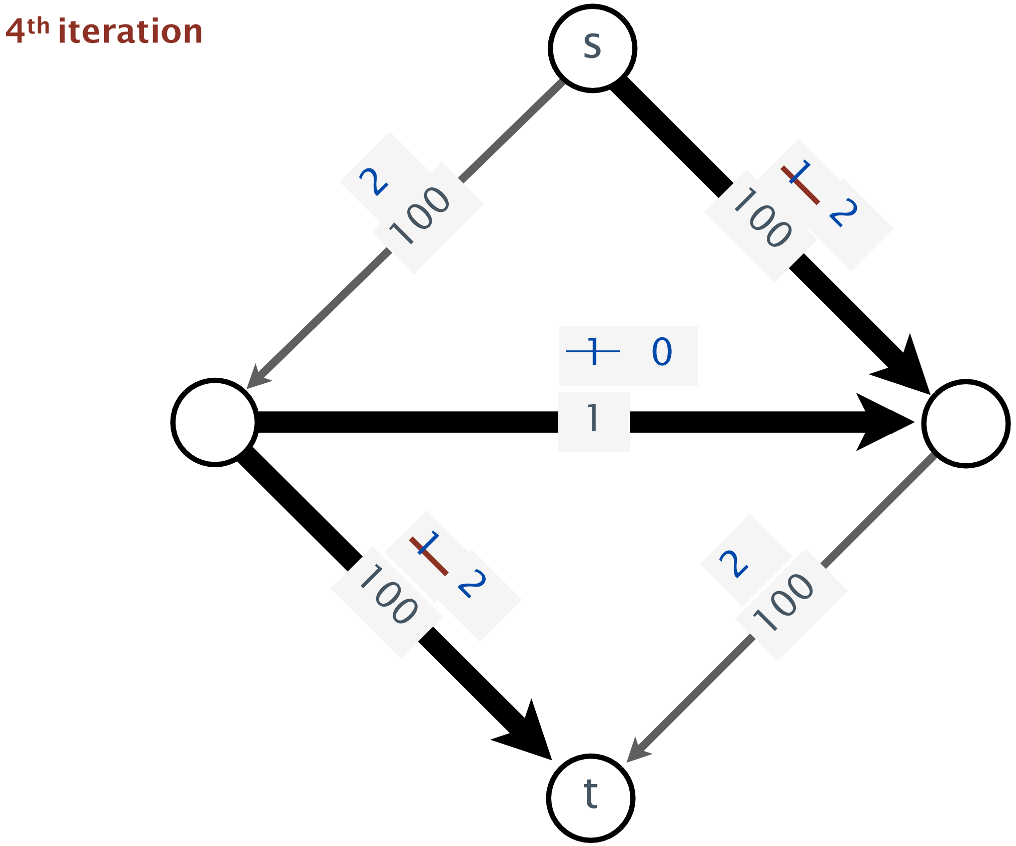
Bad case for Ford-Fulkerson

Bad news. Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maxflow.



Bad case for Ford-Fulkerson

Bad news. Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maxflow.



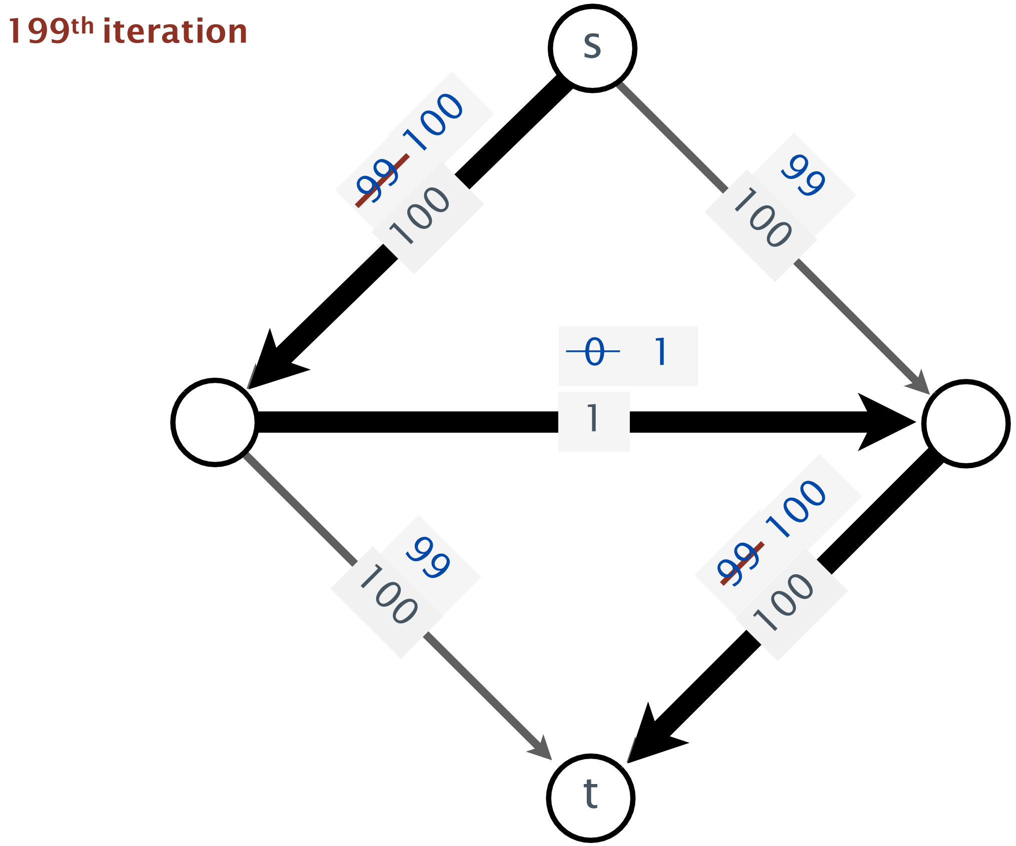
Bad case for Ford-Fulkerson

Bad news. Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maxflow.

• • •

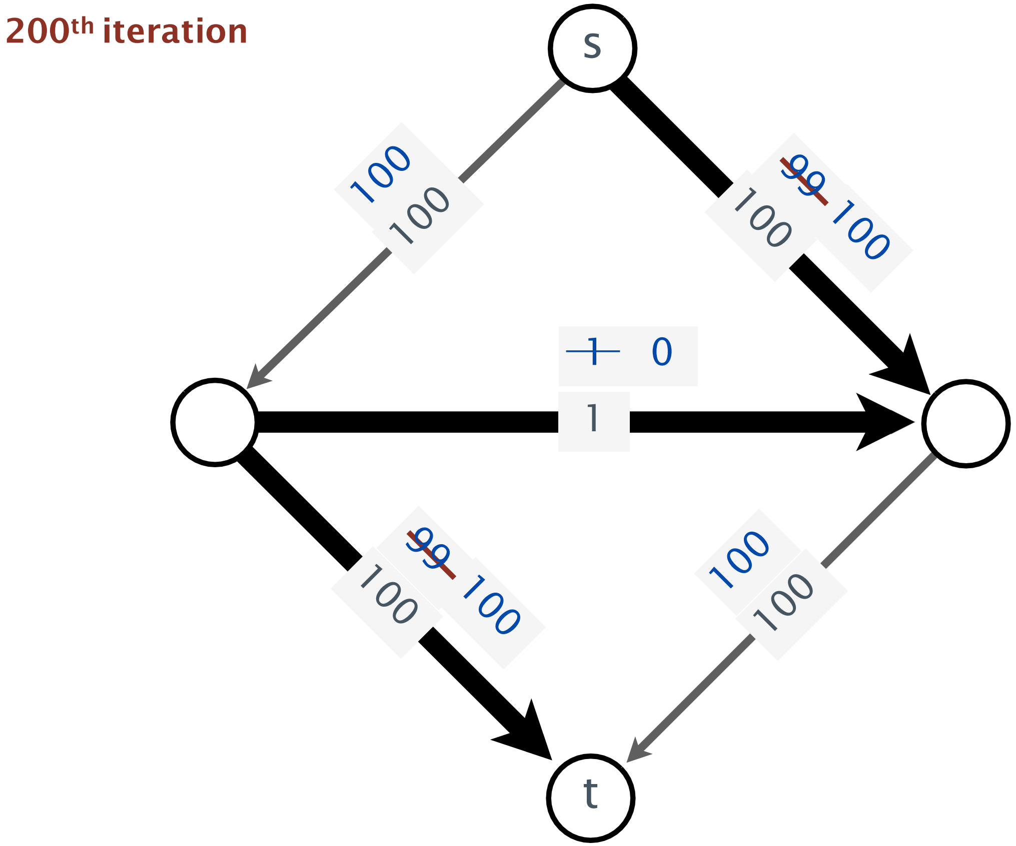
Bad case for Ford-Fulkerson

Bad news. Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maxflow.



Bad case for Ford-Fulkerson

Bad news. Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maxflow.

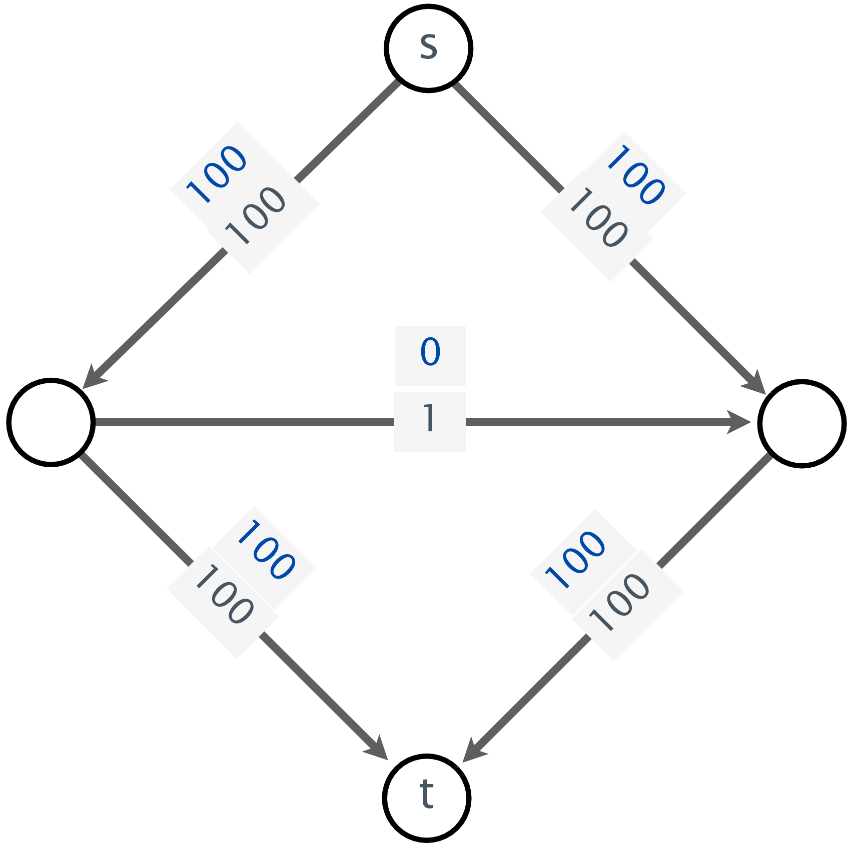


Bad case for Ford-Fulkerson

Bad news. Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maxflow.

can be exponential in input size

Good news. This case is easily avoided. [use shortest/fattest path]



How to choose augmenting paths?

Use care when selecting augmenting paths.

- Some choices lead to exponential algorithms.
- Clever choices lead to polynomial algorithms.

augmenting path	number of paths	implementation
random path	$\leq E\,U$	randomized queue
DFS path	$\leq E\,U$	stack (DFS)
shortest path	$\leq \frac{1}{2}\,E\,V$	queue (BFS)
fattest path	$\leq E \ln(E\,U)$	priority queue

digraph with V vertices, E edges, and integer capacities between 1 and U

How to choose augmenting paths?

Choose augmenting paths with:

- Shortest path: fewest number of edges.
- Fattest path: max bottleneck capacity.

Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems

JACK EDMONDS

University of Waterloo, Waterloo, Ontario, Canada

AND

RICHARD M. KARP

University of California, Berkeley, California

ABSTRACT. This paper presents new algorithms for the maximum flow problem, the Hitchcock transportation problem, and the general minimum-cost flow problem. Upper bounds on the numbers of steps in these algorithms are derived, and are shown to compare favorably with upper bounds on the numbers of steps required by earlier algorithms.

Edmonds-Karp 1972 (USA)

Dokl. Akad. Nauk SSSR
Tom 194 (1970), No. 4

Soviet Math. Dokl.
Vol. 11 (1970), No. 5

ALGORITHM FOR SOLUTION OF A PROBLEM OF MAXIMUM FLOW IN A NETWORK WITH POWER ESTIMATION

UDC 518.5

E. A. DINIC

Different variants of the formulation of the problem of maximal stationary flow in a network and its many applications are given in [1]. There also is given an algorithm solving the problem in the case where the initial data are integers (or, what is equivalent, commensurable). In the general case this algorithm requires preliminary rounding off of the initial data, i.e. only an approximate solution of the problem is possible. In this connection the rapidity of convergence of the algorithm is inversely proportional to the relative precision.

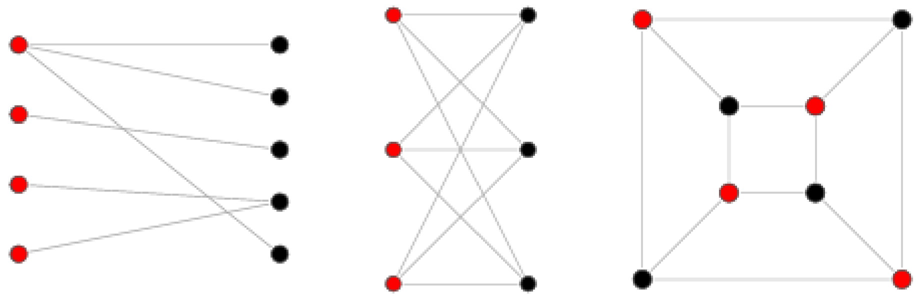
Dinic 1970 (Soviet Union)

Implementation

- Refer to the book

Applications – bipartite matching

- Matching applications to jobs, ads to ad slots (with associated revenue), matching jobs to servers in load balancing etc
- Bipartite graph – A bipartite graph (bigraph,) is a set of graph vertices decomposed into two disjoint sets such that no two graph vertices within the same set are adjacent



Bipartite matching problem

N students apply for N jobs.

Each gets several offers.

Is there a way to match all students to jobs?

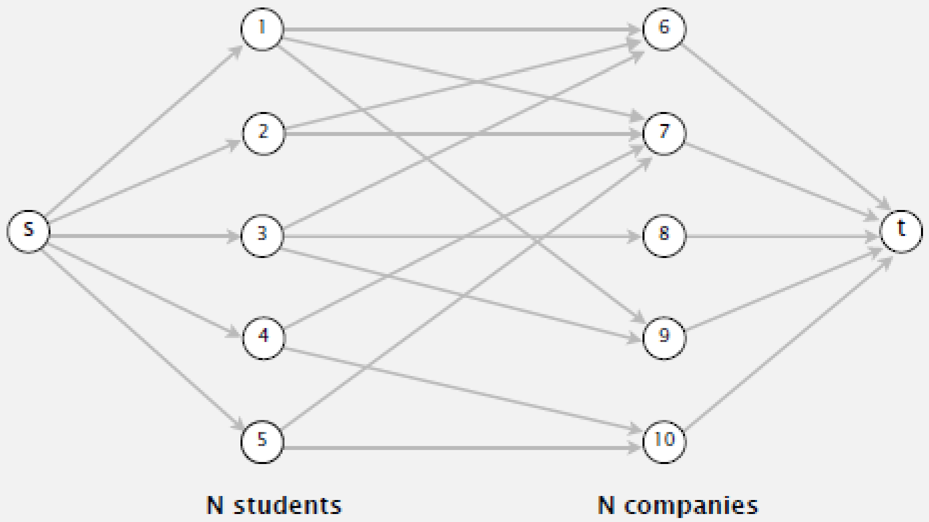
bipartite matching problem

1	Alice	6	Adobe
	Adobe		Alice
	Amazon		Bob
	Google		Carol
2	Bob	7	Amazon
	Adobe		Alice
	Amazon		Bob
3	Carol		Dave
	Adobe		Eliza
	Faceboo k	8	Facebook
	Google		Carol
4	Dave	9	Google
	Amazon		Alice
	Yahoo		Carol
5	Eliza	10	Yahoo
	Amazon		Dave
	Yahoo		Eliza

Network flow formulation of bipartite matching

- Create s, t, one vertex for each student, and one vertex for each job.
- Add edge from s to each student (capacity 1).
- Add edge from each job to t (capacity 1).
- Add edge from student to each job offered (infinite capacity).

flow network



bipartite matching problem

1 Alice
- Adobe
- Amazon
- Google

2 Bob
- Adobe
- Amazon

3 Carol
- Adobe
- Facebook
- Google

4 Dave
- Amazon
- Yahoo

5 Eliza
- Amazon
- Yahoo

6 Adobe
- Alice
- Bob
- Carol

7 Amazon
- Alice
- Bob
- Dave
- Eliza

8 Facebook
- Carol

9 Google
- Alice
- Carol

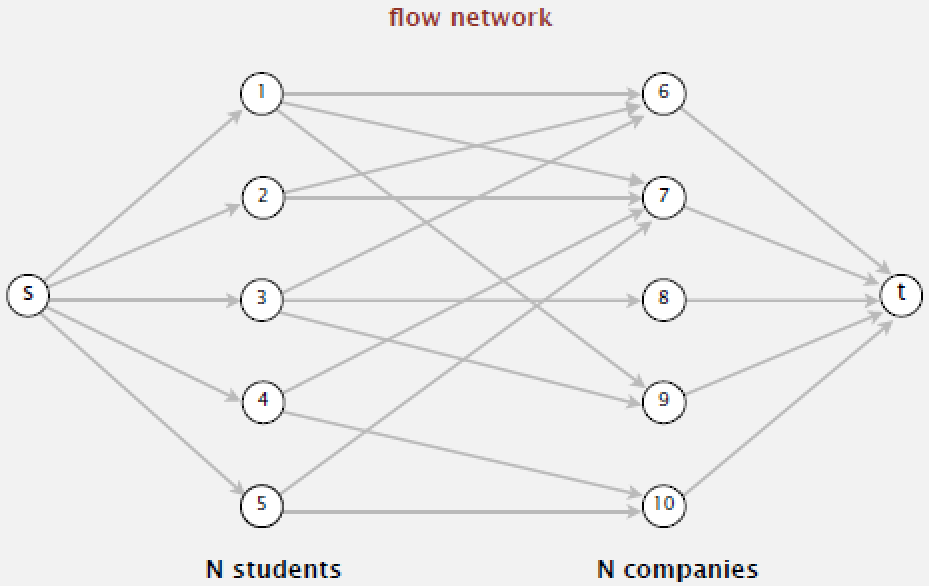
10 Yahoo
- Dave
- Eliza

Really good demo of bipartite matching

http://rosulek.github.io/vamonos/demos/bipartite-matching.html

Network flow formulation of bipartite matching

- Create s, t, one vertex for each student, and one vertex for each job.
- Add edge from s to each student (capacity 1).
- Add edge from each job to t (capacity 1).
- Add edge from student to each job offered (infinite capacity).

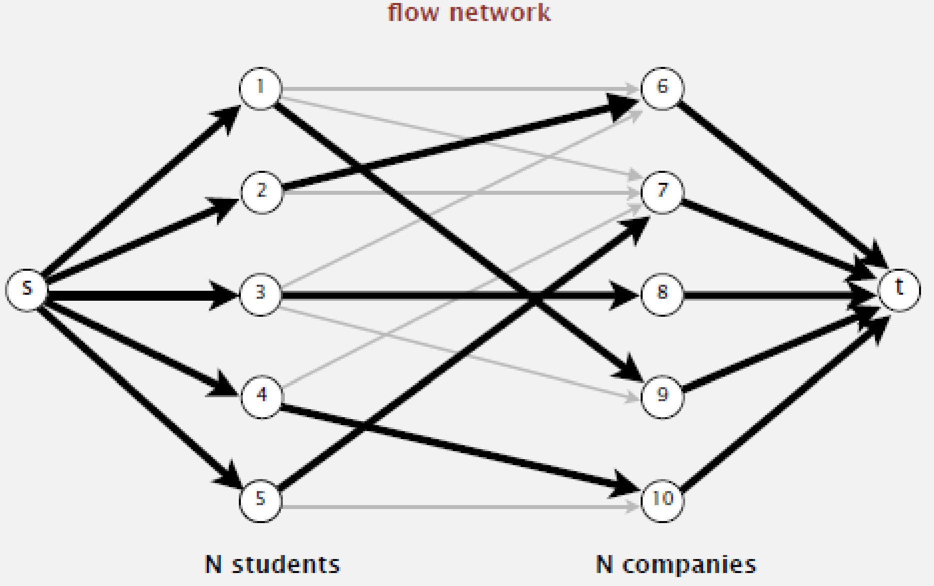


bipartite matching problem

1	Alice	6	Adobe
	Adobe		Alice
	Amazon		Bob
	Google		Carol
2	Bob	7	Amazon
	Adobe		Alice
	Amazon		Bob
3	Carol		Dave
	Adobe		Eliza
	Facebook	8	Facebook
	Google		Carol
4	Dave	9	Google
	Amazon		Alice
	Yahoo		Carol
5	Eliza	10	Yahoo
	Amazon		Dave
	Yahoo		Eliza

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value N.



bipartite matching problem

1 Alice	6 Adobe
Adobe	Alice
Amazon	Bob
Google	Carol
2 Bob	7 Amazon
Adobe	Alice
Amazon	Bob
3 Carol	Dave
Adobe	Eliza
Facebook	8 Facebook
Google	Carol
4 Dave	9 Google
Amazon	Alice
Yahoo	Carol
5 Eliza	10 Yahoo
Amazon	Dave
Yahoo	Eliza

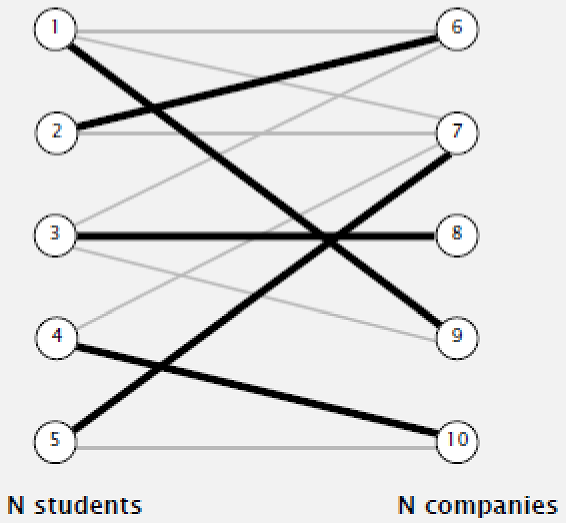
Bipartite matching problem

Given a bipartite graph, find a perfect matching.

perfect matching (solution)

Alice	——	Google
Bob	——	Adobe
Carol	——	Facebook
Dave	——	Yahoo
Eliza	——	Amazon

bipartite graph



bipartite matching problem

1 Alice
- Adobe
- Amazon
- Google

2 Bob
- Adobe
- Amazon

3 Carol
- Adobe
- Facebook
- Google

4 Dave
- Amazon
- Yahoo

5 Eliza
- Amazon
- Yahoo

6 Adobe
- Alice
- Bob
- Carol

7 Amazon
- Alice
- Bob
- Dave
- Eliza

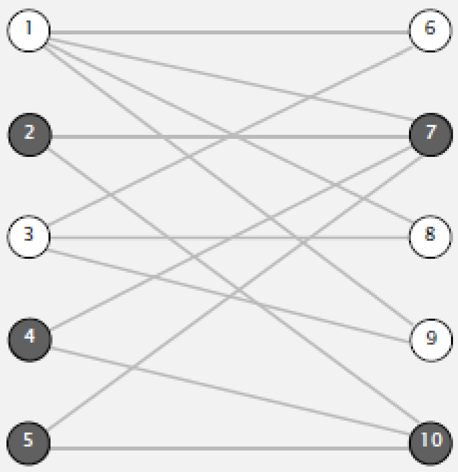
8 Facebook
- Carol

9 Google
- Alice
- Carol

10 Yahoo
- Dave
- Eliza

What the mincut tells us

Goal. When no perfect matching, explain why.



no perfect matching exists

S = { 2, 4, 5 }
T = { 7, 10 }

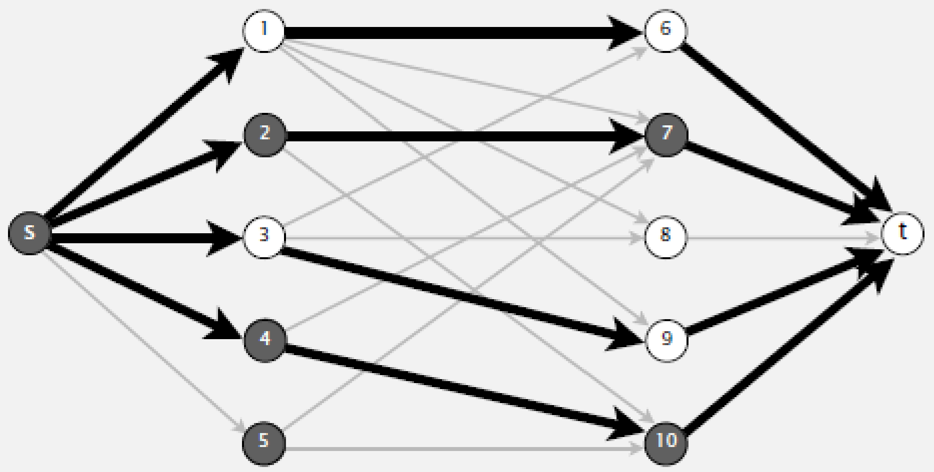
student in S
can be matched
only to
companies in T

| S | > | T |

What the mincut tells us

Mincut. Consider mincut (A, B).

- Let S = students on s side of cut.
- Let T = companies on s side of cut.
- Fact: $|S| > |T|$; students in S can be matched only to companies in T.



no perfect matching exists

S = { 2, 4, 5 }
T = { 7, 10 }

student in S can be matched only to companies in T

| S | > | T |

Bottom line. When no perfect matching, mincut explains why.

Multi-source multi-sink networks?

› Add super source and super sink

Additional material

- Good tutorial

https://www.topcoder.com/community/data-science/data-science-tutorials/maximum-flow-section-1/