CS2010: ALGORITHMS AND DATA STRUCTURES

Lecture 22: Max Flow – Min Cut Optimization Problems

Vasileios Koutavas

School of Computer Science and Statistics
Trinity College Dublin

http://algs4.cs.princeton.edu

6.4 MAXIMUM FLOW

- *introduction*
- *Ford-Fulkerson algorithm*
- *maxflow-mincut theorem*
- *analysis of running time*
- *Java implementation*
- *applications*

http://algs4.cs.princeton.edu

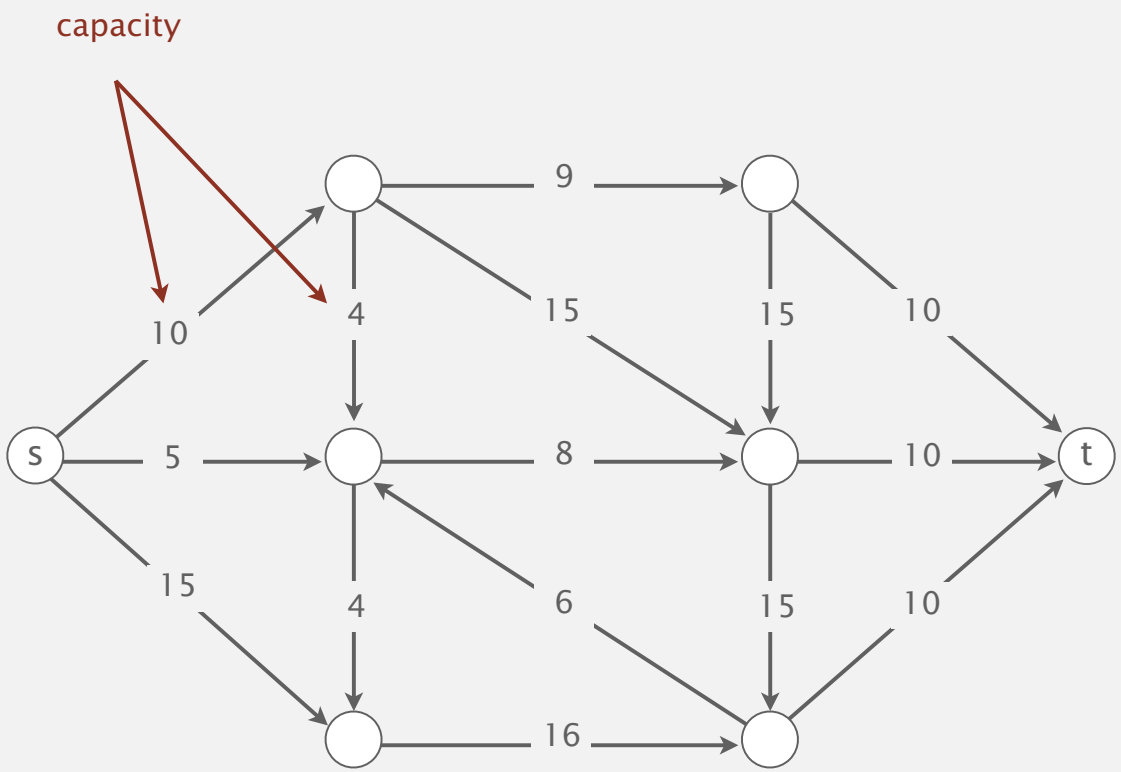
6.4 Maximum Flow

- *introduction*
- *Ford-Fulkerson algorithm*
- *maxflow-mincut theorem*
- *analysis of running time*
- *Java implementation*
- *applications*

Mincut problem

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

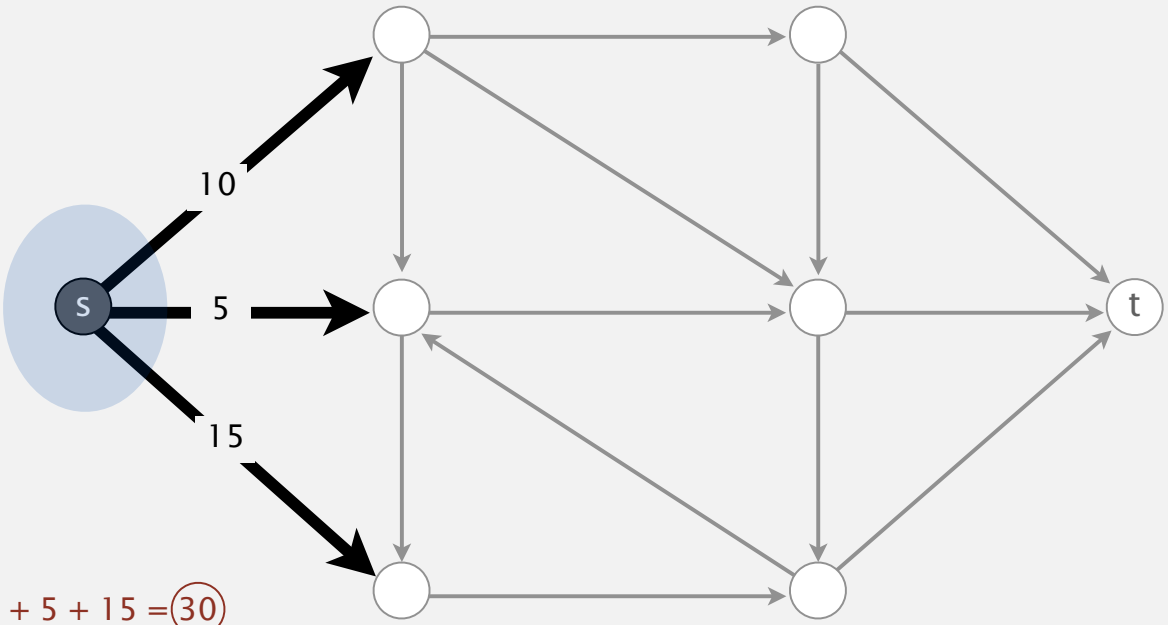
each edge has a positive capacity



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.

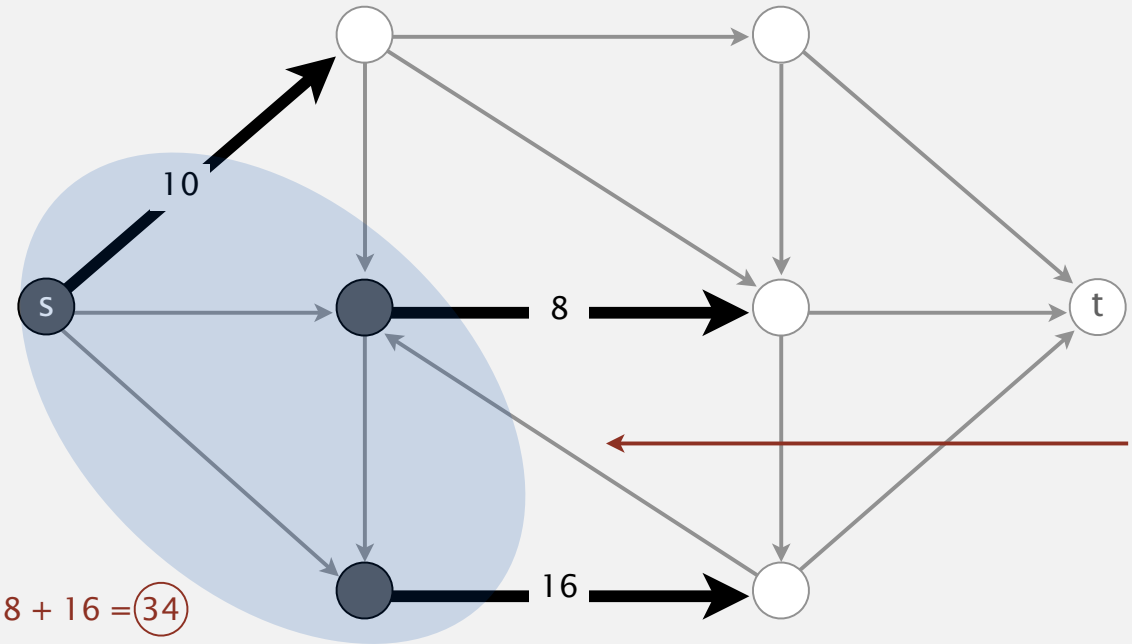


capacity = 10 + 5 + 15 = 30

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.



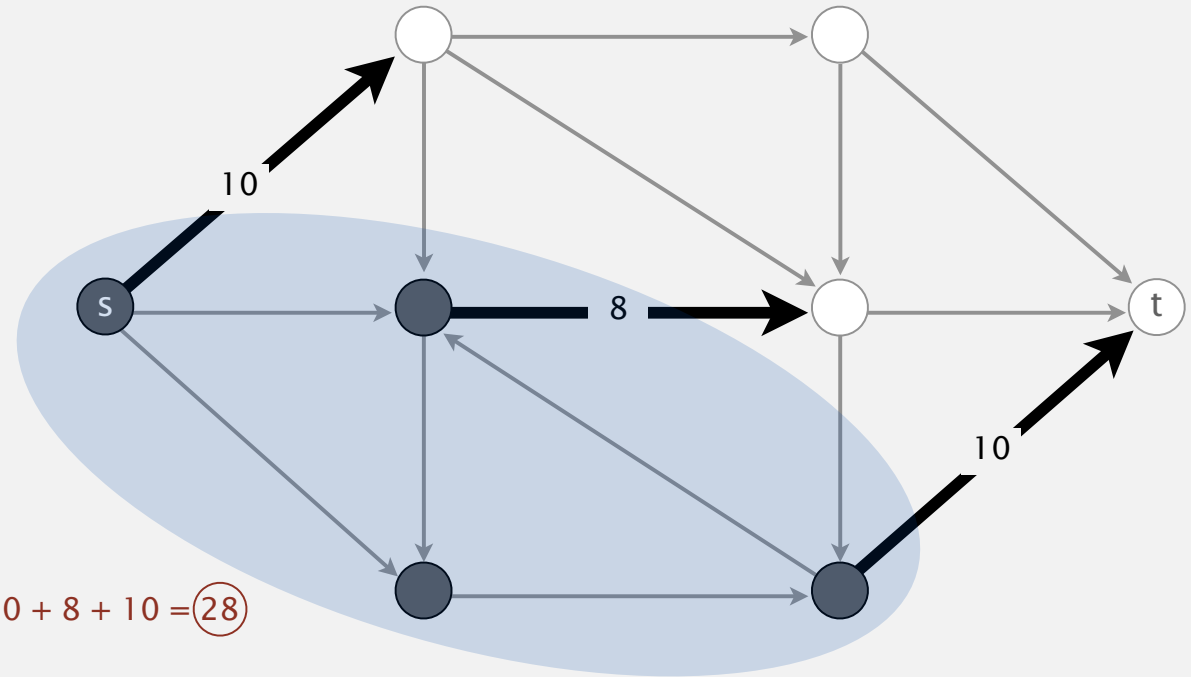
capacity = 10 + 8 + 16 = 34

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.

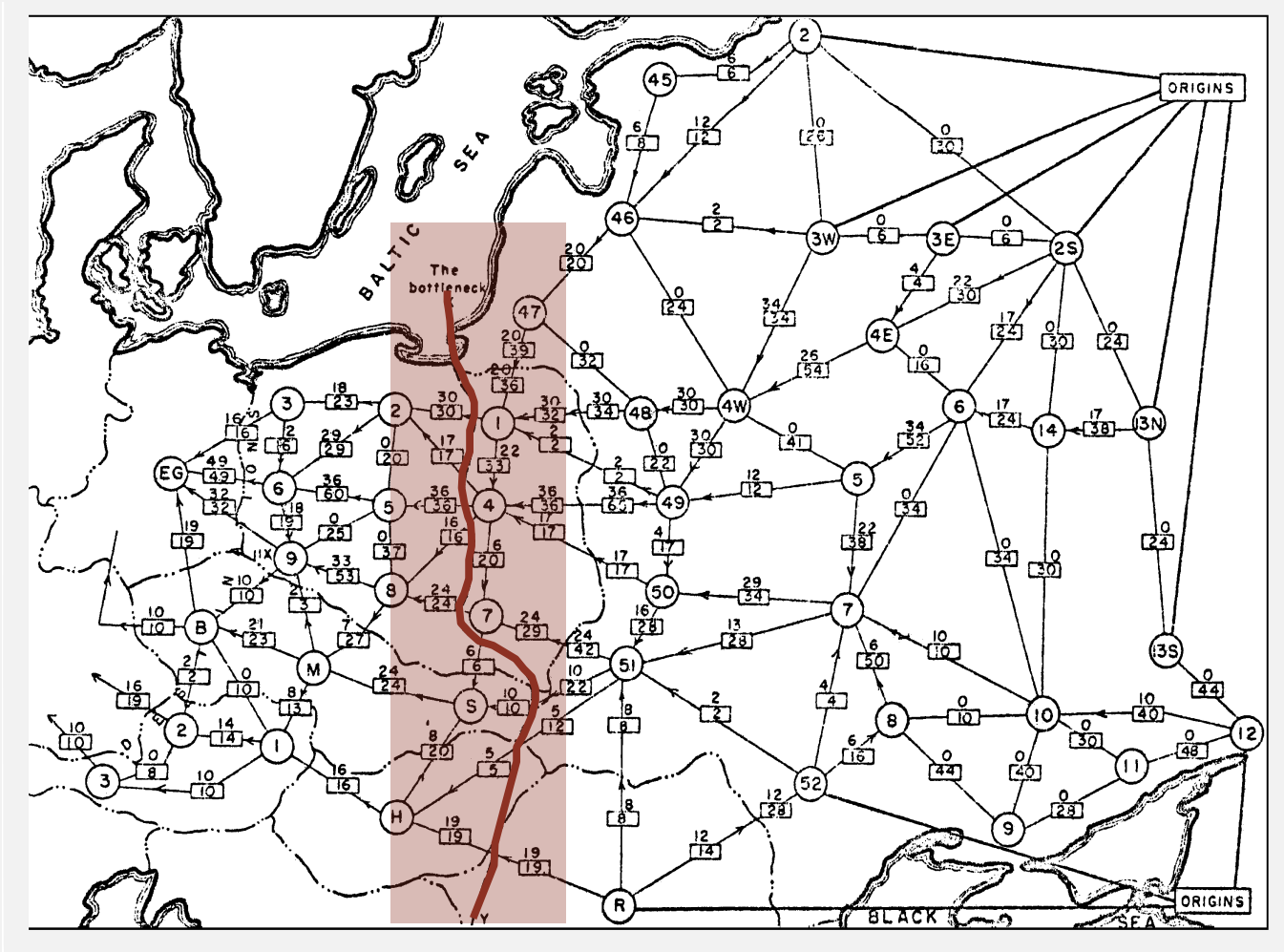
Minimum st-cut (mincut) problem. Find a cut of minimum capacity.



capacity = 10 + 8 + 10 = 28

Mincut application (RAND 1950s)

"Free world" goal. Cut supplies (if cold war turns into real war).



rail network connecting Soviet Union with Eastern European countries
(map declassified by Pentagon in 1999)

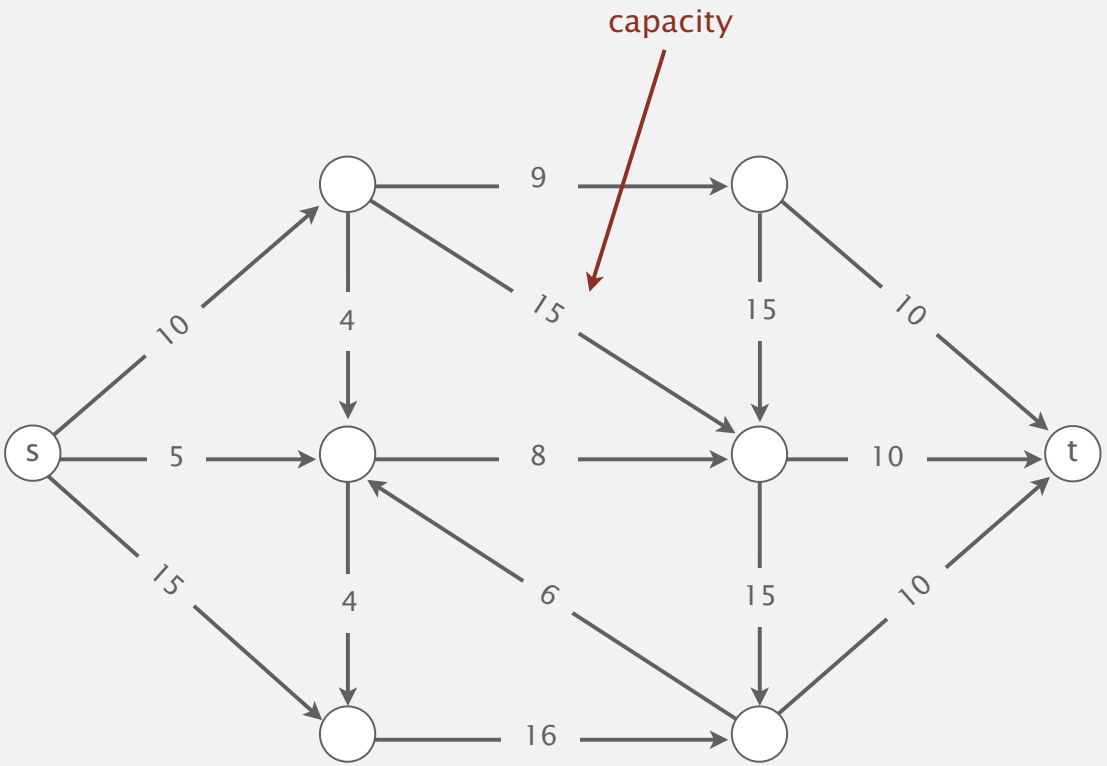
Potential mincut application (2010s)

Government-in-power's goal. Cut off communication to set of people.

Maxflow problem

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

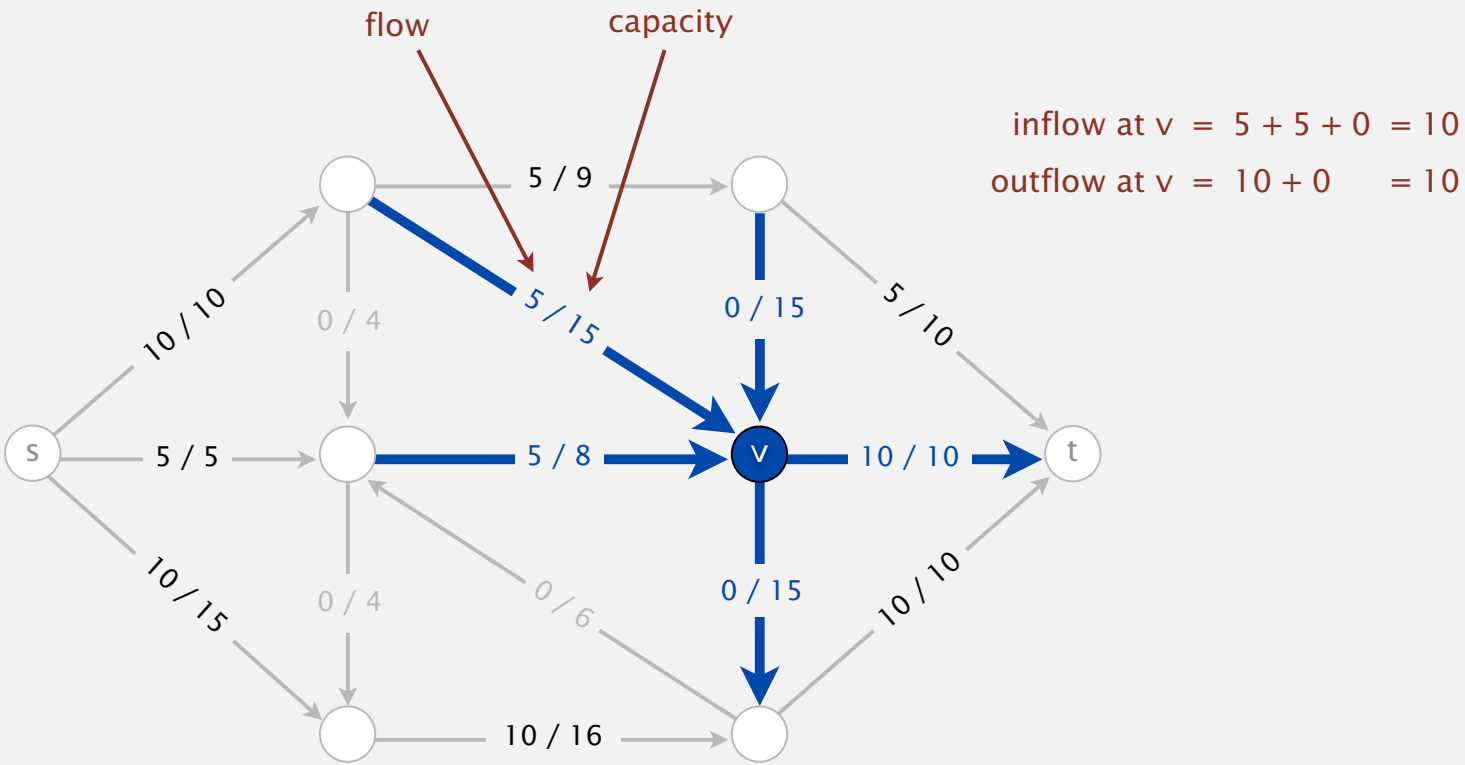
each edge has a positive capacity



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).



inflow at v = 5 + 5 + 0 = 10

outflow at v = 10 + 0 = 10

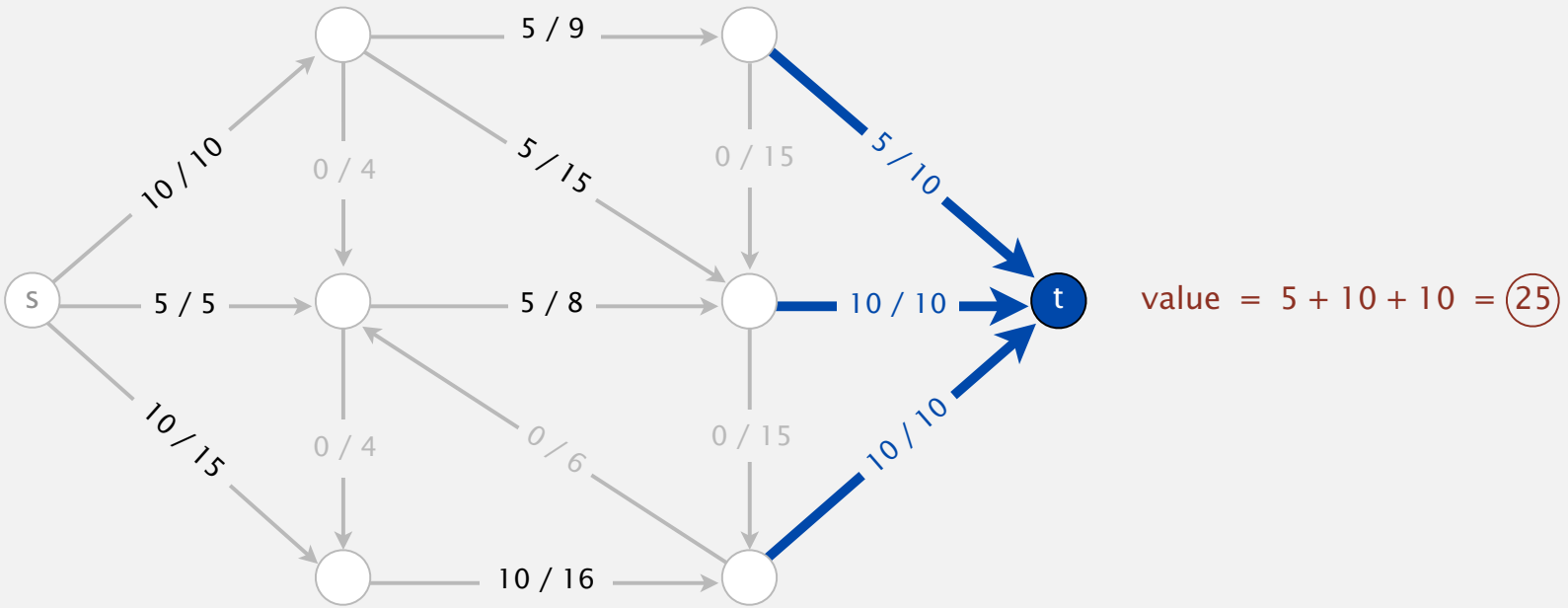
Maxflow problem

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



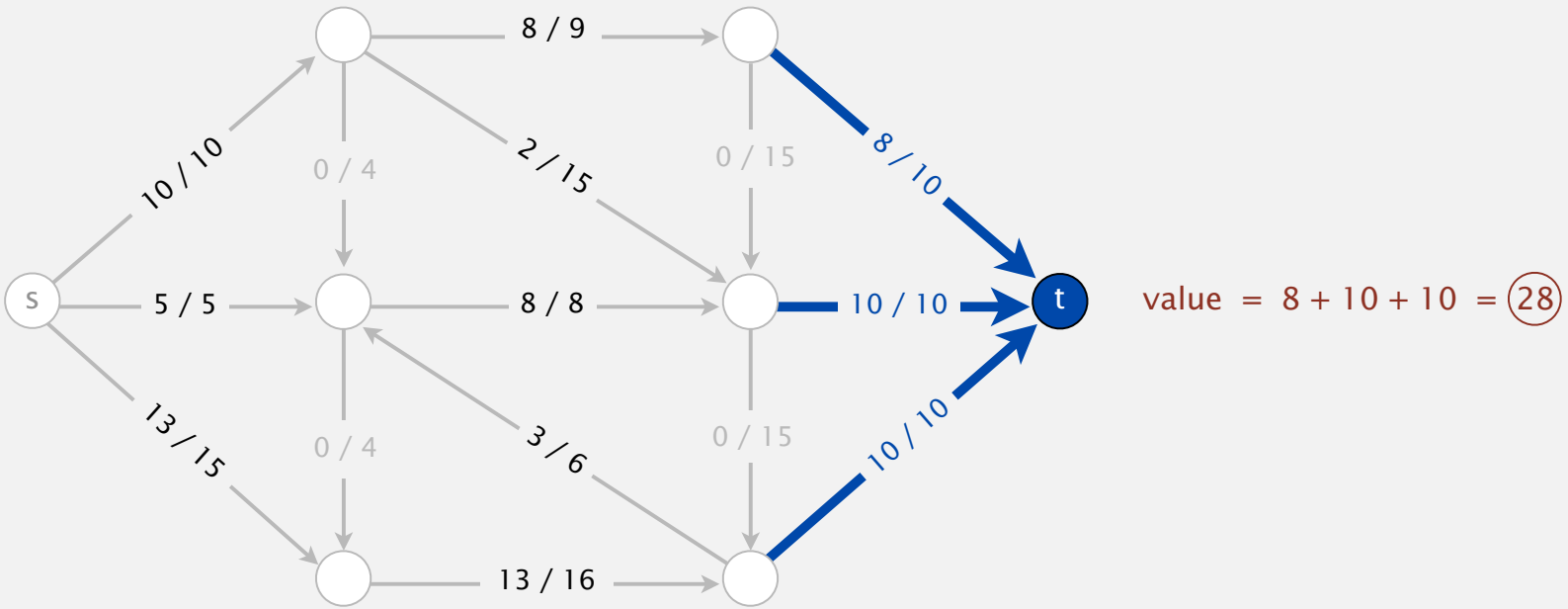
Maxflow problem

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).

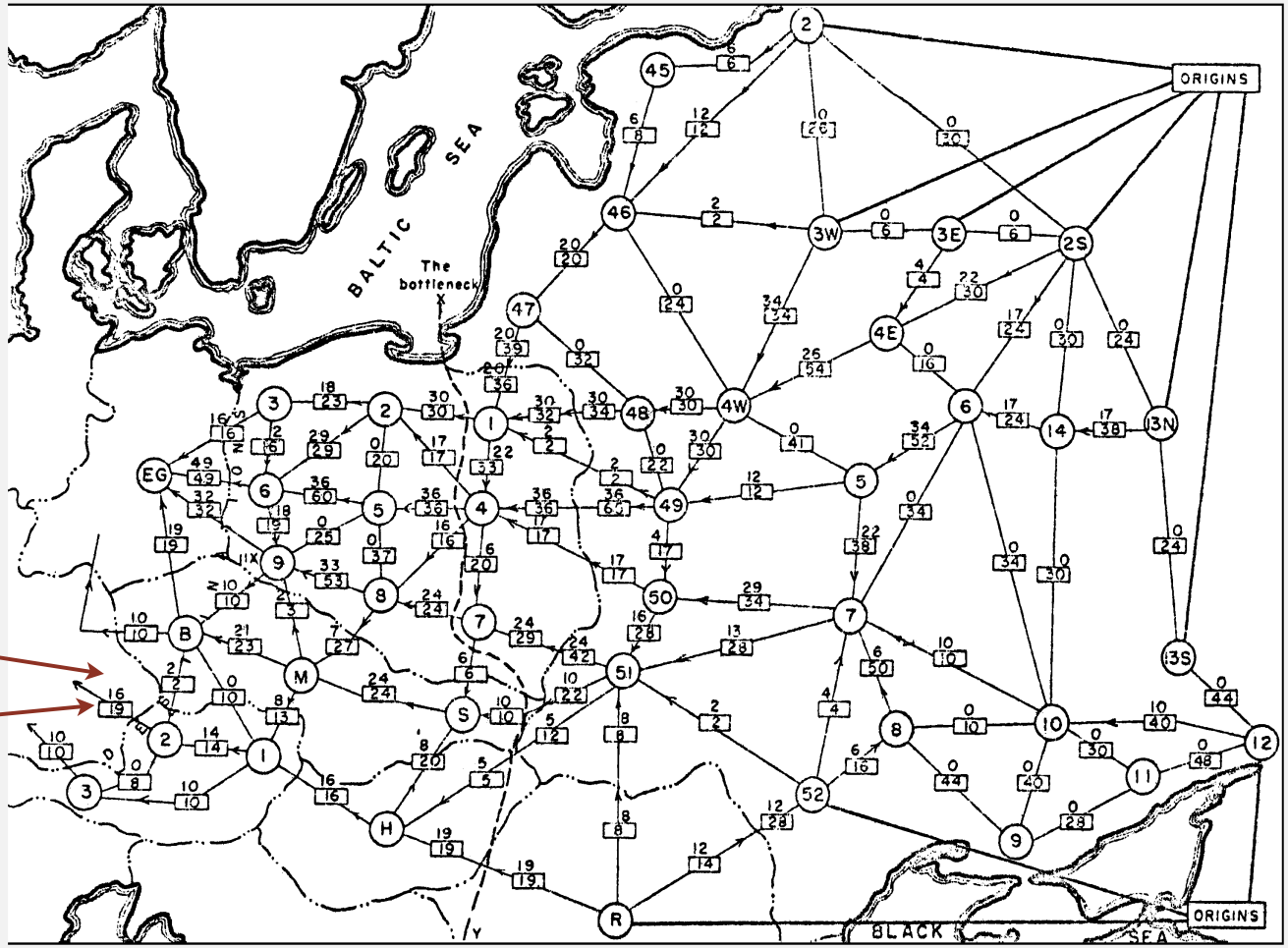
Def. The value of a flow is the inflow at t.

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



Maxflow application (Tolstoĭ 1930s)

Soviet Union goal. Maximize flow of supplies to Eastern Europe.



rail network connecting Soviet Union with Eastern European countries

(map declassified by Pentagon in 1999)

Potential maxflow application (2010s)

"Free world" goal. Maximize flow of information to specified set of people.

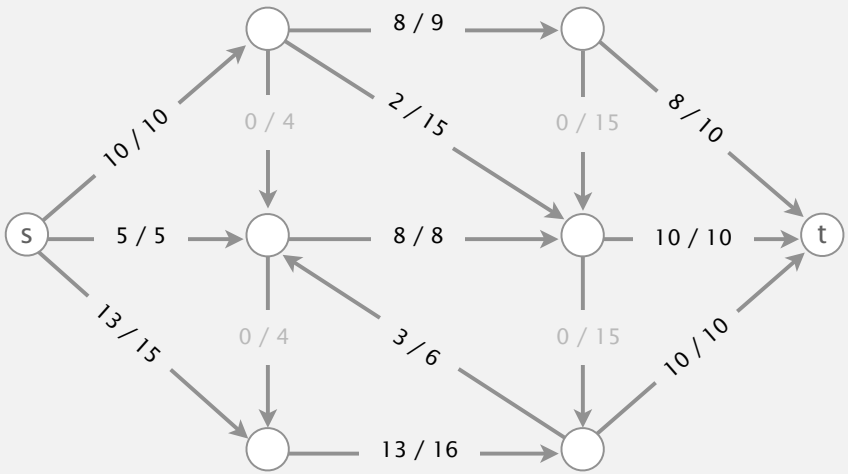
facebook graph

Summary

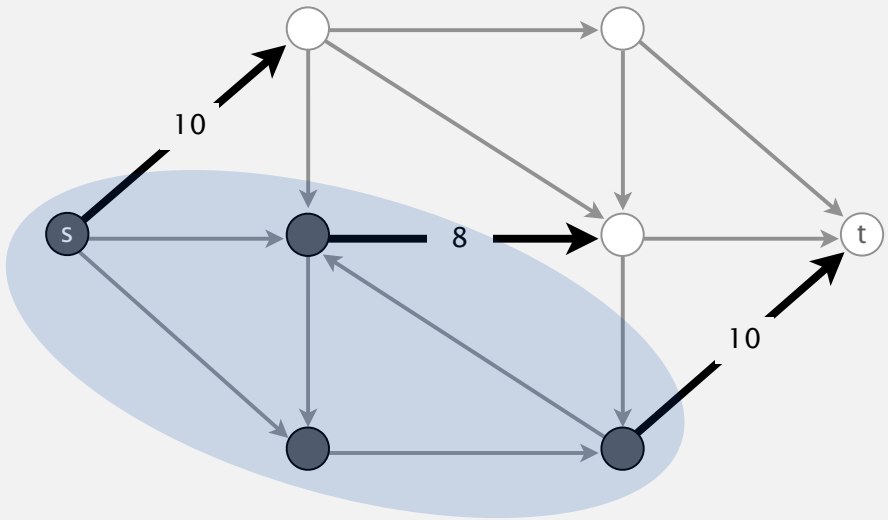
Input. A weighted digraph, source vertex s, and target vertex t.

Mincut problem. Find a cut of minimum capacity.

Maxflow problem. Find a flow of maximum value.



value of flow = 28



capacity of cut = 28

Remarkable fact. These two problems are dual!

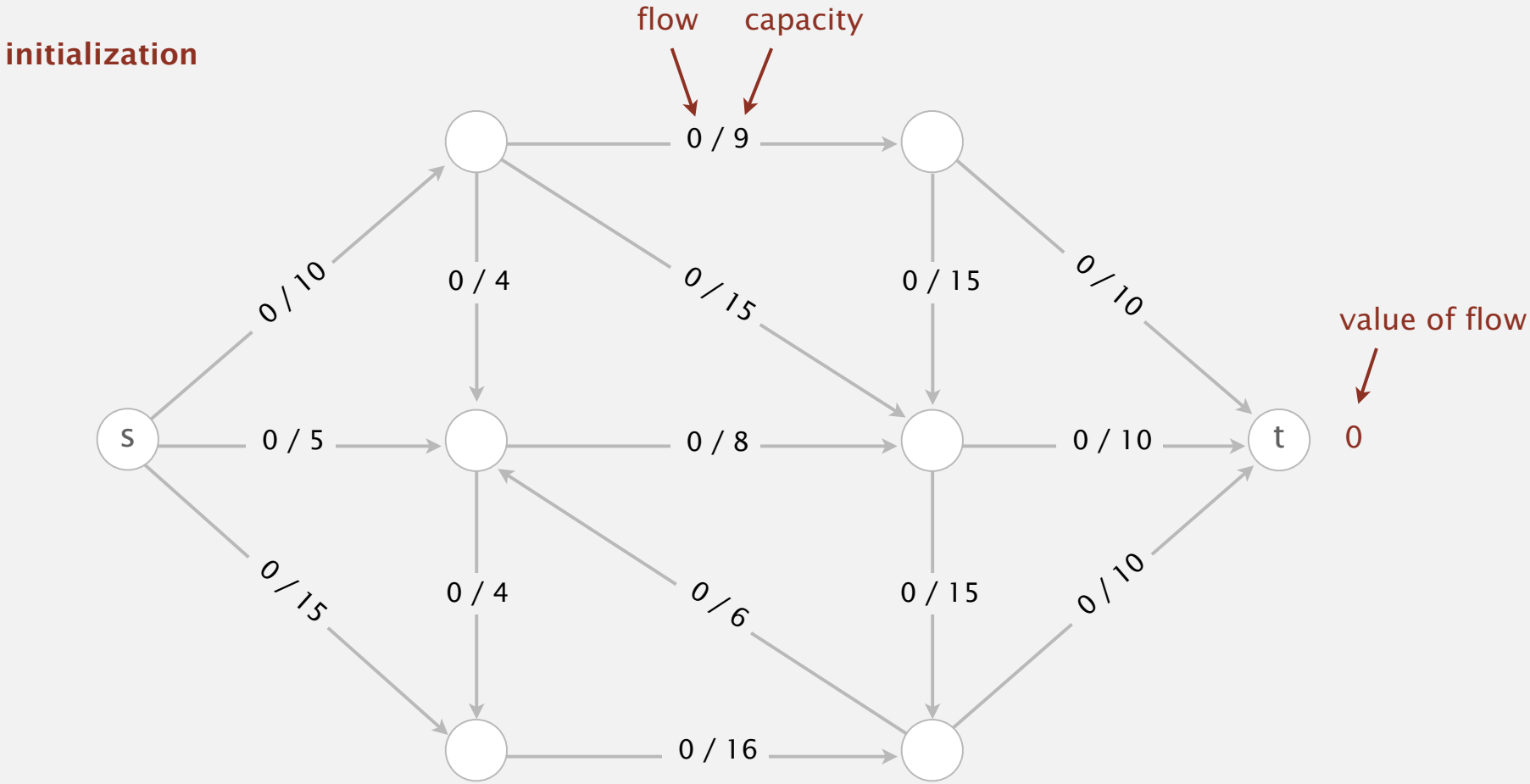
http://algs4.cs.princeton.edu

6.4 Maximum Flow

- *introduction*
- *Ford-Fulkerson algorithm*
- *maxflow-mincut theorem*
- *analysis of running time*
- *Java implementation*
- *applications*

Ford-Fulkerson algorithm

Initialization. Start with 0 flow.

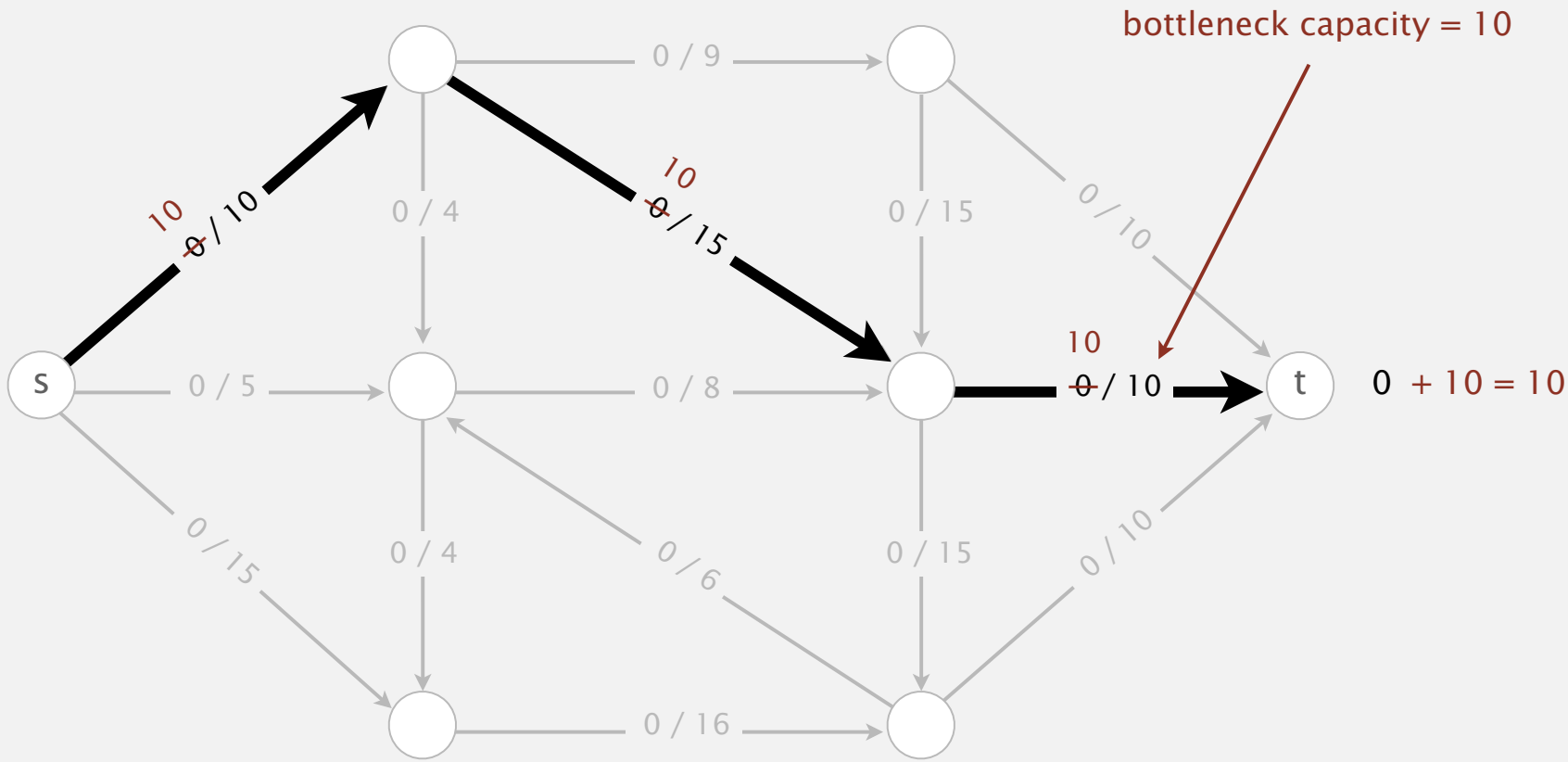


Idea: increase flow along augmenting paths

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

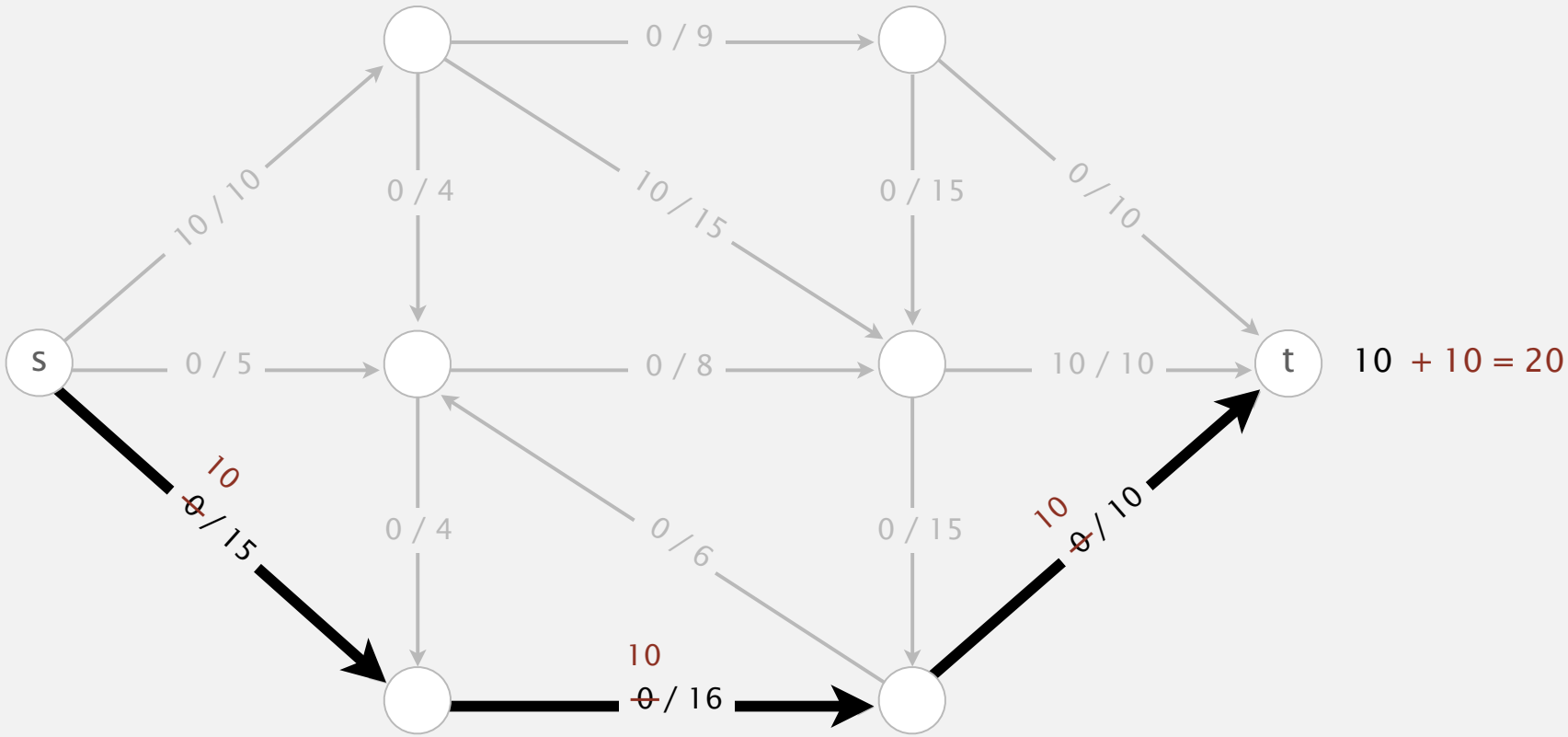


Idea: increase flow along augmenting paths

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).

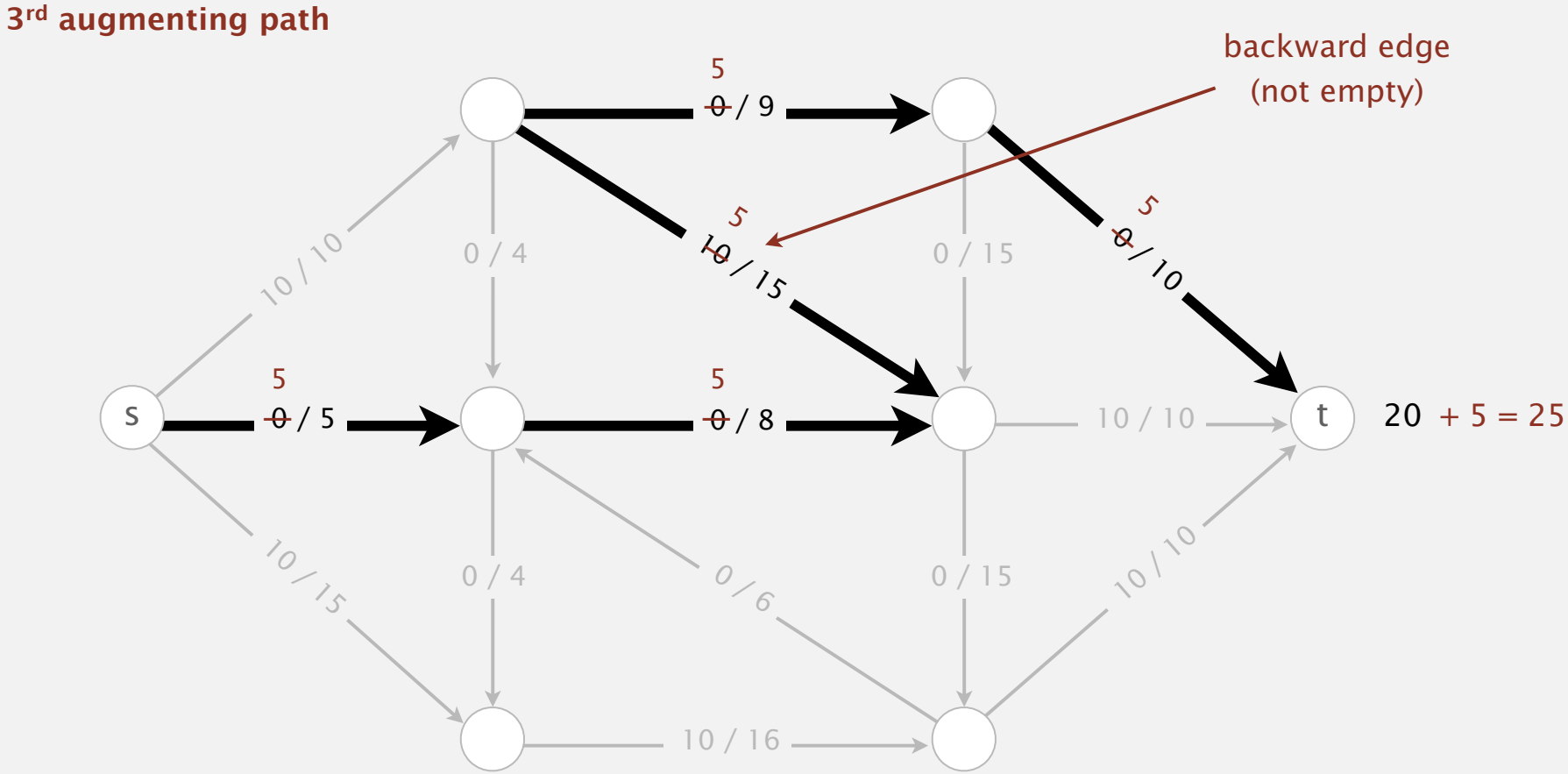
2nd augmenting path



Idea: increase flow along augmenting paths

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

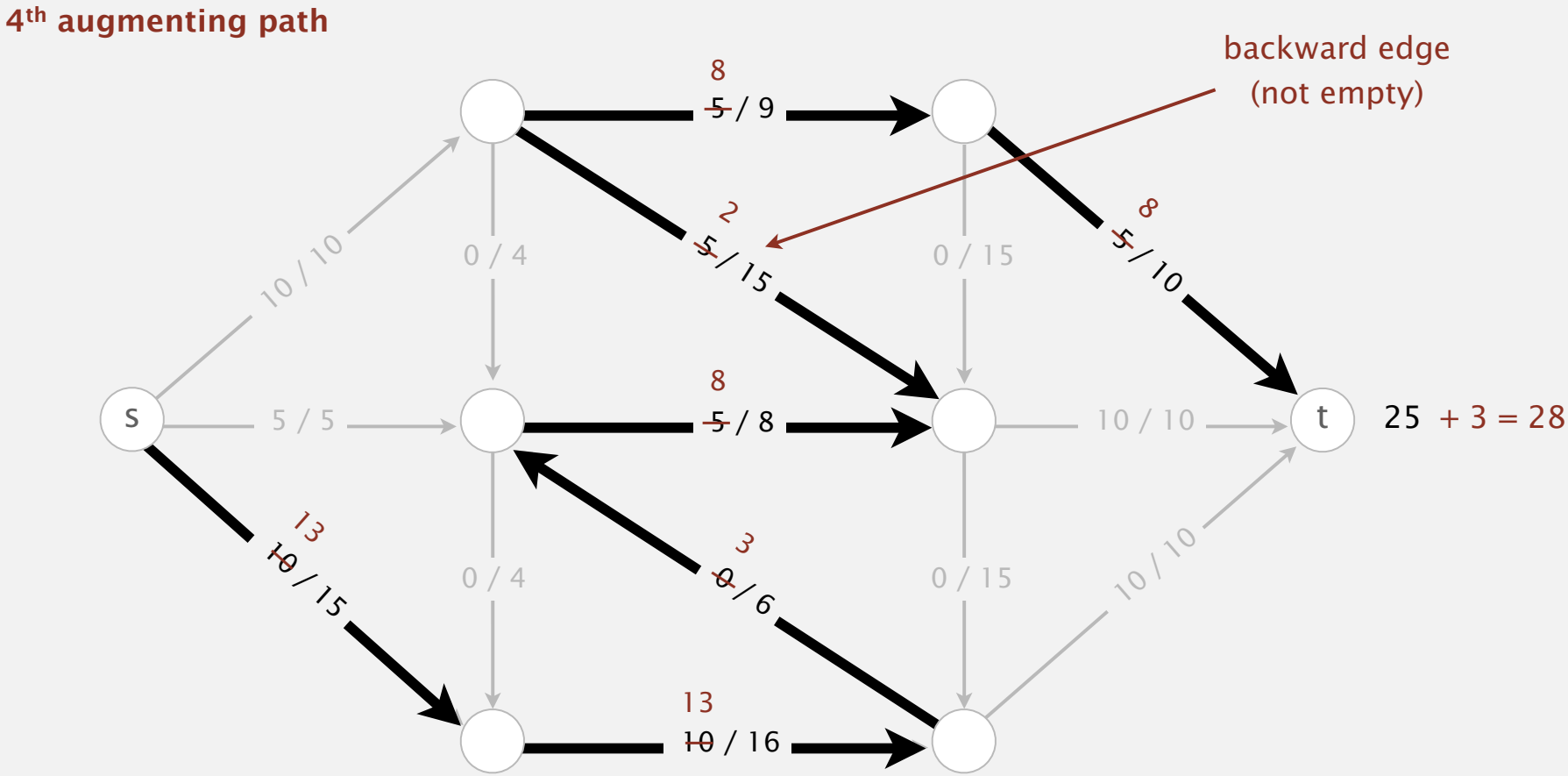
- Can increase flow on forward edges (not full).
- Can decrease flow on backward edge (not empty).



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).

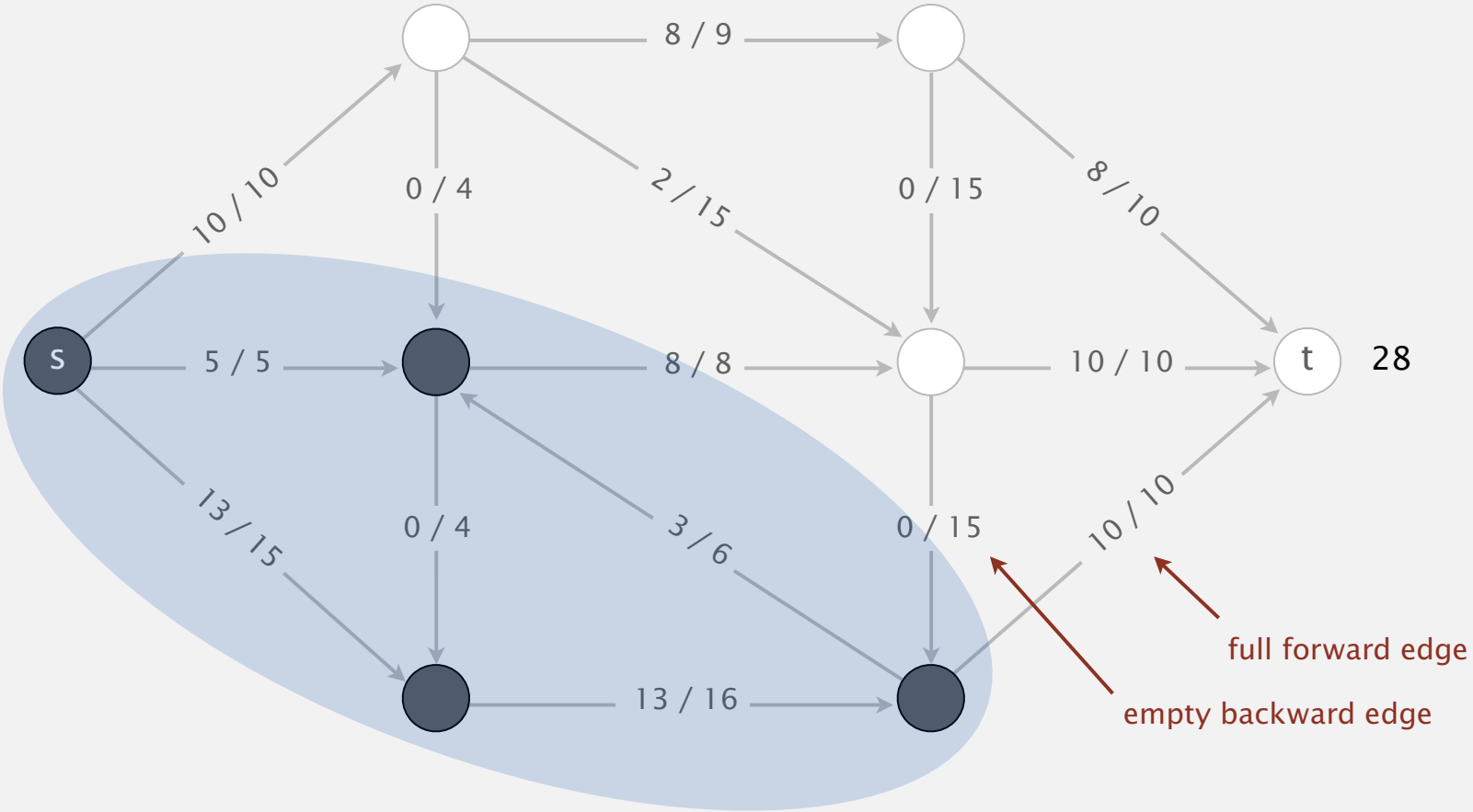


Idea: increase flow along augmenting paths

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

- Full forward edge.
- Empty backward edge.

no more augmenting paths



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?

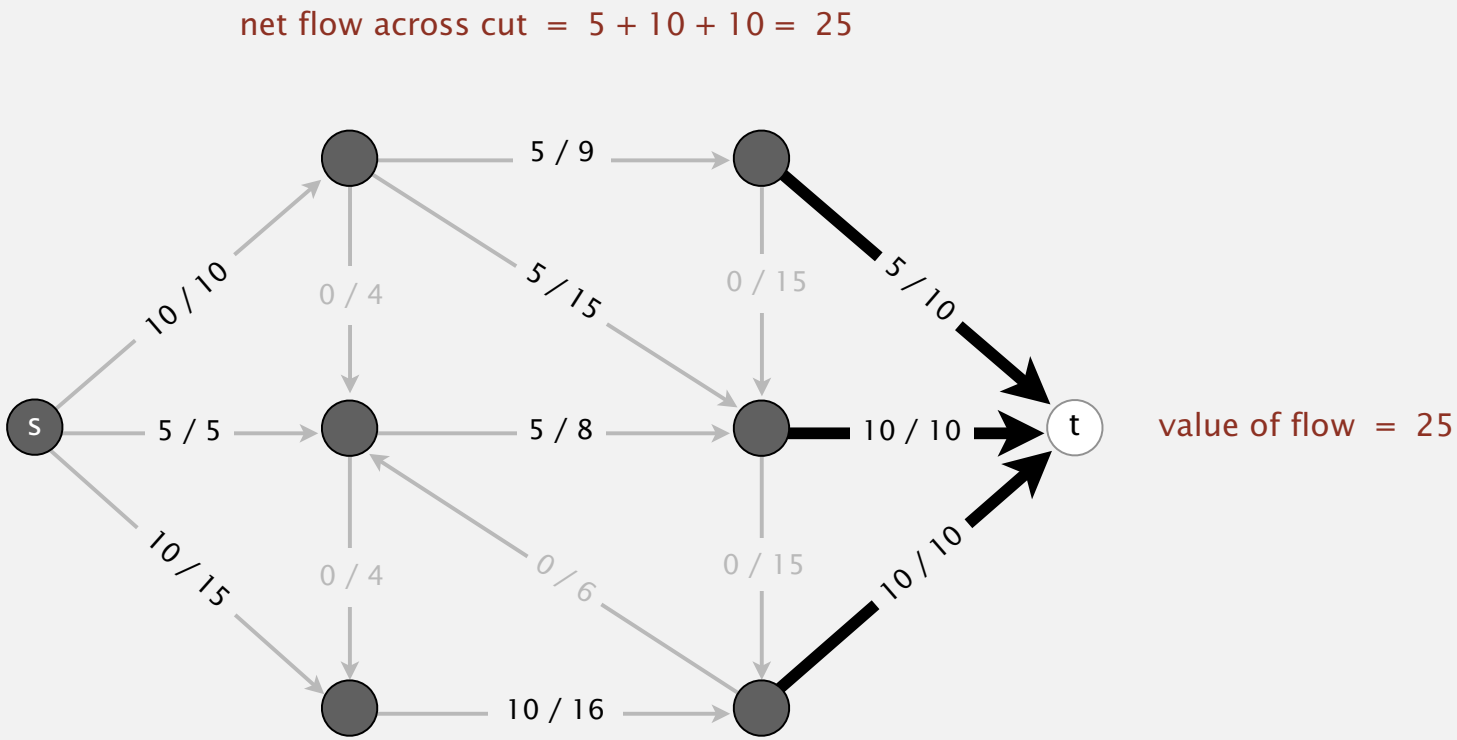
http://algs4.cs.princeton.edu

6.4 Maximum Flow

- *introduction*
- *Ford-Fulkerson algorithm*
- *maxflow-mincut theorem*
- *analysis of running time*
- *Java implementation*
- *applications*

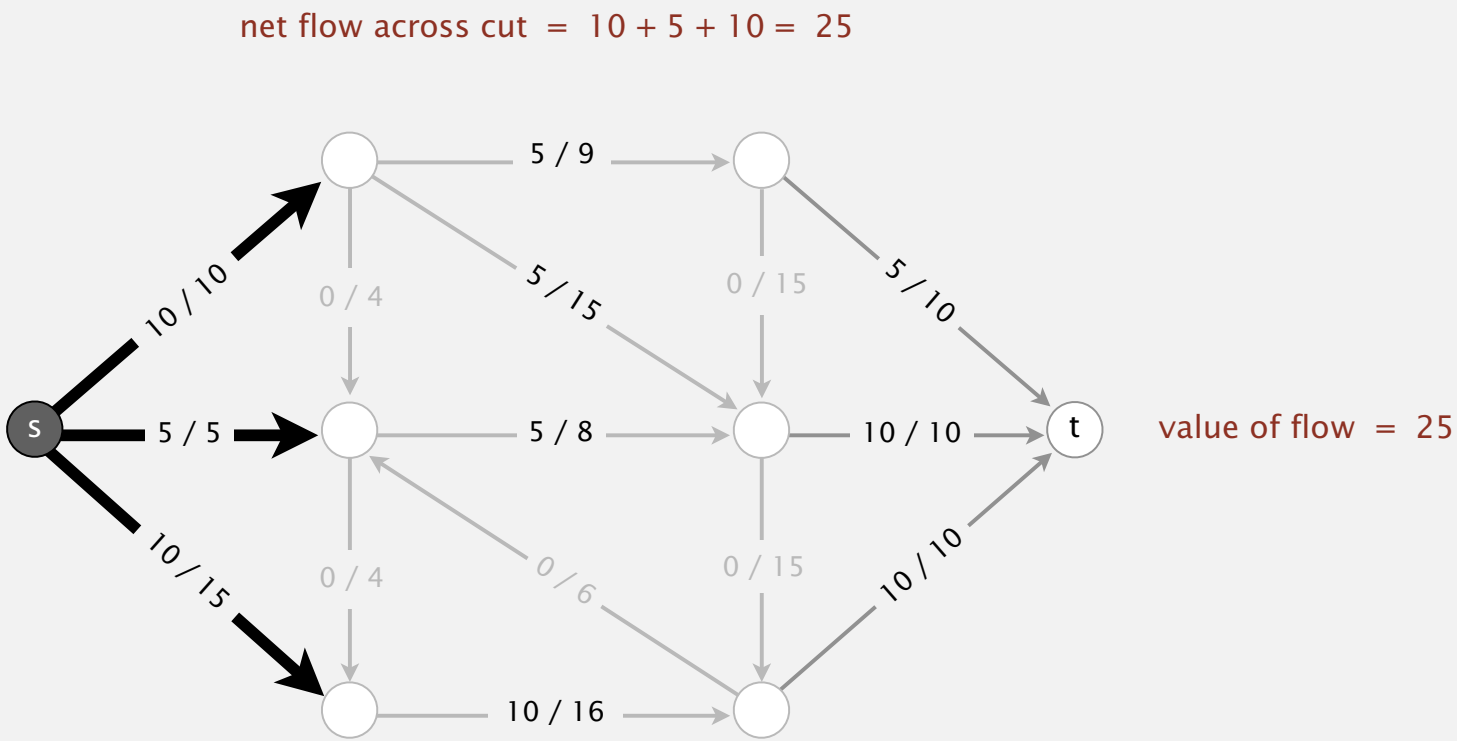
Relationship between flows and cuts

Def. The net flow across a cut (A, B) is the sum of the flows on its edges from A to B minus the sum of the flows on its edges from B to A.



Relationship between flows and cuts

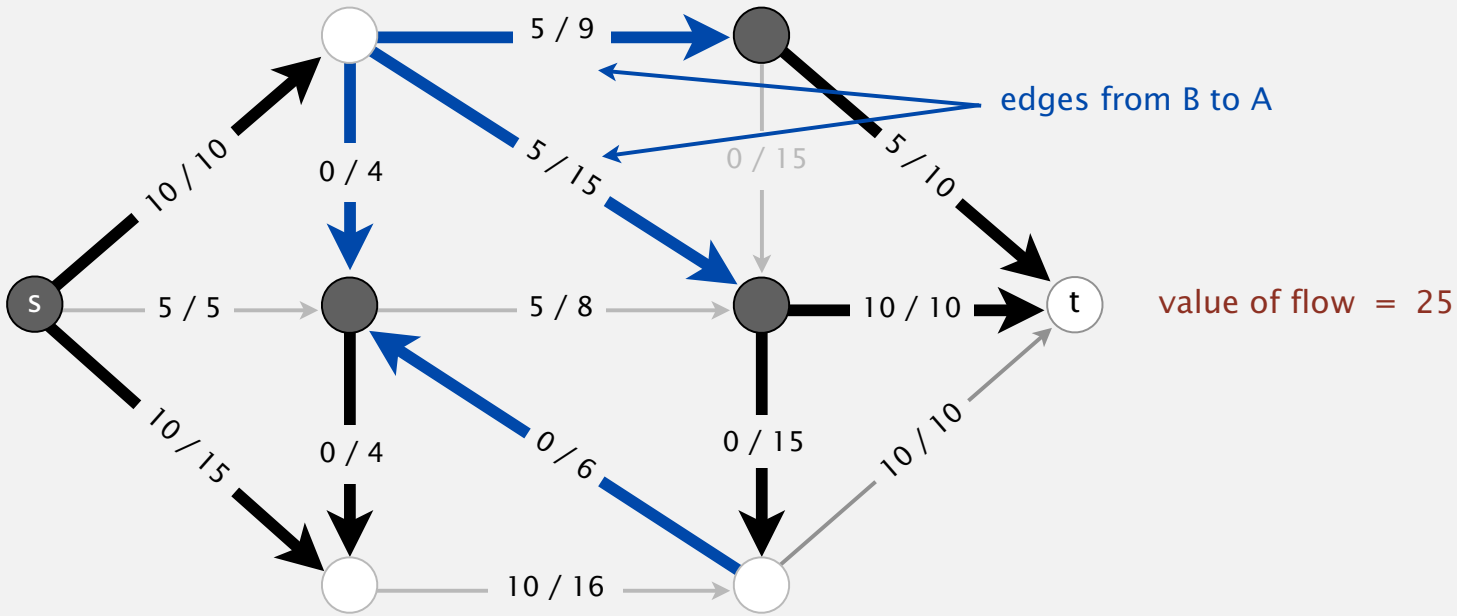
Def. The net flow across a cut (A, B) is the sum of the flows on its edges from A to B minus the sum of the flows on its edges from B to A.



Relationship between flows and cuts

Def. The net flow across a cut (A, B) is the sum of the flows on its edges from A to B minus the sum of the flows on its edges from B to A.

net flow across cut = (10 + 10 + 5 + 10 + 0 + 0) − (5 + 5 + 0 + 0) = 25



Relationship between flows and cuts

Flow-value lemma. Let f be any flow and let (A, B) be any cut. Then, the net flow across (A, B) equals the value of f.

Intuition. Conservation of flow.

Pf. By induction on the size of B.

- Base case: $B = \{ t \}$.
- Induction step: remains true by local equilibrium when moving any vertex from A to B.

Corollary. Outflow from s = inflow to t = value of flow.

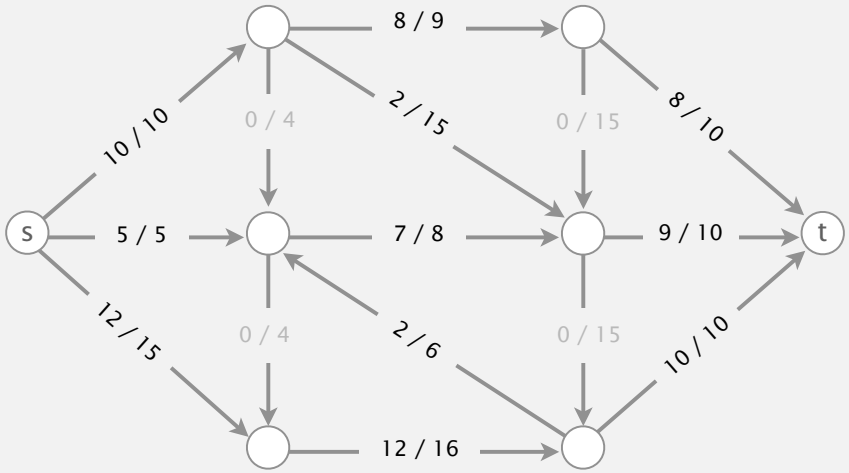
Relationship between flows and cuts

Weak duality. Let f be any flow and let (A, B) be any cut.
Then, the value of the flow ≤ the capacity of the cut.

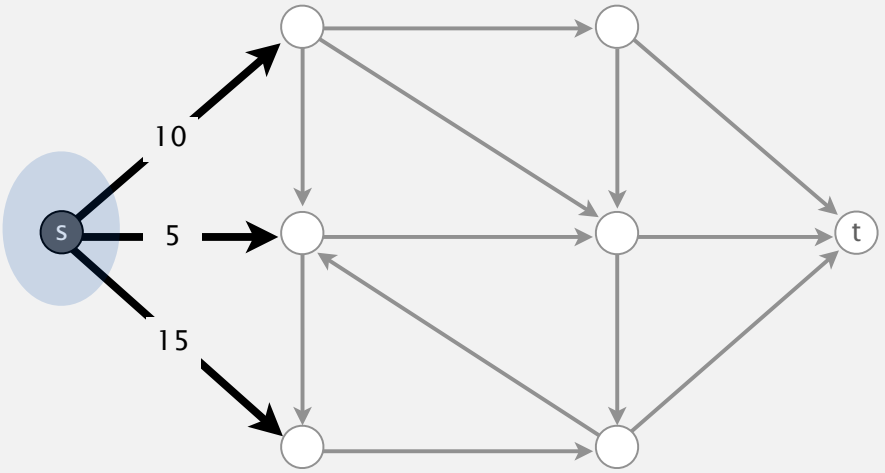
Pf. Value of flow f = net flow across cut (A, B) ≤ capacity of cut (A, B).

flow-value lemma

flow bounded by capacity



value of flow = 27



capacity of cut = 30

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:

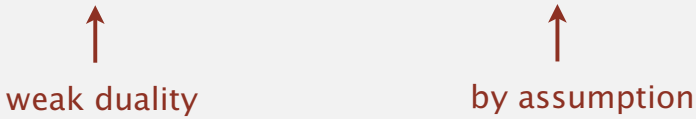
i. There exists a cut whose capacity equals the value of the flow f.

ii. f is a maxflow.

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

[i $\Rightarrow$ ii]

- Suppose that (A, B) is a cut with capacity equal to the value of f.
- Then, the value of any flow f' $\leq$ capacity of (A, B) $=$ value of f.
- Thus, f is a maxflow.



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.

[ii ⇒ iii] We prove contrapositive: ~iii ⇒ ~ii.

- Suppose that there is an augmenting path with respect to f.
- Can improve flow f by sending flow along this path.
- Thus, f is not a maxflow.

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.

[iii ⇒ i]

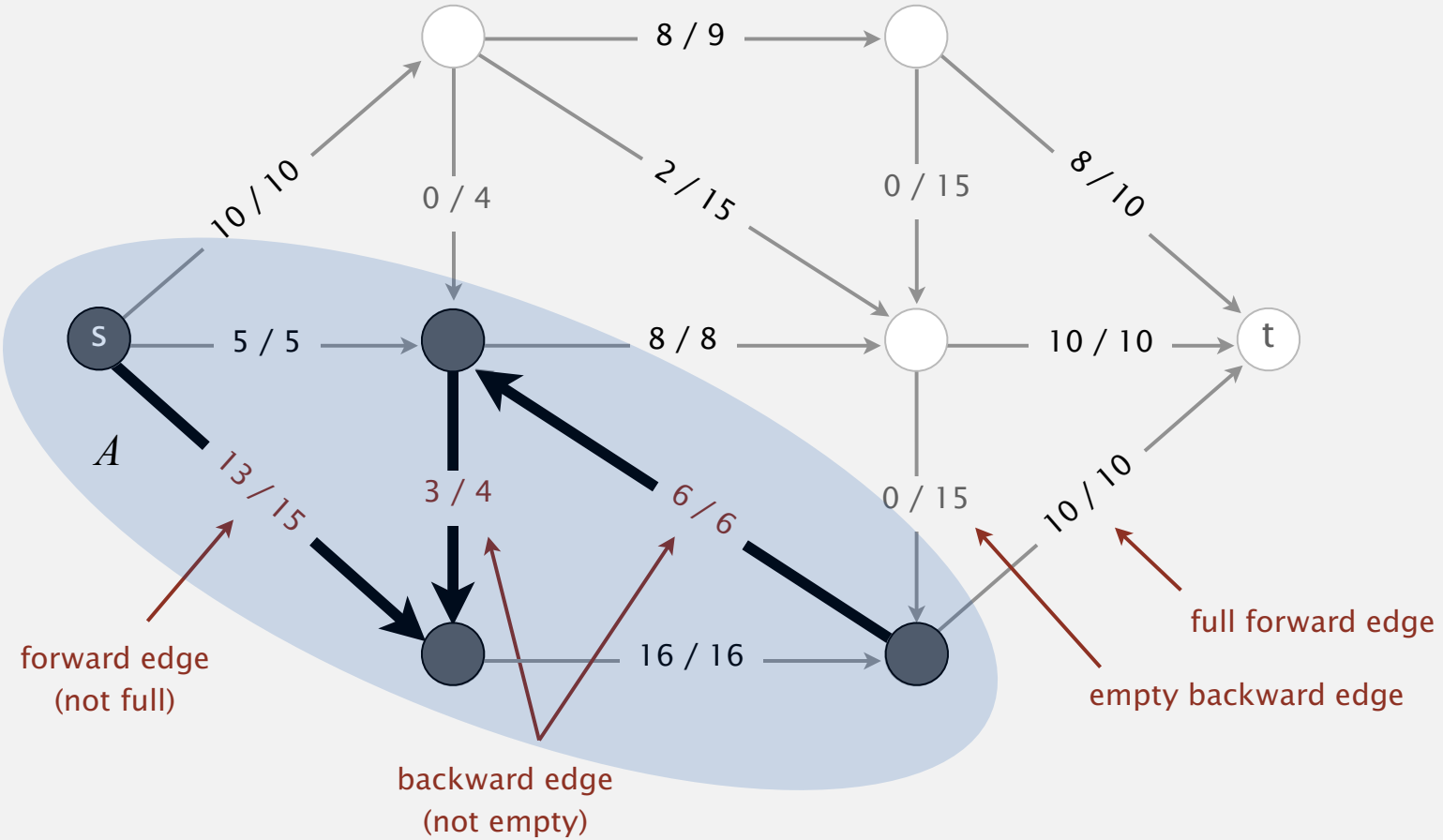
Suppose that there is no augmenting path with respect to f.

- Let (A, B) be a cut where A is the set of vertices connected to s by an undirected path with no full forward or empty backward edges.
- By definition of cut, s is in A.
- Since no augmenting path, t is in B.
- Capacity of cut = net flow across cut ← forward edges full; backward edges empty
  = value of flow f. ← flow-value lemma

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.



http://algs4.cs.princeton.edu

6.4 Maximum Flow

- *introduction*
- *Ford-Fulkerson algorithm*
- *maxflow-mincut theorem*
- *analysis of running time*
- *Java implementation*
- *applications*

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 ≤ the value of the maxflow.

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

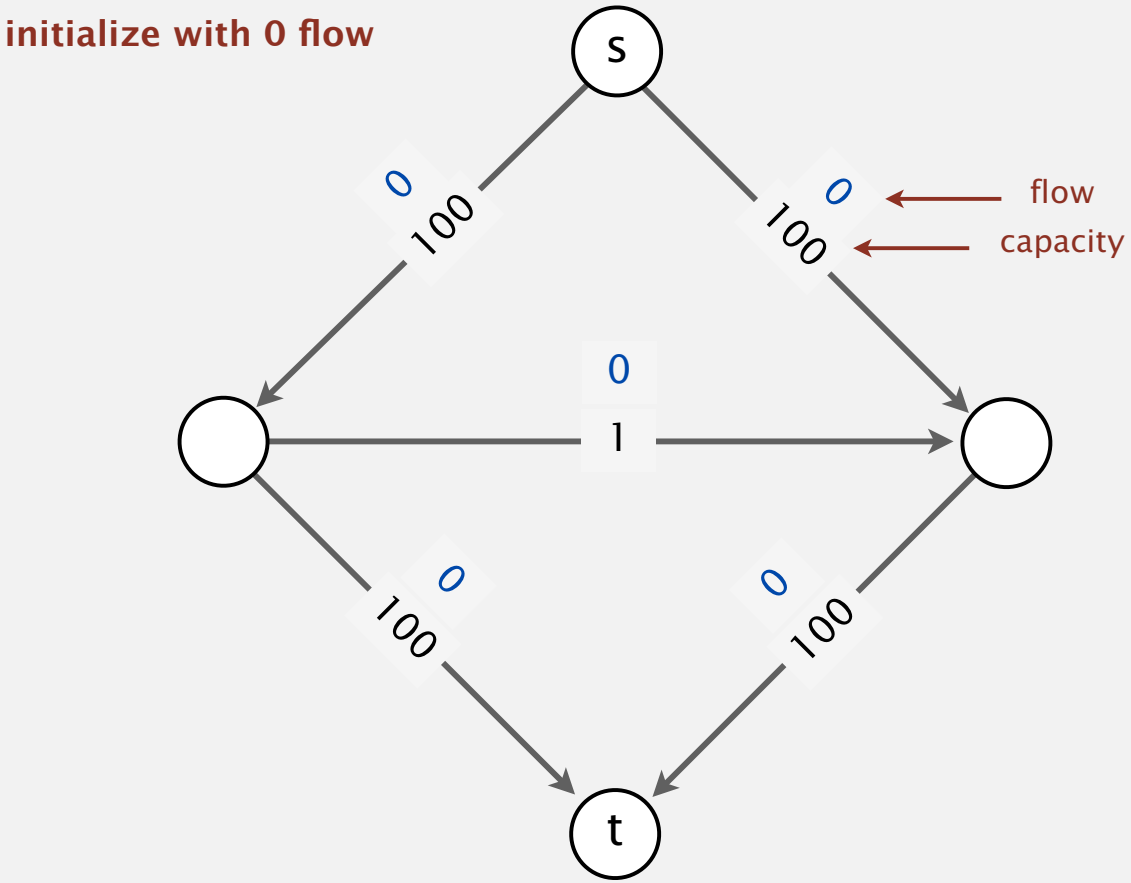
critical for some applications (stay tuned)

Integrality theorem. There exists an integer-valued maxflow.

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

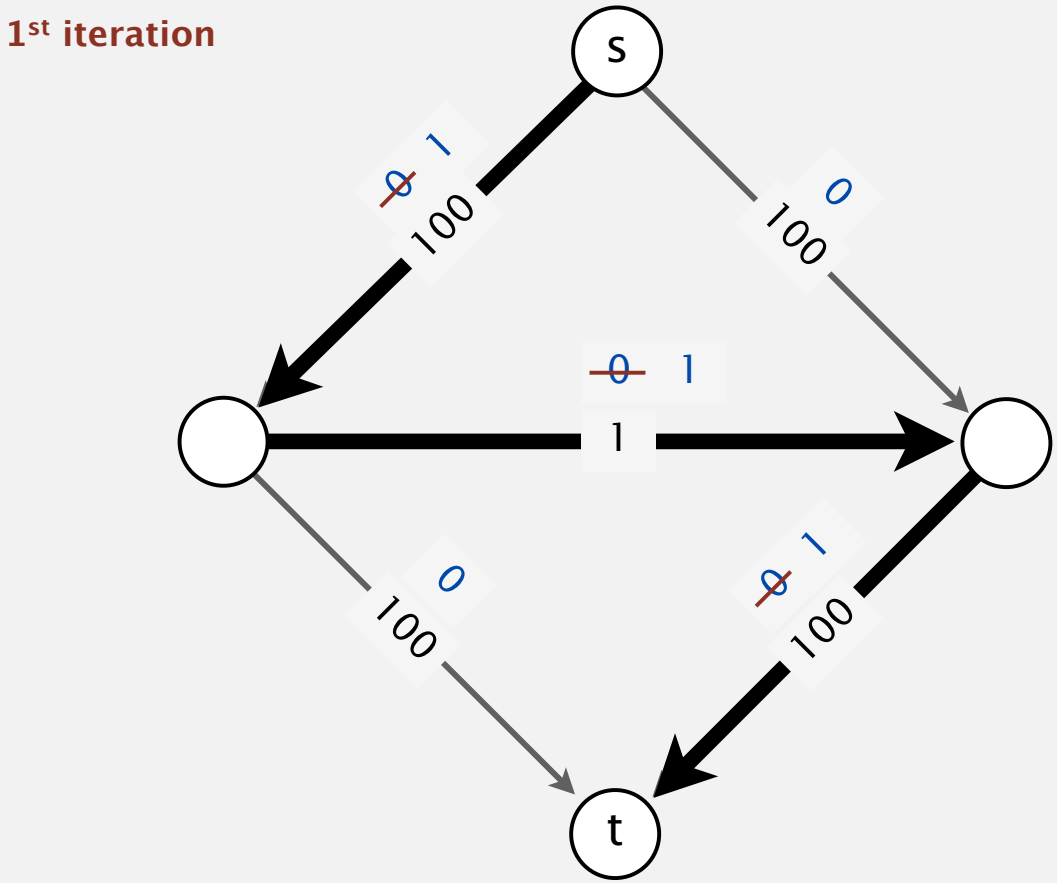
Bad case for Ford-Fulkerson

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



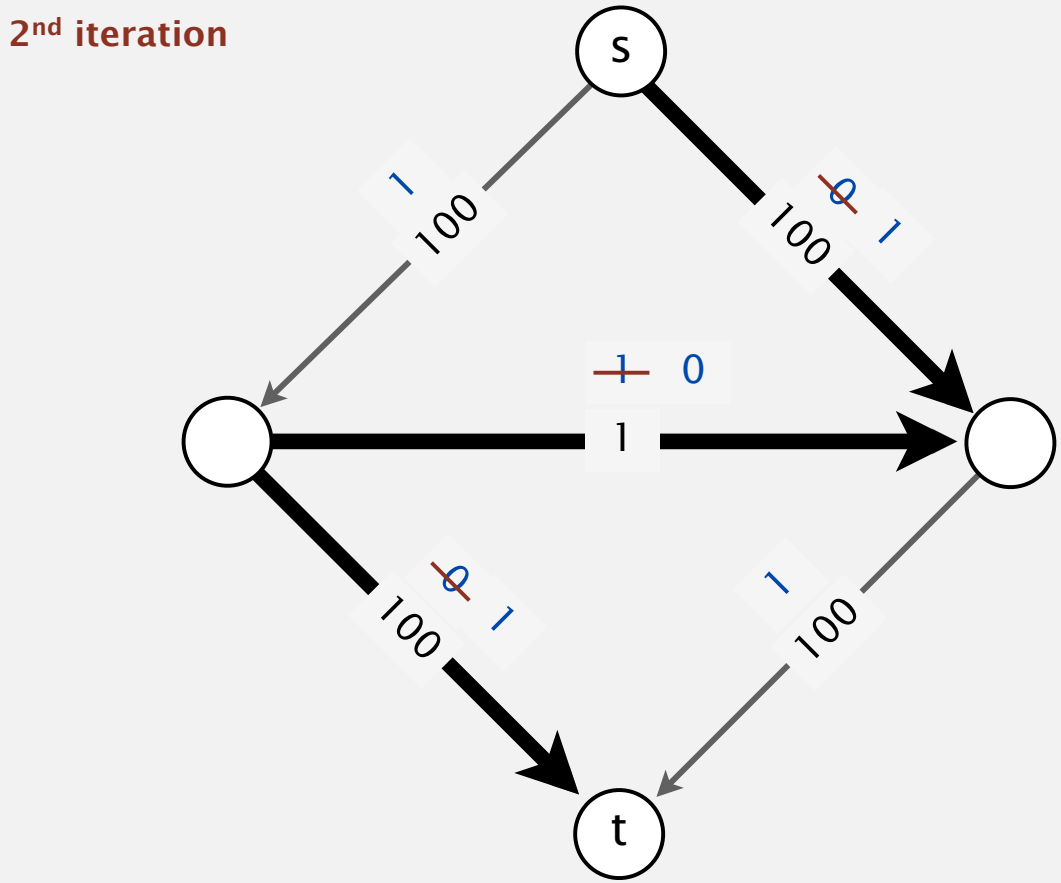
Bad case for Ford-Fulkerson

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



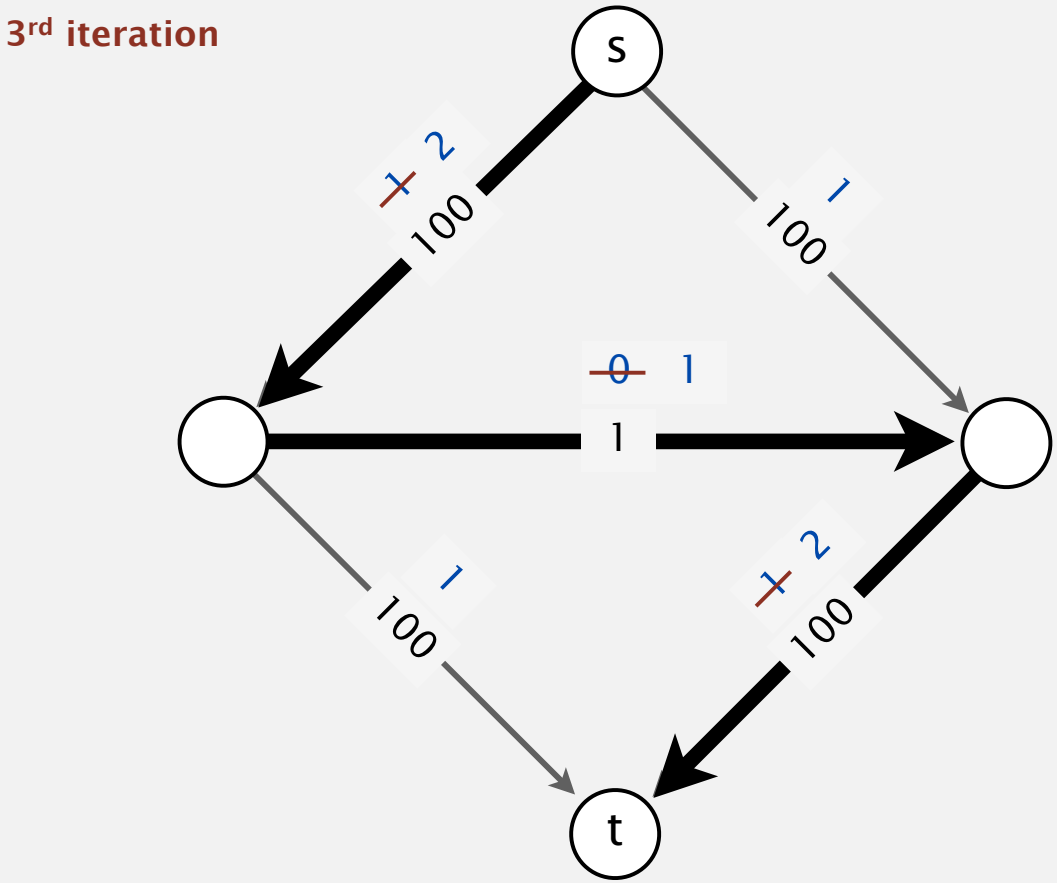
Bad case for Ford-Fulkerson

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



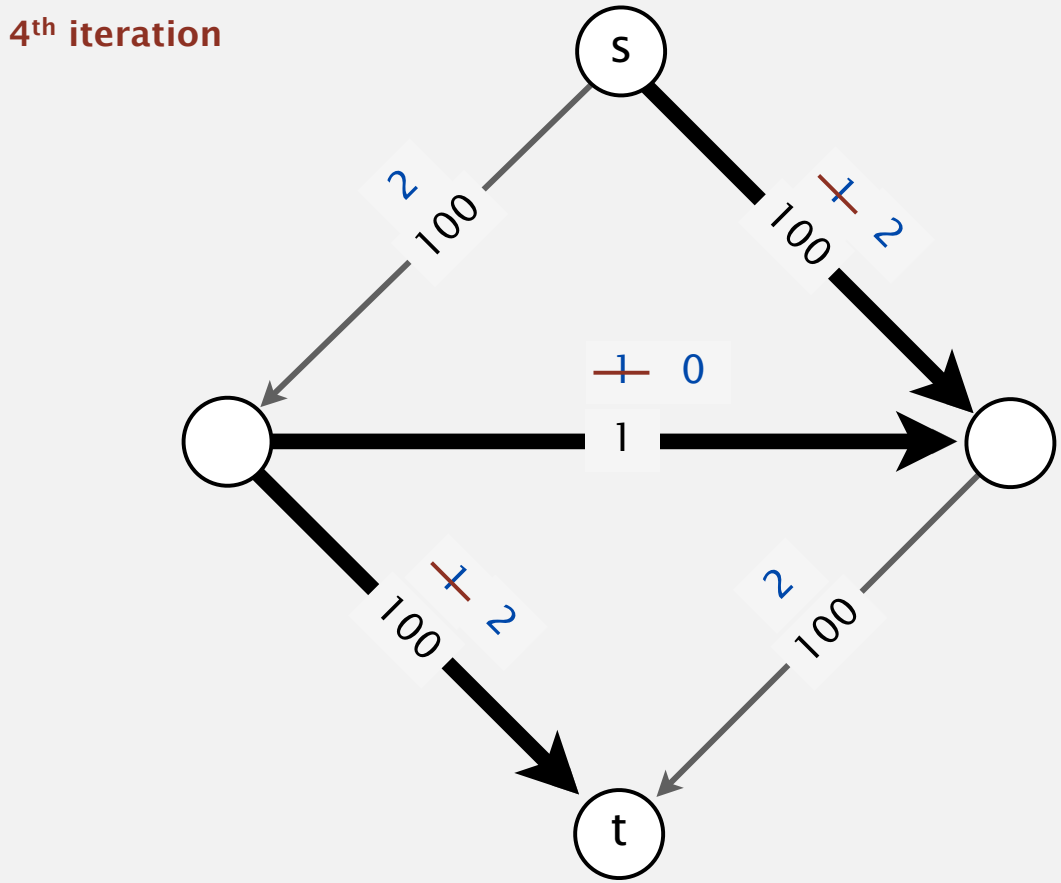
Bad case for Ford-Fulkerson

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.

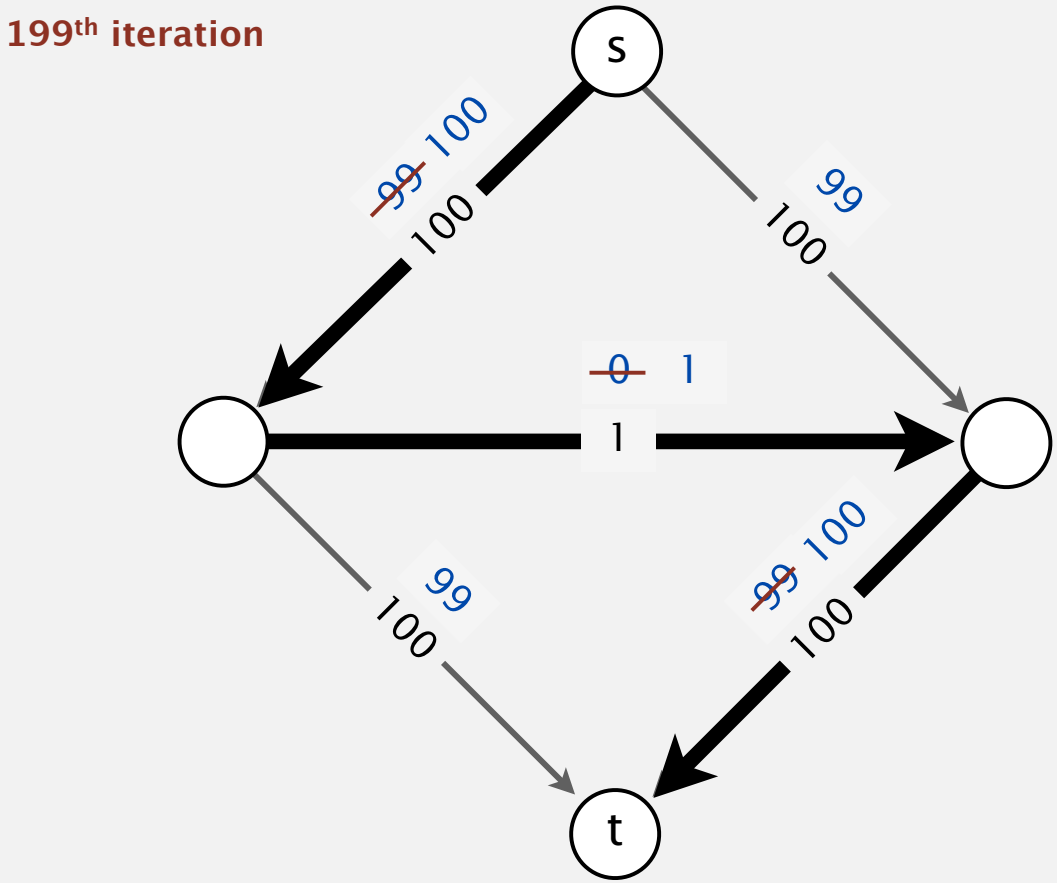


Bad case for Ford-Fulkerson

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

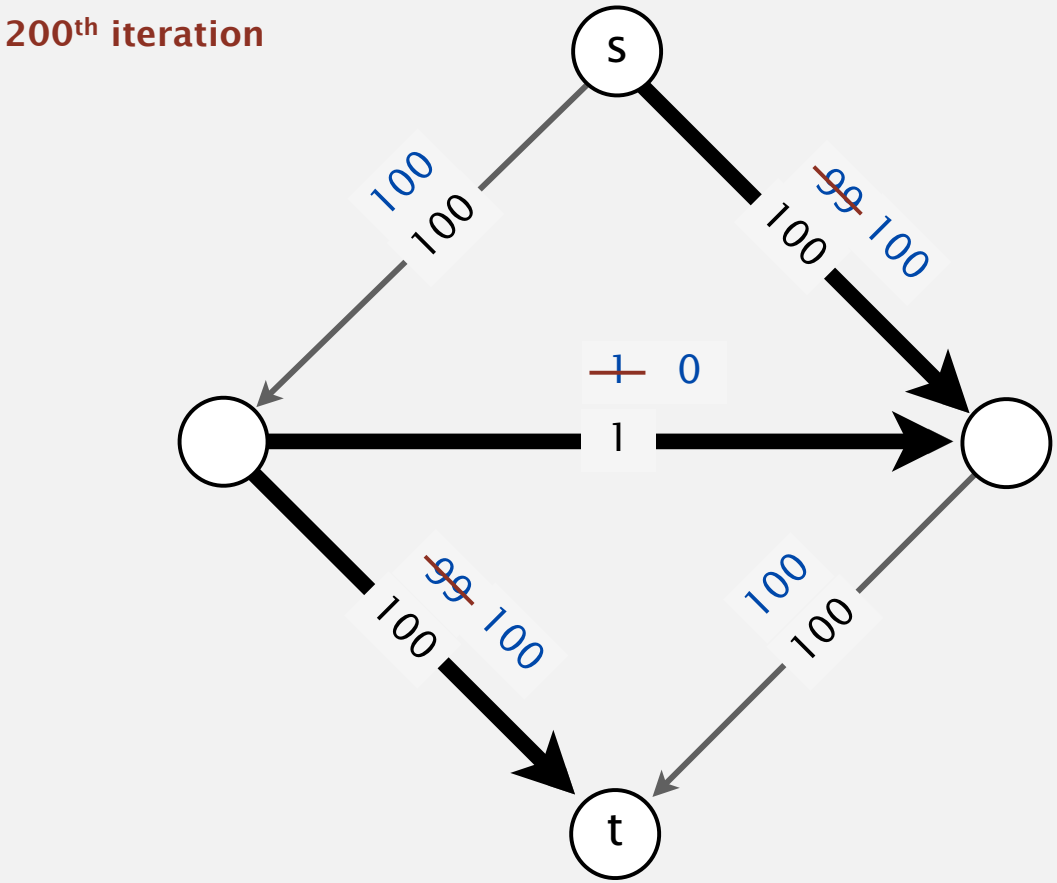
Bad case for Ford-Fulkerson

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.

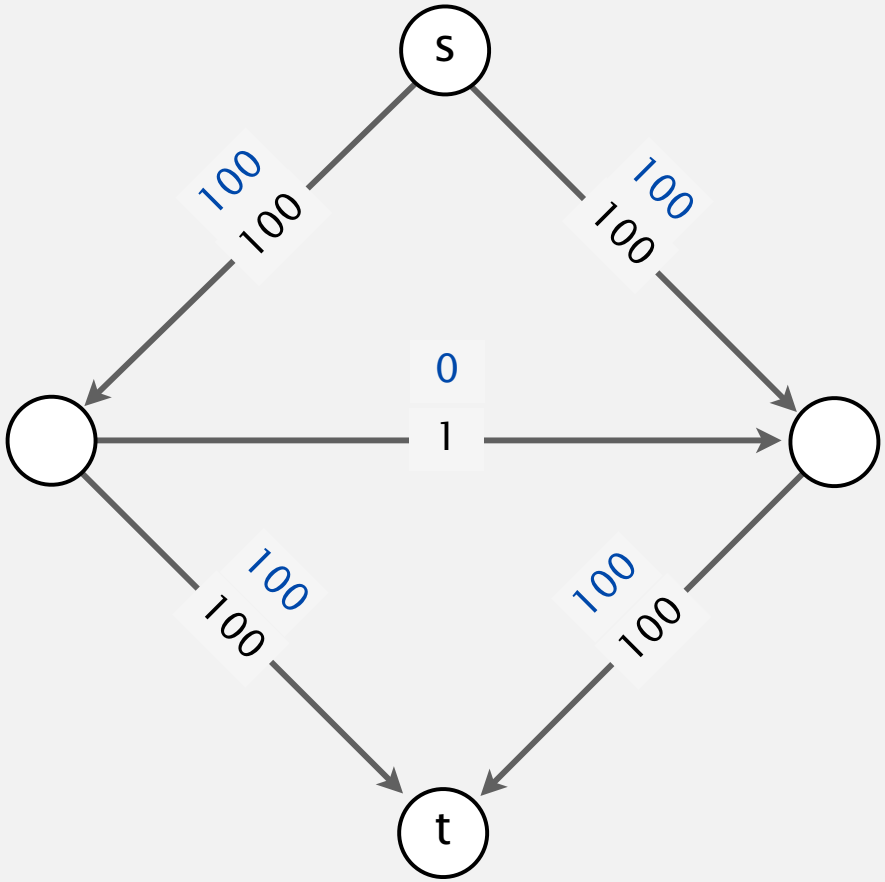


Bad case for Ford-Fulkerson

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.

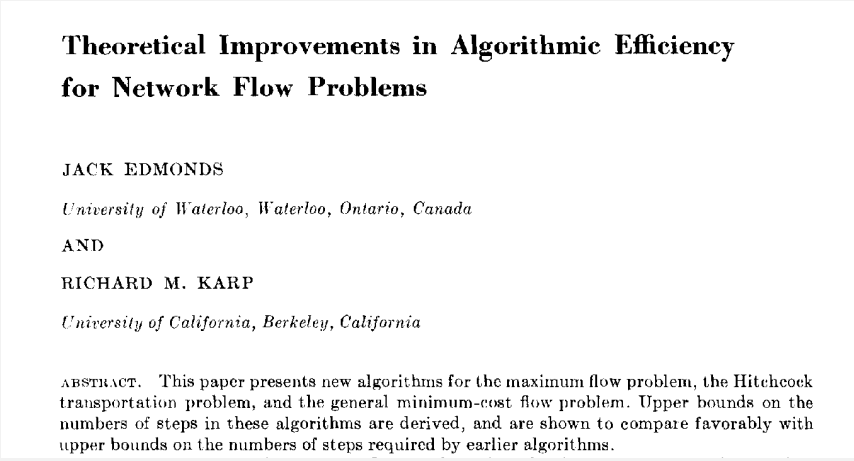
augmenting path	number of paths	implementation
random path	$\leq E\,U$	randomized queue
DFS path	$\leq E\,U$	stack (DFS)
shortest path	$\leq \frac{1}{2}\,E\,V$	queue (BFS)
fattest path	$\leq E \ln(E\,U)$	priority queue

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

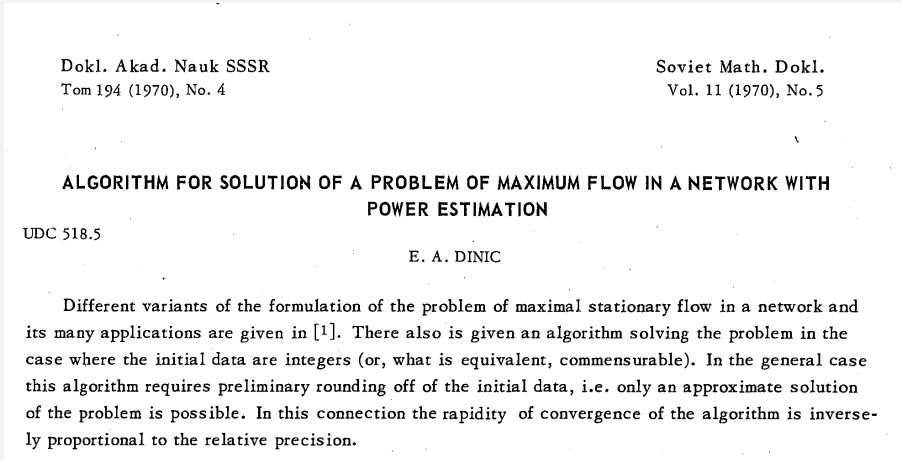
AND

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)

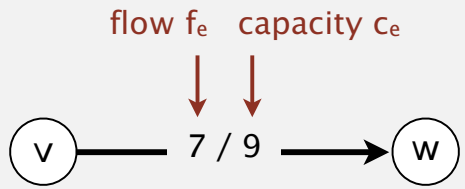
http://algs4.cs.princeton.edu

6.4 Maximum Flow

- *introduction*
- *Ford-Fulkerson algorithm*
- *maxflow-mincut theorem*
- *analysis of running time*
- *Java implementation*
- *applications*

Flow network representation

Flow edge data type. Associate flow f_e and capacity c_e with edge $e = v \rightarrow w$.



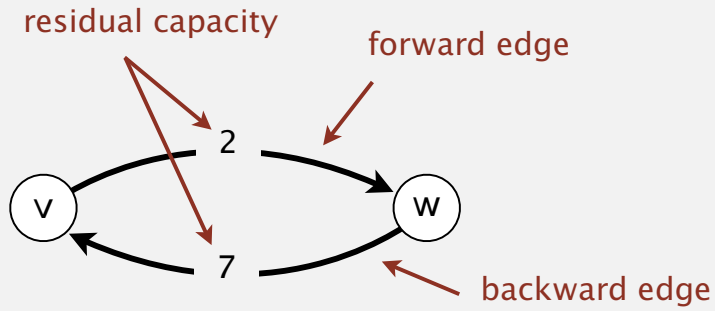
Flow network data type. Must be able to process edge $e = v \rightarrow w$ in either direction: include e in adjacency lists of both v and w.

Residual (spare) capacity.

- Forward edge: residual capacity $= c_e - f_e$.
- Backward edge: residual capacity $= f_e$.

Augment flow.

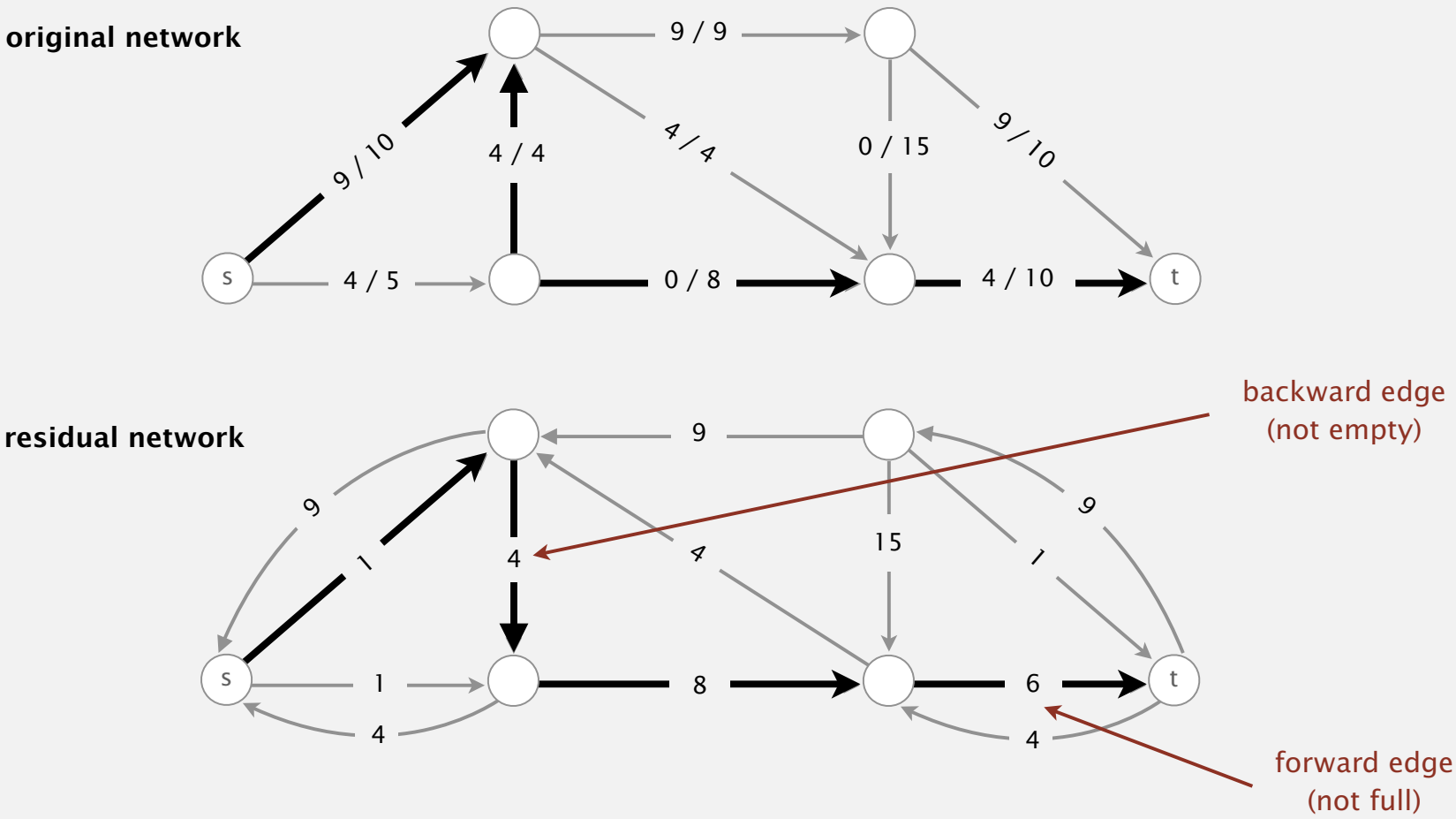
- Forward edge: add Δ.
- Backward edge: subtract Δ.



Flow network representation

Residual network. A useful view of a flow network.

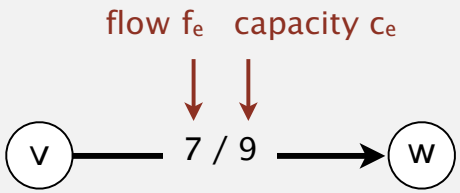
includes all edges with positive residual capacity

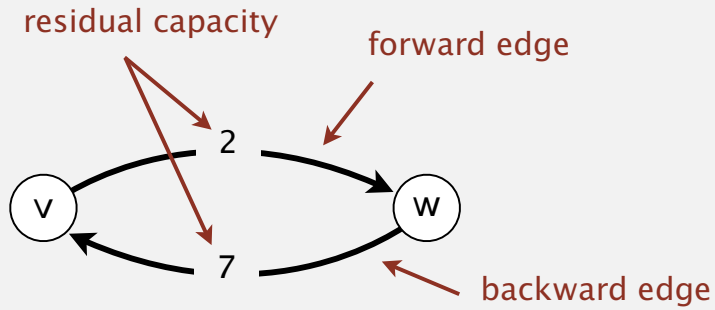


Key point. Augmenting paths in original network are in 1-1 correspondence with directed paths in residual network.

Flow edge API

public class FlowEdge	
FlowEdge(int v, int w, double capacity)	*create a flow edge v→w*
int from()	*vertex this edge points from*
int to()	*vertex this edge points to*
int other(int v)	*other endpoint*
double capacity()	*capacity of this edge*
double flow()	*flow in this edge*
double residualCapacityTo(int v)	*residual capacity toward v*
void addResidualFlowTo(int v, double delta)	*add delta flow toward v*





Flow edge: Java implementation

```
public class FlowEdge
{
   private final int v, w;           // from and to
   private final double capacity;    // capacity
   private double flow;              // flow

   public FlowEdge(int v, int w, double capacity)
   {
      this.v        = v;
      this.w        = w;
      this.capacity = capacity;
   }

   public int from()         { return v;        }
   public int to()           { return w;        }
   public double capacity()  { return capacity; }
   public double flow()      { return flow;     }

   public int other(int vertex)
   {
      if      (vertex == v) return w;
      else if (vertex == w) return v;
      else throw new IllegalArgumentException();
   }

   public double residualCapacityTo(int vertex)              {...}
   public void addResidualFlowTo(int vertex, double delta)   {...}
}
```

flow variable (mutable)

next slide

Flow edge: Java implementation (continued)

```
public double residualCapacityTo(int vertex)
{
   if      (vertex == v) return flow;
   else if (vertex == w) return capacity - flow;
   else throw new IllegalArgumentException();
}
```

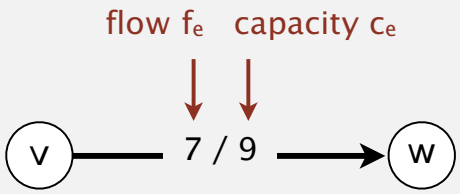
forward edge

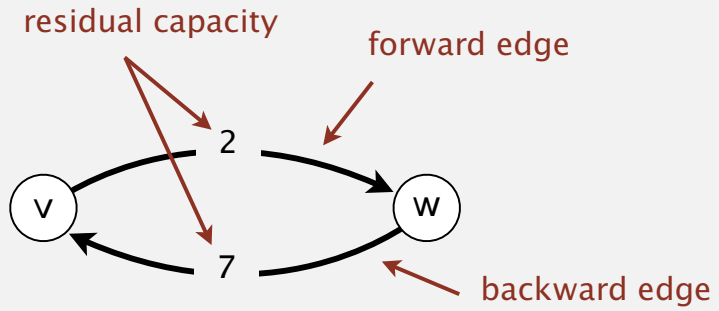
backward edge

```
public void addResidualFlowTo(int vertex, double delta)
{
   if      (vertex == v) flow -= delta;
   else if (vertex == w) flow += delta;
   else throw new IllegalArgumentException();
}
```

forward edge

backward edge





Flow network API

public class FlowNetwork		
	FlowNetwork(int V)	*create an empty flow network with V vertices*
	FlowNetwork(In in)	*construct flow network input stream*
void	addEdge(FlowEdge e)	*add flow edge e to this flow network*
Iterable<FlowEdge>	adj(int v)	*forward and backward edges incident to v*
Iterable<FlowEdge>	edges()	*all edges in this flow network*
int	V()	*number of vertices*
int	E()	*number of edges*
String	toString()	*string representation*

Conventions. Allow self-loops and parallel edges.

Flow network: Java implementation

```
public class FlowNetwork
{
    private final int V;
    private Bag<FlowEdge>[] adj;

    public FlowNetwork(int V)
    {
       this.V = V;
       adj = (Bag<FlowEdge>[]) new Bag[V];
       for (int v = 0; v < V; v++)
          adj[v] = new Bag<FlowEdge>();
    }

    public void addEdge(FlowEdge e)
    {
       int v = e.from();
       int w = e.to();
       adj[v].add(e);
       adj[w].add(e);
    }

    public Iterable<FlowEdge> adj(int v)
    {  return adj[v]; }
}
```

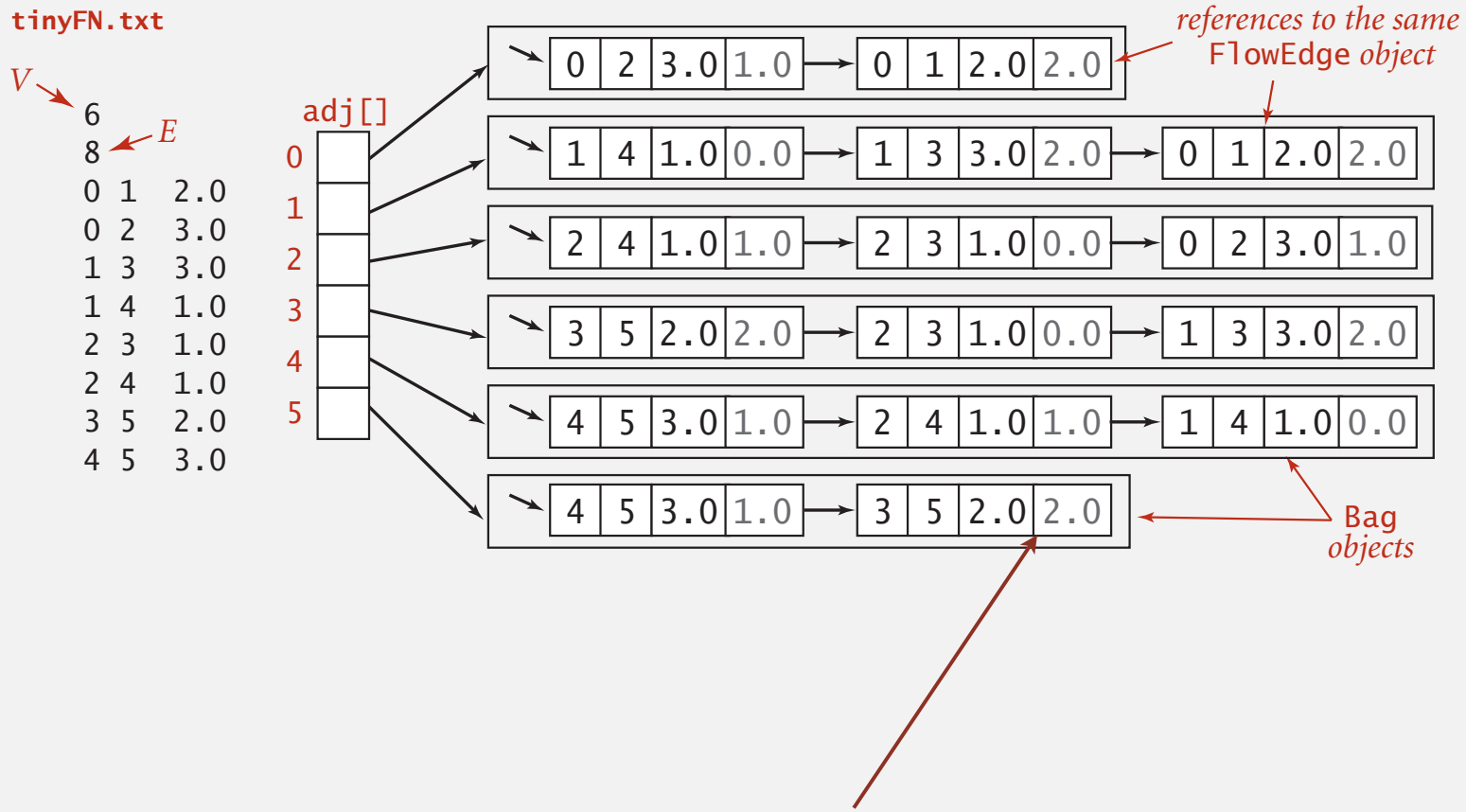
same as EdgeWeightedGraph, but adjacency lists of FlowEdges instead of Edges

add forward edge

add backward edge

Flow network: adjacency-lists representation

Maintain vertex-indexed array of `FlowEdge` lists (use `Bag` abstraction).



Note. Adjacency list includes edges with 0 residual capacity.

(residual network is represented implicitly)

Finding a shortest augmenting path (cf. breadth-first search)

```
private boolean hasAugmentingPath(FlowNetwork G, int s, int t)
{
    edgeTo = new FlowEdge[G.V()];
    marked = new boolean[G.V()];

    Queue<Integer> queue = new Queue<Integer>();
    queue.enqueue(s);
    marked[s] = true;
    while (!queue.isEmpty())
    {
        int v = queue.dequeue();

        for (FlowEdge e : G.adj(v))
        {
            int w = e.other(v);
            if (!marked[w] && (e.residualCapacityTo(w) > 0) )
            {
               edgeTo[w] = e;
               marked[w] = true;
               queue.enqueue(w);
            }
        }
    }

    return marked[t];
}
```

found path from s to w in the residual network?

save last edge on path to w;
mark w;
add w to the queue

is t reachable from s in residual network?

Ford-Fulkerson: Java implementation

```
public class FordFulkerson
{
   private boolean[] marked;   // true if s->v path in residual network
   private FlowEdge[] edgeTo;  // last edge on s->v path
   private double value;       // value of flow

   public FordFulkerson(FlowNetwork G, int s, int t)
   {
      value = 0.0;
      while (hasAugmentingPath(G, s, t))
      {
         double bottle = Double.POSITIVE_INFINITY;
         for (int v = t; v != s; v = edgeTo[v].other(v))
            bottle = Math.min(bottle, edgeTo[v].residualCapacityTo(v));

         for (int v = t; v != s; v = edgeTo[v].other(v))
            edgeTo[v].addResidualFlowTo(v, bottle);

         value += bottle;
      }
   }

   private boolean hasAugmentingPath(FlowNetwork G, int s, int t)
   {  /* See previous slide. */  }

   public double value()
   {  return value;  }

   public boolean inCut(int v)
   {  return marked[v];  }
}
```

compute edgeTo[]

compute bottleneck capacity

augment flow

is v reachable from s in residual network?

http://algs4.cs.princeton.edu

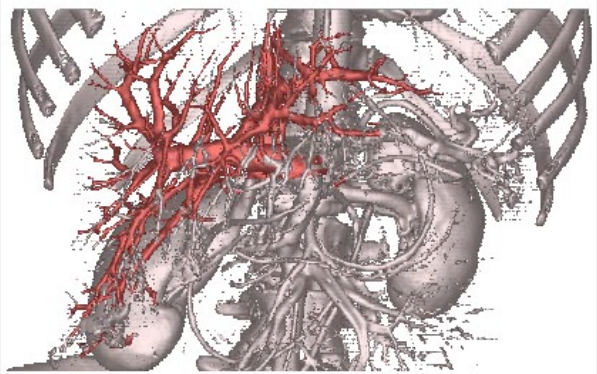
6.4 Maximum Flow

- *introduction*
- *Ford-Fulkerson algorithm*
- *maxflow-mincut theorem*
- *analysis of running time*
- *Java implementation*
- *applications*

Maxflow and mincut applications

Maxflow/mincut is a widely applicable problem-solving model.

- Data mining.
- Open-pit mining.
- Bipartite matching.
- Network reliability.
- Baseball elimination.
- Image segmentation.
- Network connectivity.
- Distributed computing.
- Security of statistical data.
- Egalitarian stable matching.
- Multi-camera scene reconstruction.
- Sensor placement for homeland security.
- Many, many, more.



liver and hepatic vascularization segmentation

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
	Facebook	8	Facebook
	Google		Carol
4	Dave	9	Google
	Amazon		Alice
	Yahoo		Carol
5	Eliza	10	Yahoo
	Amazon		Dave
	Yahoo		Eliza

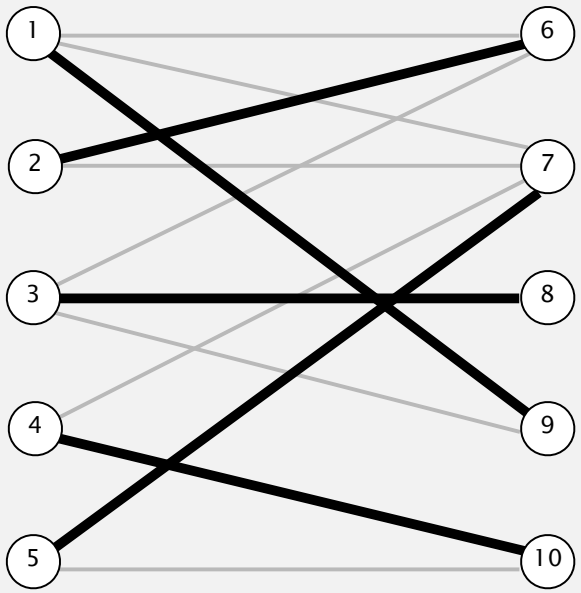
Bipartite matching problem

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

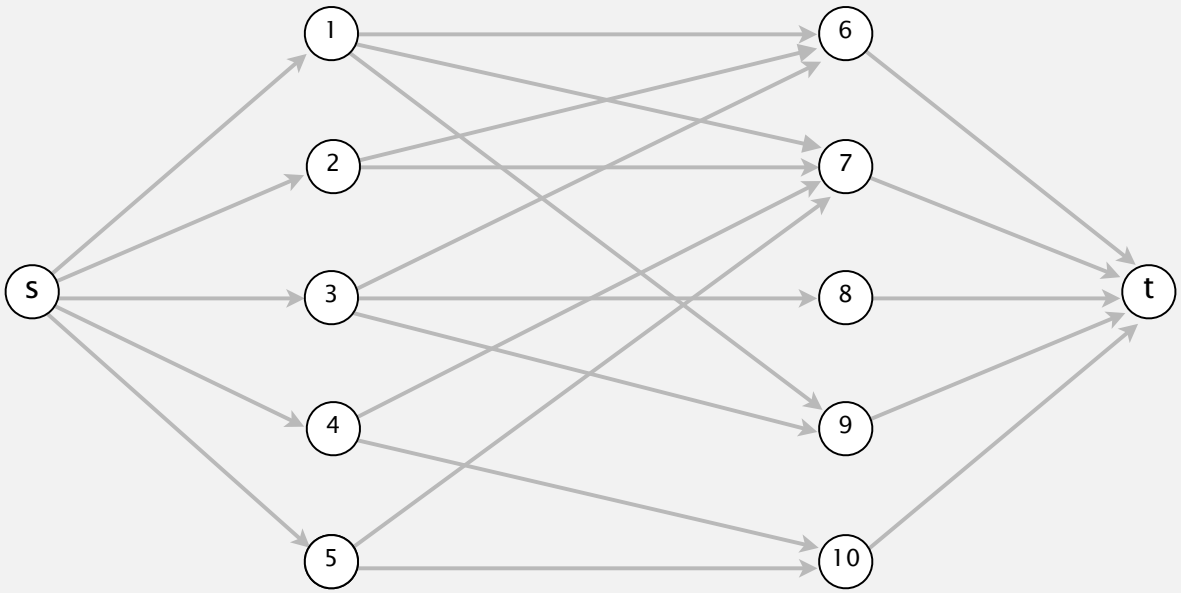
1 Alice
Adobe
Amazon
Google
2 Bob
Adobe
Amazon
3 Carol
Adobe
Facebook
Google
4 Dave
Amazon
Yahoo
5 Eliza
Amazon
Yahoo

6 Adobe
Alice
Bob
Carol
7 Amazon
Alice
Bob
Dave
Eliza
8 Facebook
Carol
9 Google
Alice
Carol
10 Yahoo
Dave
Eliza

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



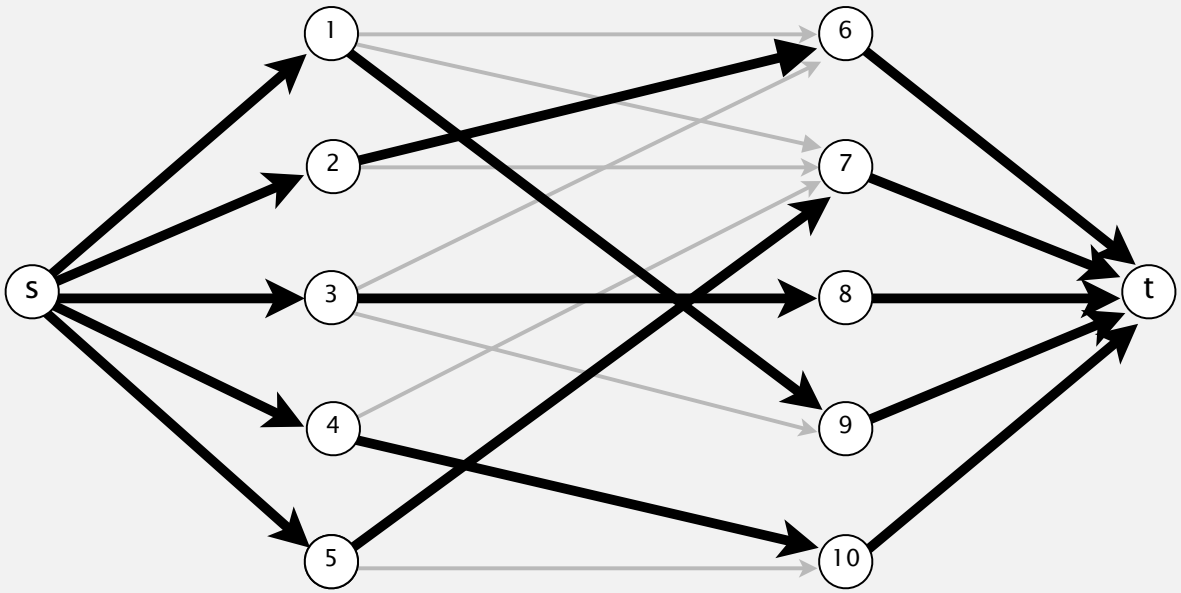
bipartite matching problem

1	Alice	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.

flow network

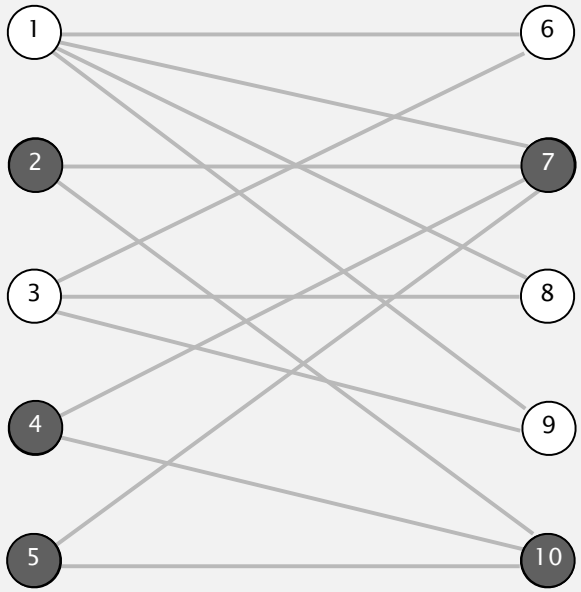


bipartite matching problem

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

What the mincut tells us

Goal. When no perfect matching, explain why.



no perfect matching exists

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

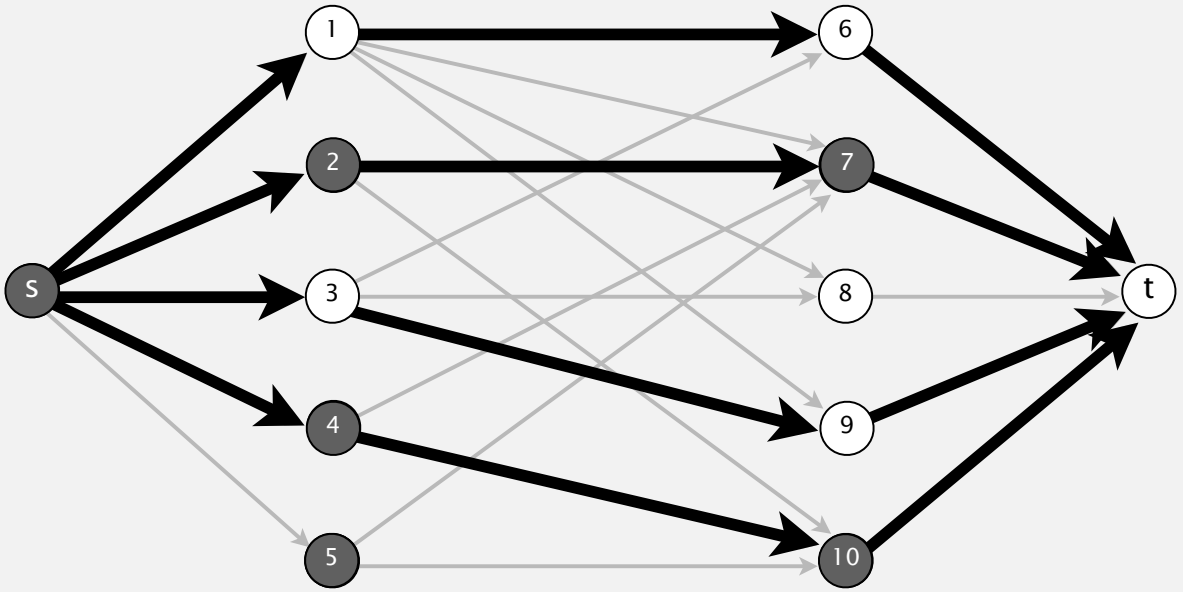
student in S
can be matched
only to
companies in T

| 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.



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

student in S can be matched only to companies in T

| S | > | T |

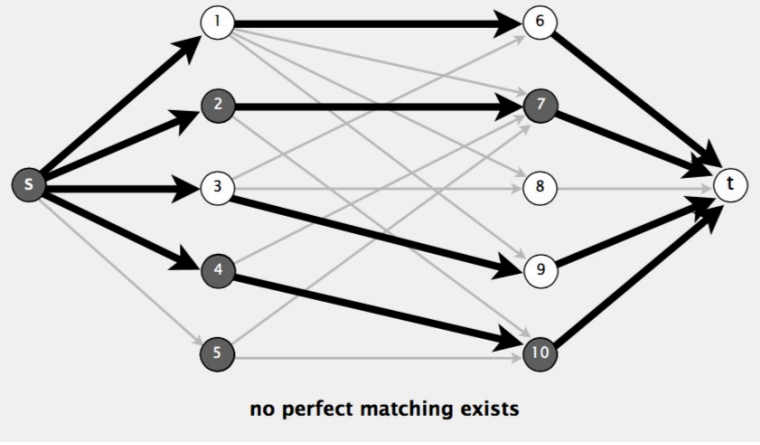
no perfect matching exists

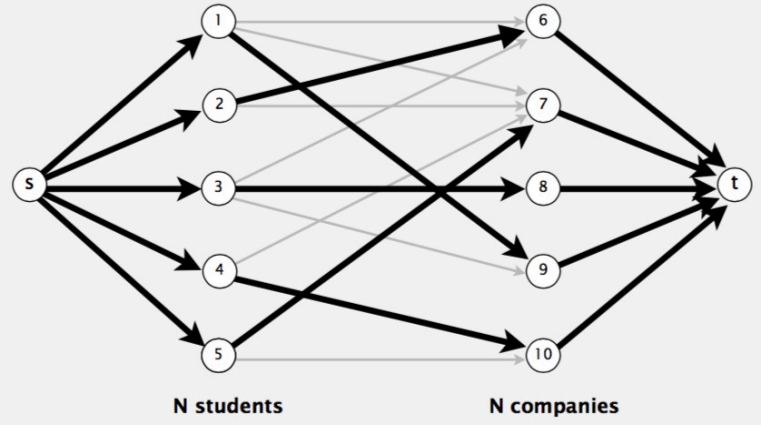
Bottom line. When no perfect matching, mincut explains why.

WHAT THE FLOW TELLS US

→ If there is a flow k then there is a matching with k edges.
  → 0 or 1 units in and out of every internal vertex
  → source s supplies k units ⇒ all students get a job
  → target t consumes k units ⇒ all jobs get a student

→ If there is a matching with k edges then there is a flow k.





Bipartite matching running time

Theorem. The Ford-Fulkerson algorithm solves the bipartite matching problem in $O(m\,n)$ time.

Theorem. [Hopcroft-Karp 1973] The bipartite matching problem can be solved in $O(m\,n^{1/2})$ time.

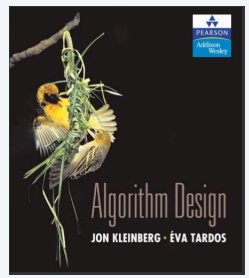
SIAM J. COMPUT.
Vol. 2, No. 4, December 1973

AN $n^{5/2}$ ALGORITHM FOR MAXIMUM MATCHINGS IN BIPARTITE GRAPHS*

JOHN E. HOPCROFT† AND RICHARD M. KARP‡

Abstract. The present paper shows how to construct a maximum matching in a bipartite graph with n vertices and m edges in a number of computation steps proportional to $(m + n)\sqrt{n}$.

Key words. algorithm, algorithmic analysis, bipartite graphs, computational complexity, graphs, matching



Baseball elimination problem

Q. Which teams have a chance of finishing the season with the most wins?

i	team		wins	losses	to play	ATL	PHI	NYM	MON
0		Atlanta	83	71	8	–	1	6	1
1		Philly	80	79	3	1	–	0	2
2		New York	78	78	6	6	0	–	0
3		Montreal	77	82	3	1	2	0	–

Montreal is mathematically eliminated.

- Montreal finishes with ≤ 80 wins.
- Atlanta already has 83 wins.

Baseball elimination problem

Q. Which teams have a chance of finishing the season with the most wins?

i	team		wins	losses	to play	ATL	PHI	NYM	MON
0		Atlanta	83	71	8	–	1	6	1
1		Philly	80	79	3	1	–	0	2
2		New York	78	78	6	6	0	–	0
3		Montreal	77	82	3	1	2	0	–

Philadelphia is mathematically eliminated.

- Philadelphia finishes with ≤ 83 wins.
- Either New York or Atlanta will finish with ≥ 84 wins.

Observation. Answer depends not only on how many games already won and left to play, but on whom they're against.

Baseball elimination problem

Q. Which teams have a chance of finishing the season with the most wins?

i	team	wins	losses	to play	NYY	BAL	BOS	TOR	DET
0	New York	75	59	28	–	3	8	7	3
1	Baltimore	71	63	28	3	–	2	7	4
2	Boston	69	66	27	8	2	–	0	0
3	Toronto	63	72	27	7	7	0	–	0
4	Detroit	49	86	27	3	4	0	0	–

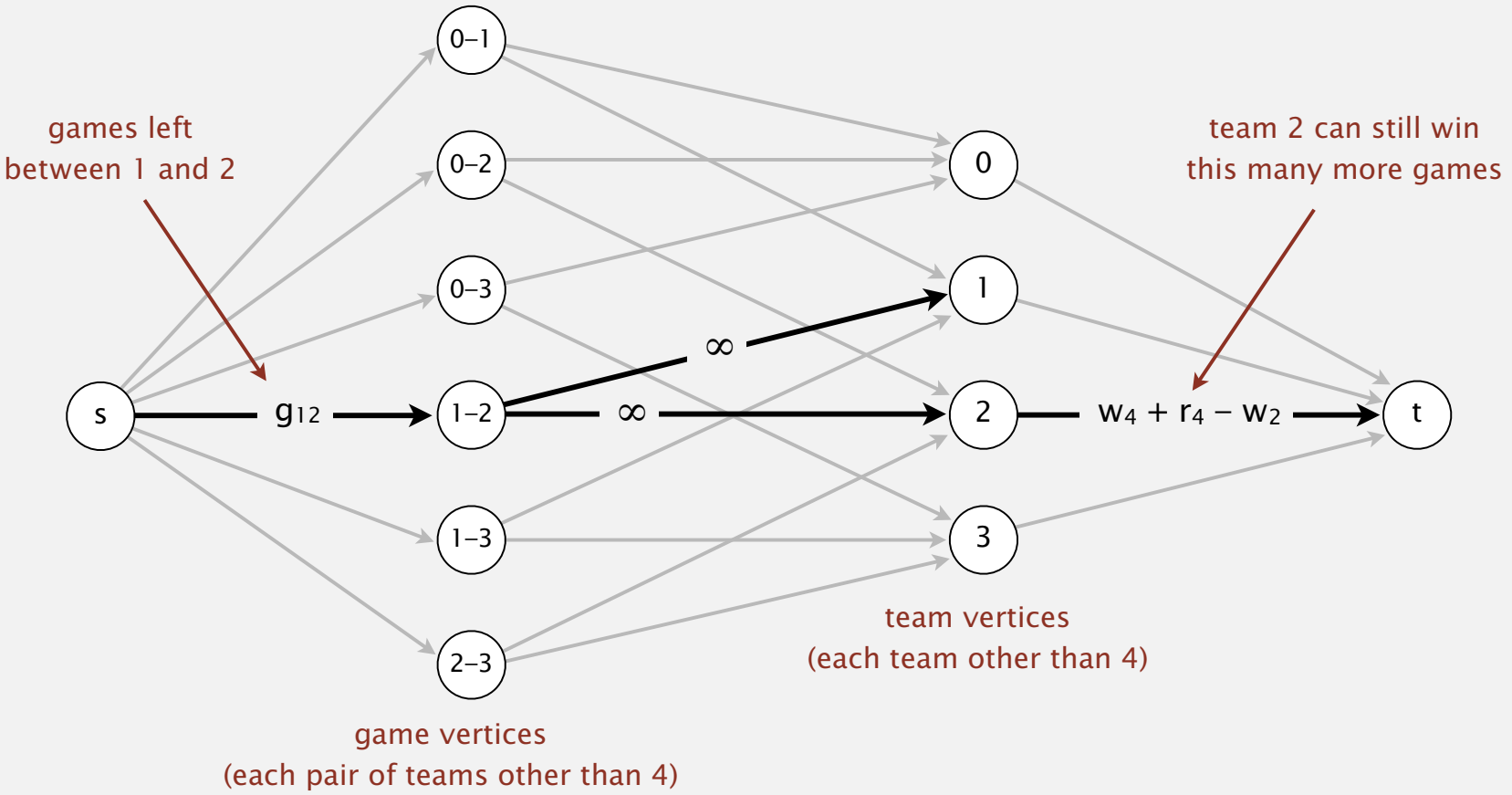
AL East (August 30, 1996)

Detroit is mathematically eliminated.

- Detroit finishes with ≤ 76 wins.
- Wins for $R = \{ \text{NYY, BAL, BOS, TOR} \} = 278$.
- Remaining games among $\{ \text{NYY, BAL, BOS, TOR} \} = 3 + 8 + 7 + 2 + 7 = 27$.
- Average team in R wins $305/4 = 76.25$ games.

Baseball elimination problem: maxflow formulation

Intuition. Remaining games flow from s to t.



Fact. Team 4 not eliminated iff all edges pointing from s are full in maxflow.

Maximum flow algorithms: theory

(Yet another) holy grail for theoretical computer scientists.

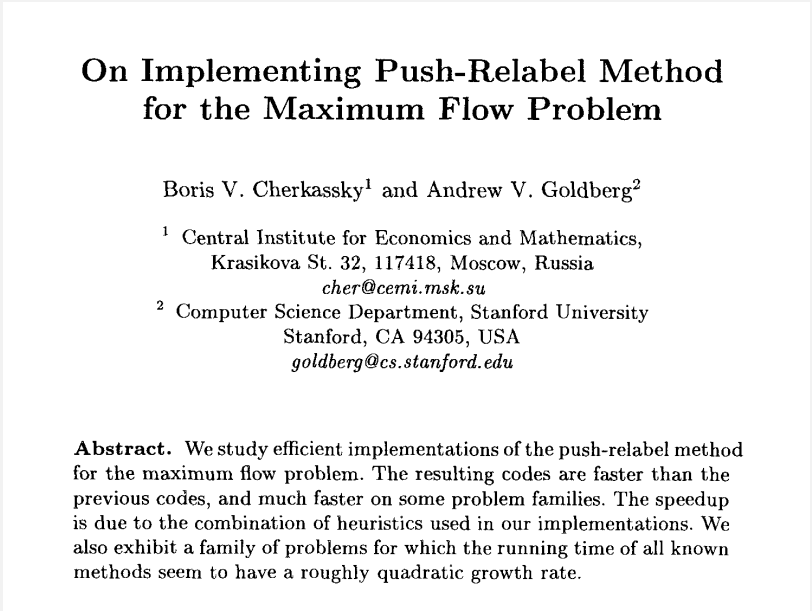
year	method	worst case	discovered by
1951	**simplex**	$E^3 U$	Dantzig
1955	**augmenting path**	$E^2 U$	Ford-Fulkerson
1970	**shortest augmenting path**	E^3	Dinitz, Edmonds-Karp
1970	**fattest augmenting path**	$E^2 \log E \log(E U)$	Dinitz, Edmonds-Karp
1977	**blocking flow**	$E^{5/2}$	Cherkasky
1978	**blocking flow**	$E^{7/3}$	Galil
1983	**dynamic trees**	$E^2 \log E$	Sleator-Tarjan
1985	**capacity scaling**	$E^2 \log U$	Gabow
1997	**length function**	$E^{3/2} \log E \log U$	Goldberg-Rao
2012	**compact network**	$E^2 / \log E$	Orlin
?	**?**	E	?

maxflow algorithms for sparse digraphs with E edges, integer capacities between 1 and U

Maximum flow algorithms: practice

Warning. Worst-case order-of-growth is generally not useful for predicting or comparing maxflow algorithm performance in practice.

Best in practice. Push-relabel method with gap relabeling: $E^{3/2}$.

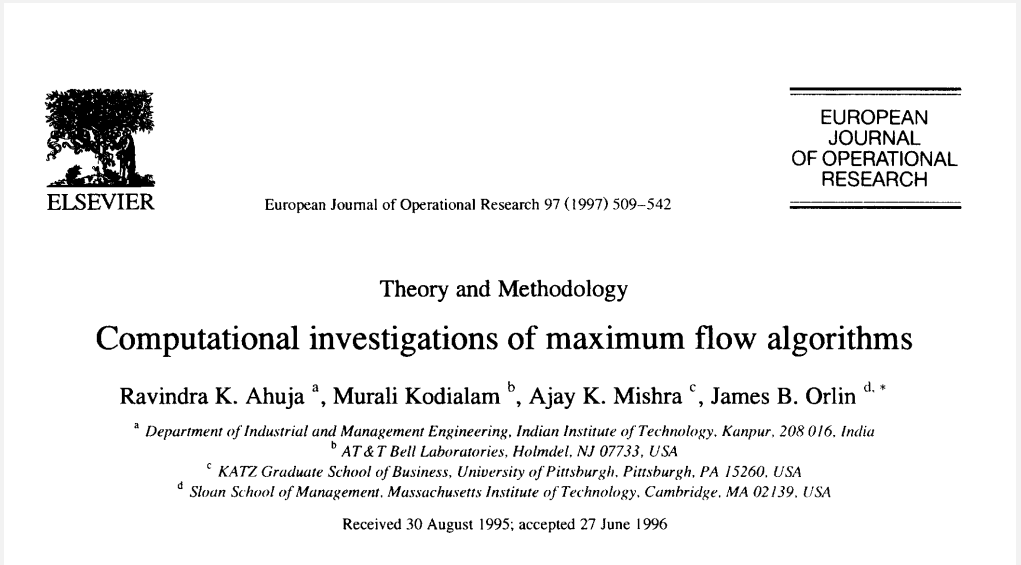


On Implementing Push-Relabel Method for the Maximum Flow Problem

Boris V. Cherkassky[1] and Andrew V. Goldberg[2]

[1] Central Institute for Economics and Mathematics, Krasikova St. 32, 117418, Moscow, Russia
cher@cemi.msk.su
[2] Computer Science Department, Stanford University Stanford, CA 94305, USA
goldberg@cs.stanford.edu

Abstract. We study efficient implementations of the push-relabel method for the maximum flow problem. The resulting codes are faster than the previous codes, and much faster on some problem families. The speedup is due to the combination of heuristics used in our implementations. We also exhibit a family of problems for which the running time of all known methods seem to have a roughly quadratic growth rate.



ELSEVIER

European Journal of Operational Research 97 (1997) 509–542

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

Theory and Methodology

Computational investigations of maximum flow algorithms

Ravindra K. Ahuja [a], Murali Kodialam [b], Ajay K. Mishra [c], James B. Orlin [d,*]

[a] *Department of Industrial and Management Engineering, Indian Institute of Technology, Kanpur, 208 016, India*
[b] *AT & T Bell Laboratories, Holmdel, NJ 07733, USA*
[c] *KATZ Graduate School of Business, University of Pittsburgh, Pittsburgh, PA 15260, USA*
[d] *Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA 02139, USA*

Received 30 August 1995; accepted 27 June 1996

Summary

Mincut problem. Find an st-cut of minimum capacity.

Maxflow problem. Find an st-flow of maximum value.

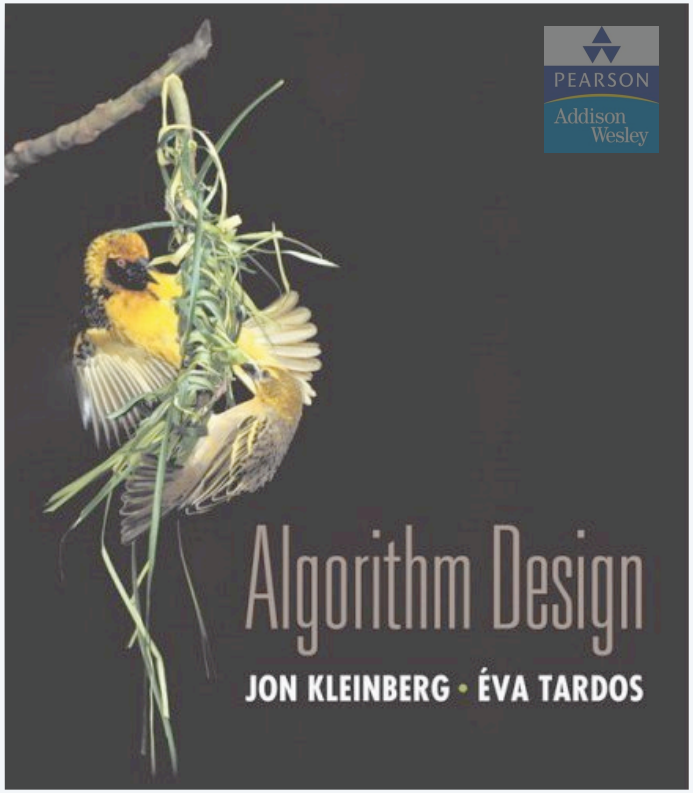
Duality. Value of the maxflow = capacity of mincut.

Proven successful approaches.

- Ford-Fulkerson (various augmenting-path strategies).
- Preflow-push (various versions).

Open research challenges.

- Practice: solve real-world maxflow/mincut problems in linear time.
- Theory: prove it for worst-case inputs.
- Still much to be learned!



SECTION

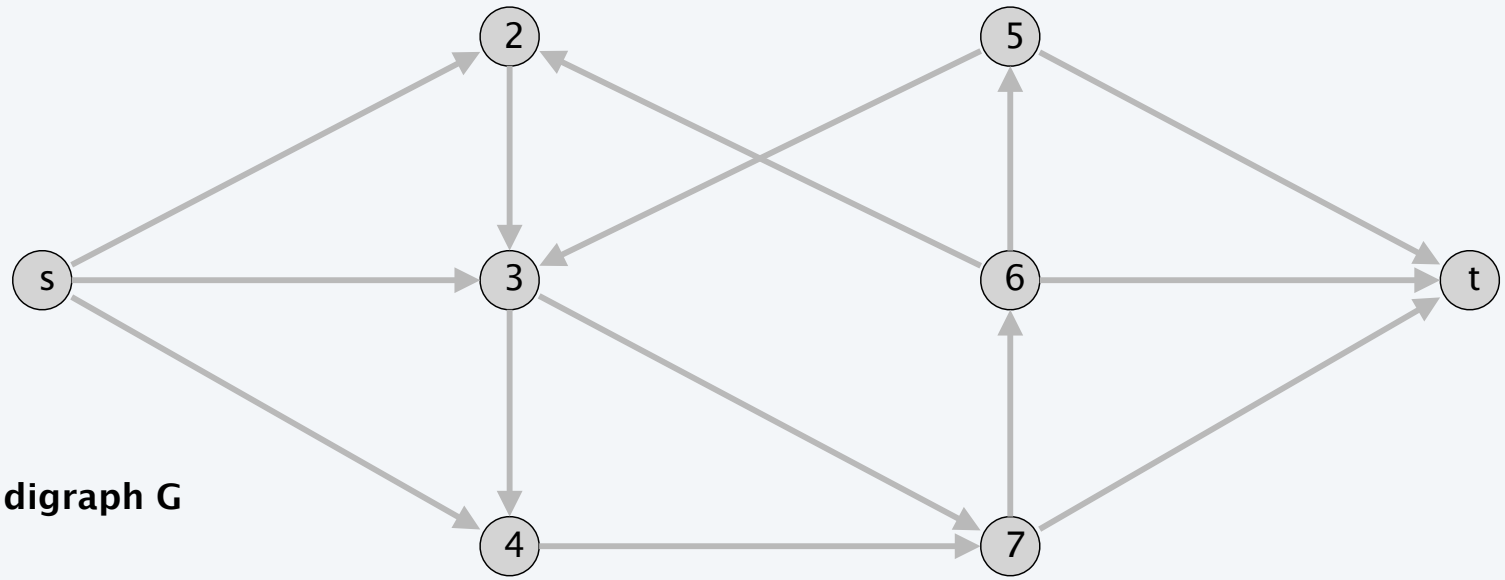
7. NETWORK FLOW II

- *bipartite matching*
- *disjoint paths*
- *extensions to max flow*
- *survey design*
- *airline scheduling*
- *image segmentation*
- *project selection*
- *baseball elimination*

Edge-disjoint paths

Def. Two paths are edge-disjoint if they have no edge in common.

Disjoint path problem. Given a digraph $G = (V, E)$ and two nodes s and t, find the max number of edge-disjoint $s \rightarrow t$ paths.



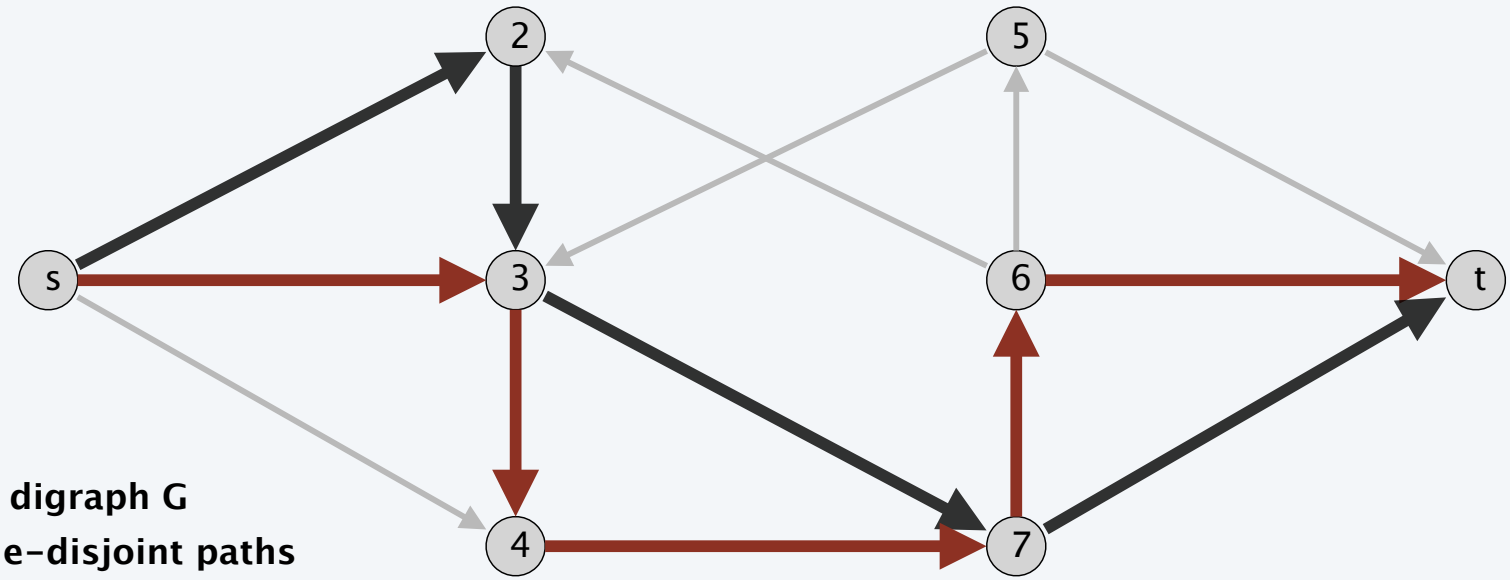
digraph G

Edge-disjoint paths

Def. Two paths are edge-disjoint if they have no edge in common.

Disjoint path problem. Given a digraph $G = (V, E)$ and two nodes s and t, find the max number of edge-disjoint $s \rightarrow t$ paths.

Ex. Communication networks.



digraph G
2 edge-disjoint paths

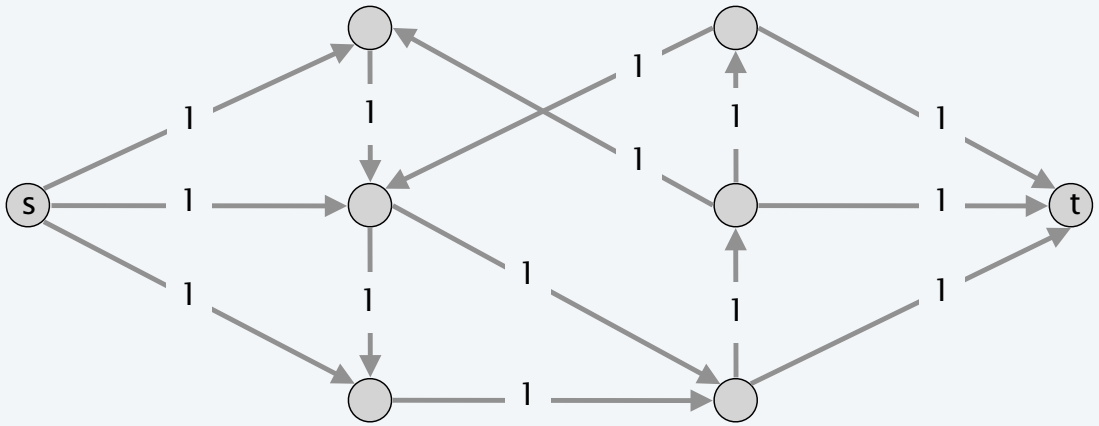
Edge-disjoint paths

Max flow formulation. Assign unit capacity to every edge.

Theorem. Max number edge-disjoint $s \rightarrow t$ paths equals value of max flow.

Pf. $\leq$

- Suppose there are k edge-disjoint $s \rightarrow t$ paths $P_1, \ldots, P_k$.
- Set $f(e) = 1$ if e participates in some path P_j; else set $f(e) = 0$.
- Since paths are edge-disjoint, f is a flow of value k. ▪



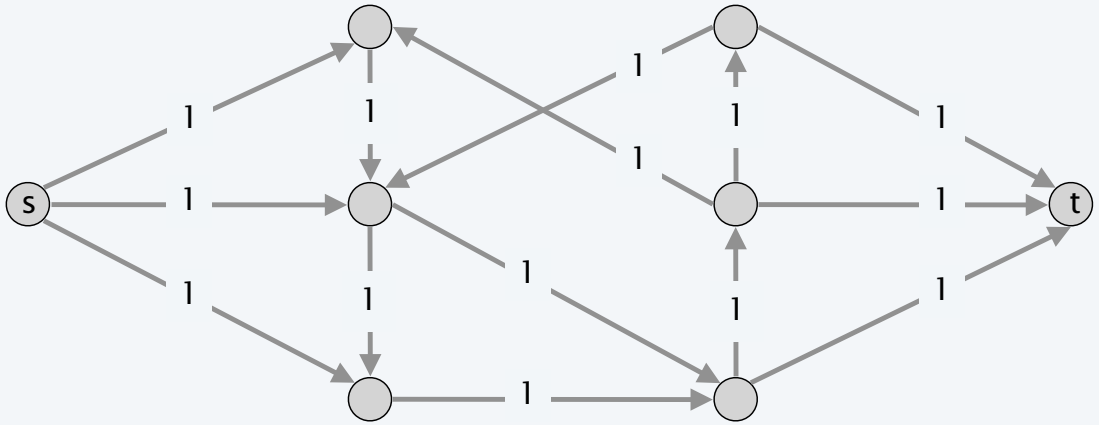
Edge-disjoint paths

Max flow formulation. Assign unit capacity to every edge.

Theorem. Max number edge-disjoint $s \rightarrow t$ paths equals value of max flow.

Pf. ≥

- Suppose max flow value is k.
- Integrality theorem $\Rightarrow$ there exists 0-1 flow f of value k.
- Consider edge (s, u) with $f(s, u) = 1$.
  - by conservation, there exists an edge (u, v) with $f(u, v) = 1$
  - continue until reach t, always choosing a new edge
- Produces k (not necessarily simple) edge-disjoint paths. ▪

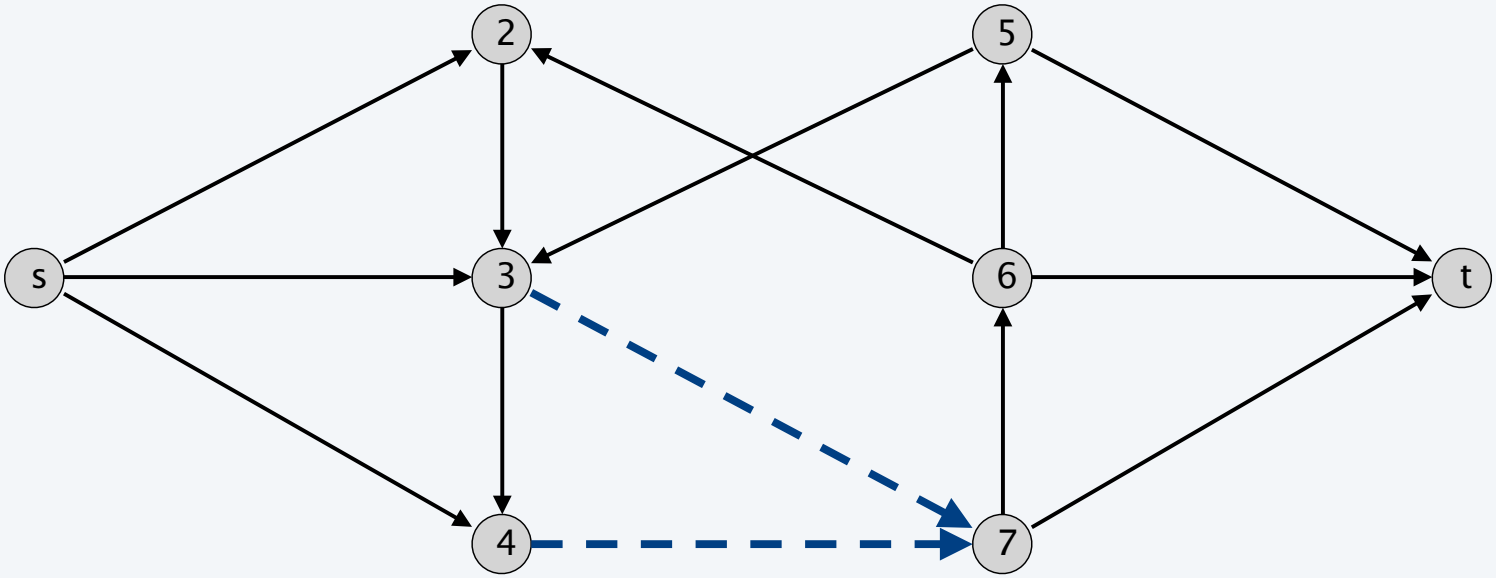


can eliminate cycles to get simple paths in O(mn) time if desired (flow decomposition)

Network connectivity

Def. A set of edges $F \subseteq E$ disconnects t from s if every $s \rightarrow t$ path uses at least one edge in F.

Network connectivity. Given a digraph $G = (V, E)$ and two nodes s and t, find min number of edges whose removal disconnects t from s.

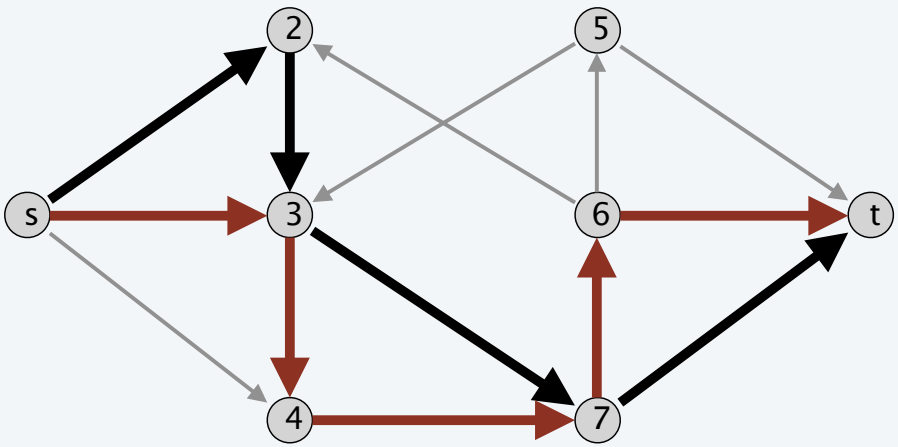


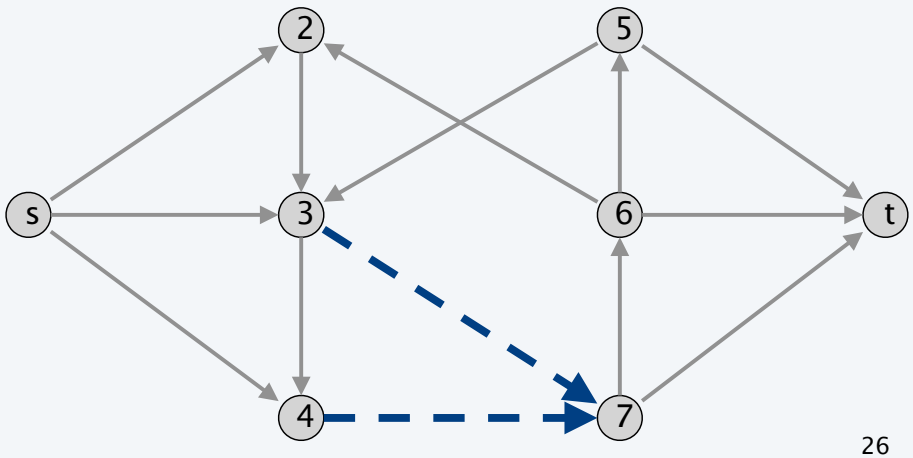
Menger's theorem

Theorem. [Menger 1927] The max number of edge-disjoint $s \rightarrow t$ paths is equal to the min number of edges whose removal disconnects t from s.

Pf. $\leq$

- Suppose the removal of $F \subseteq E$ disconnects t from s, and $|F| = k$.
- Every $s \rightarrow t$ path uses at least one edge in F.
- Hence, the number of edge-disjoint paths is $\leq k$. ▪



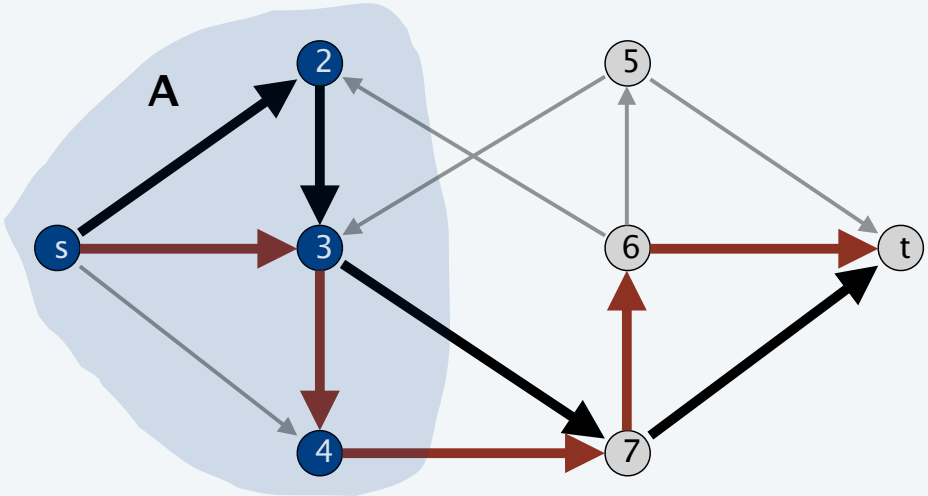


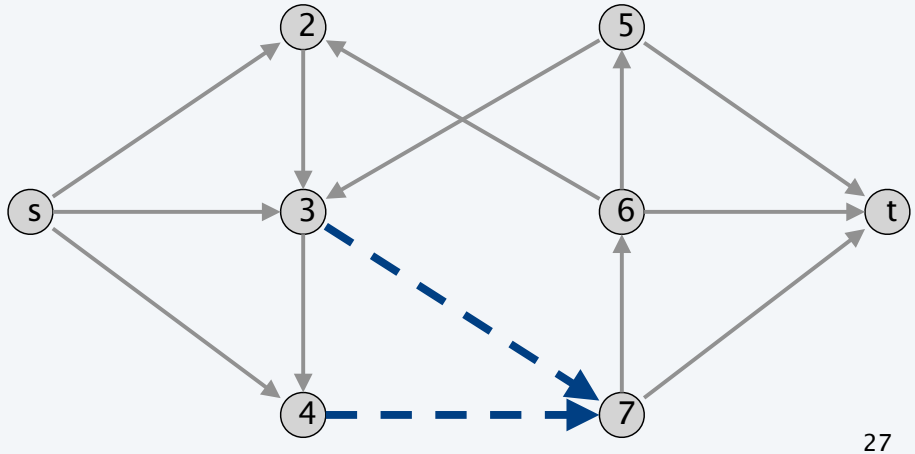
Menger's theorem

Theorem. [Menger 1927] The max number of edge-disjoint $s \rightarrow t$ paths equals the min number of edges whose removal disconnects t from s.

Pf. $\geq$

- Suppose max number of edge-disjoint paths is k.
- Then value of max flow $= k$.
- Max-flow min-cut theorem $\Rightarrow$ there exists a cut (A, B) of capacity k.
- Let F be set of edges going from A to B.
- $|F| = k$ and disconnects t from s. ▪

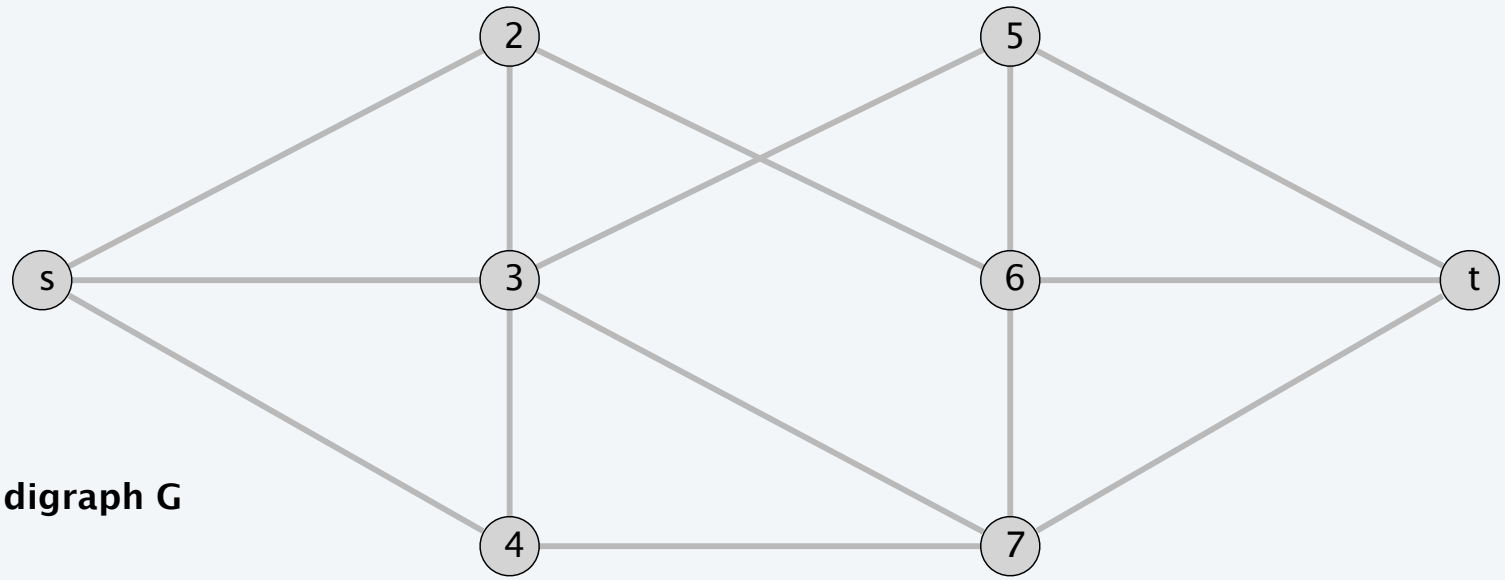




Edge-disjoint paths in undirected graphs

Def. Two paths are edge-disjoint if they have no edge in common.

Disjoint path problem in undirected graphs. Given a graph $G = (V, E)$ and two nodes s and t, find the max number of edge-disjoint s-t paths.

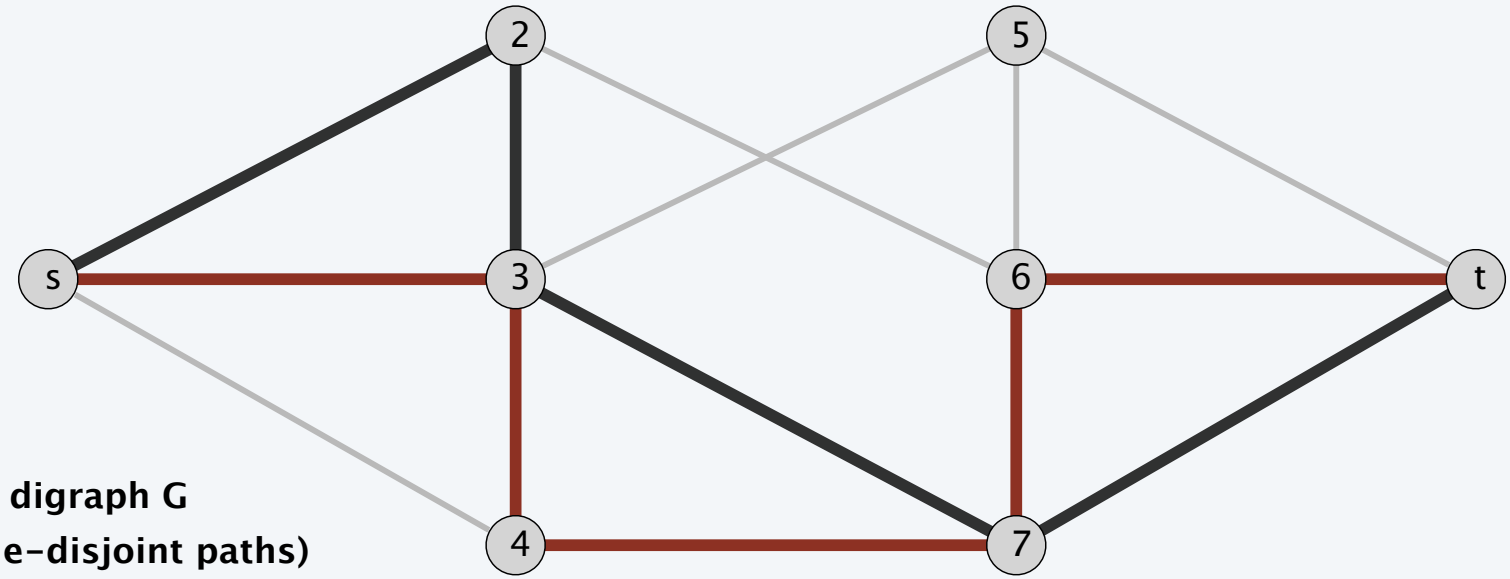


digraph G

Edge-disjoint paths in undirected graphs

Def. Two paths are edge-disjoint if they have no edge in common.

Disjoint path problem in undirected graphs. Given a graph $G = (V, E)$ and two nodes s and t, find the max number of edge-disjoint s-t paths.

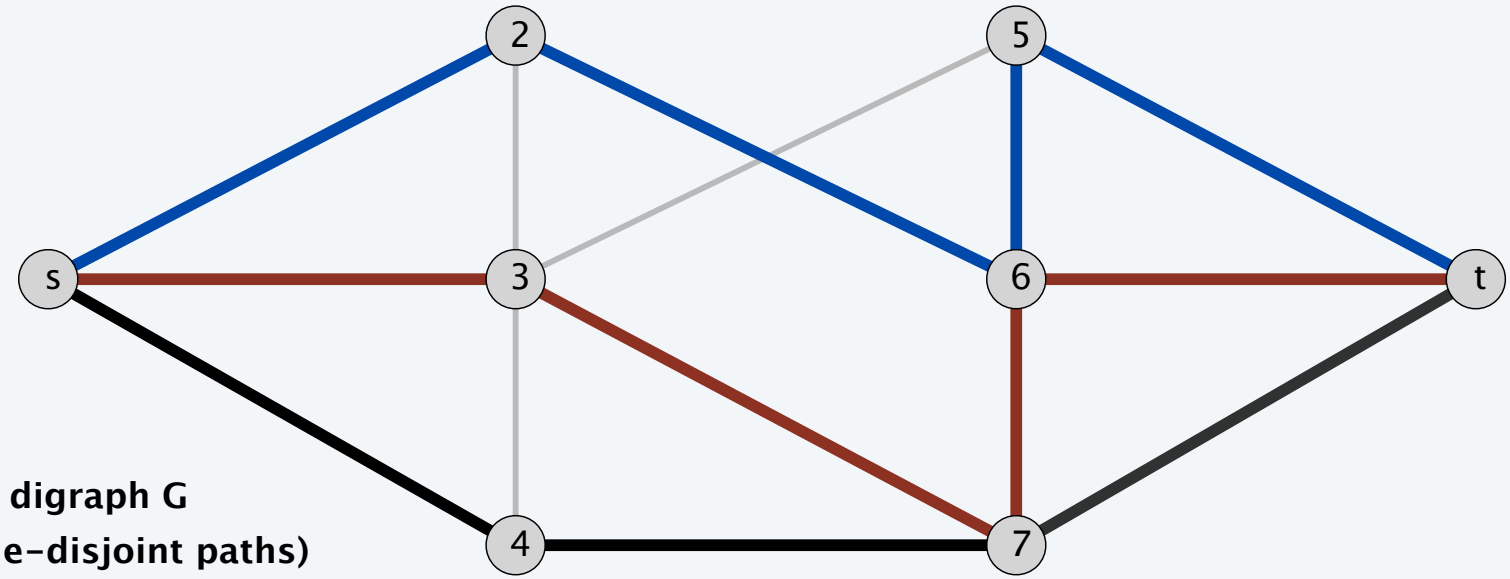


digraph G
(2 edge-disjoint paths)

Edge-disjoint paths in undirected graphs

Def. Two paths are edge-disjoint if they have no edge in common.

Disjoint path problem in undirected graphs. Given a graph $G = (V, E)$ and two nodes s and t, find the max number of edge-disjoint s-t paths.



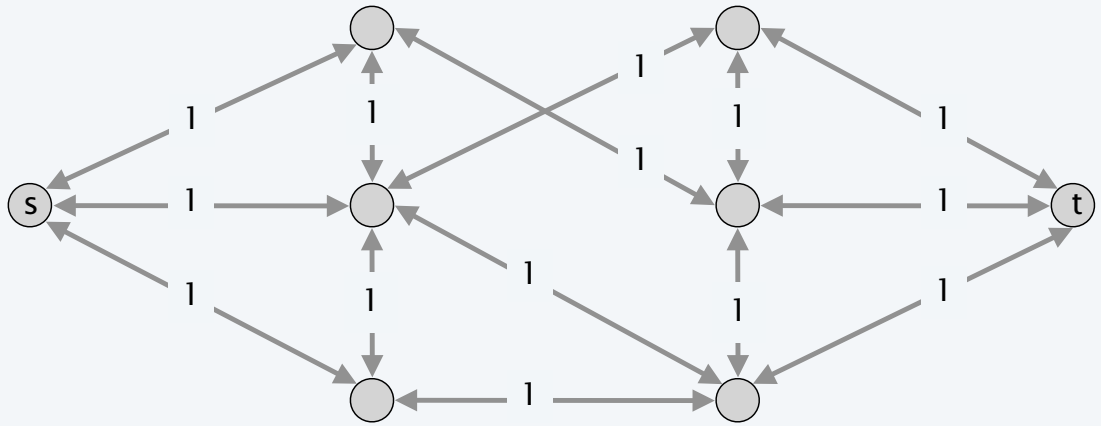
digraph G
(3 edge-disjoint paths)

Edge-disjoint paths in undirected graphs

Max flow formulation. Replace edge edge with two antiparallel edges and assign unit capacity to every edge.

Observation. Two paths P_1 and P_2 may be edge-disjoint in the digraph but not edge-disjoint in the undirected graph.

if P_1 uses edge (u, v)
and P_2 uses its antiparallel edge (v, u)



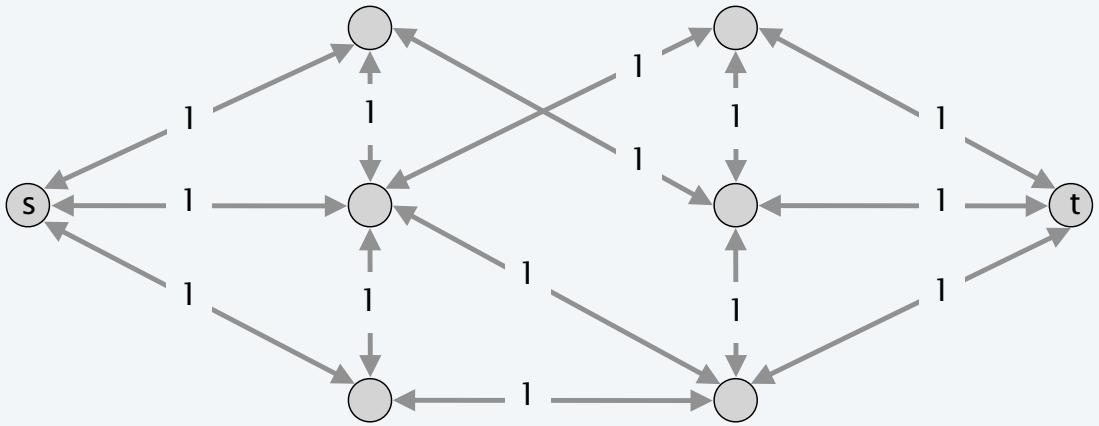
Edge-disjoint paths in undirected graphs

Max flow formulation. Replace edge edge with two antiparallel edges and assign unit capacity to every edge.

Lemma. In any flow network, there exists a maximum flow f in which for each pair of antiparallel edges e and e', either $f(e) = 0$ or $f(e') = 0$ or both. Moreover, integrality theorem still holds.

Pf. [by induction on number of such pairs of antiparallel edges]

- Suppose $f(e) > 0$ and $f(e') > 0$ for a pair of antiparallel edges e and e'.
- Set $f(e) = f(e) - \delta$ and $f(e') = f(e') - \delta$, where $\delta = \min \{ f(e), f(e') \}$.
- f is still a flow of the same value but has one fewer such pair. ▪

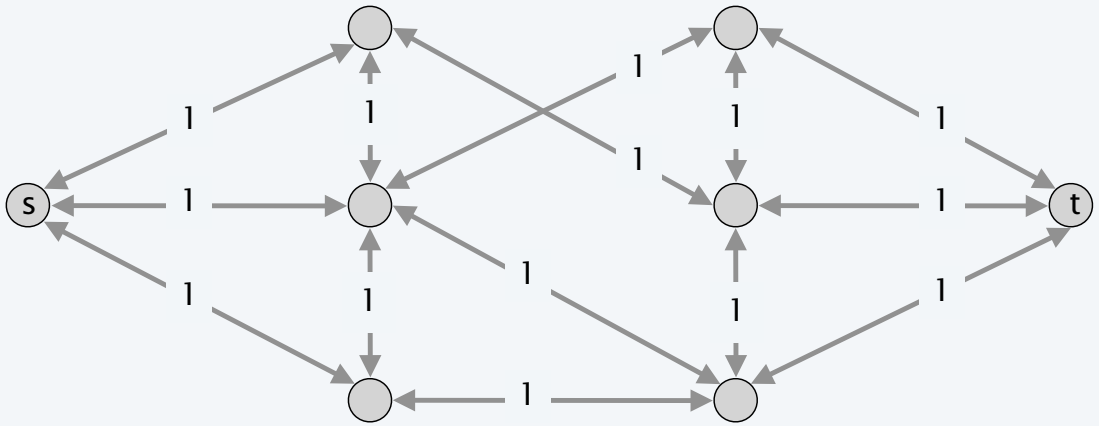


Edge-disjoint paths in undirected graphs

Max flow formulation. Replace edge edge with two antiparallel edges and assign unit capacity to every edge.

Lemma. In any flow network, there exists a maximum flow f in which for each pair of antiparallel edges e and e', either $f(e) = 0$ or $f(e') = 0$ or both. Moreover, integrality theorem still holds.

Theorem. Max number edge-disjoint $s \rightarrow t$ paths equals value of max flow.
Pf. Similar to proof in digraphs; use lemma.

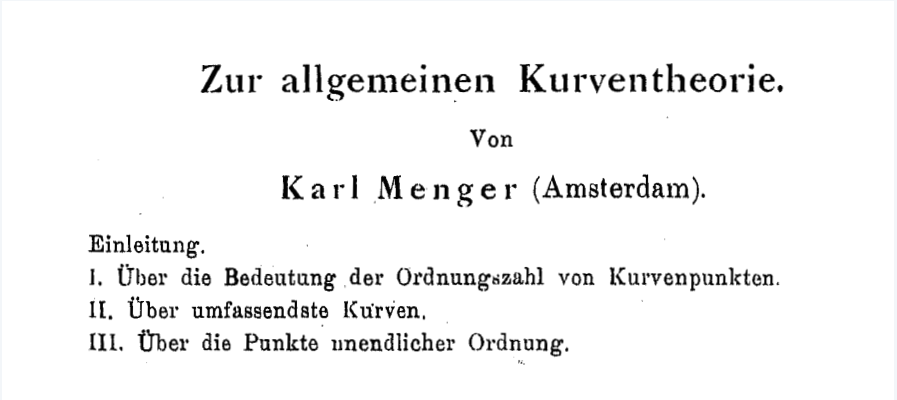


Menger's theorems

Theorem. Given an undirected graph with two nodes s and t, the max number of edge-disjoint s-t paths equals the min number of edges whose removal disconnects s and t.

Theorem. Given a undirected graph with two nonadjacent nodes s and t, the max number of internally node-disjoint s-t paths equals the min number of internal nodes whose removal disconnects s and t.

Theorem. Given an directed graph with two nonadjacent nodes s and t, the max number of internally node-disjoint $s \rightarrow t$ paths equals the min number of internal nodes whose removal disconnects t from s.

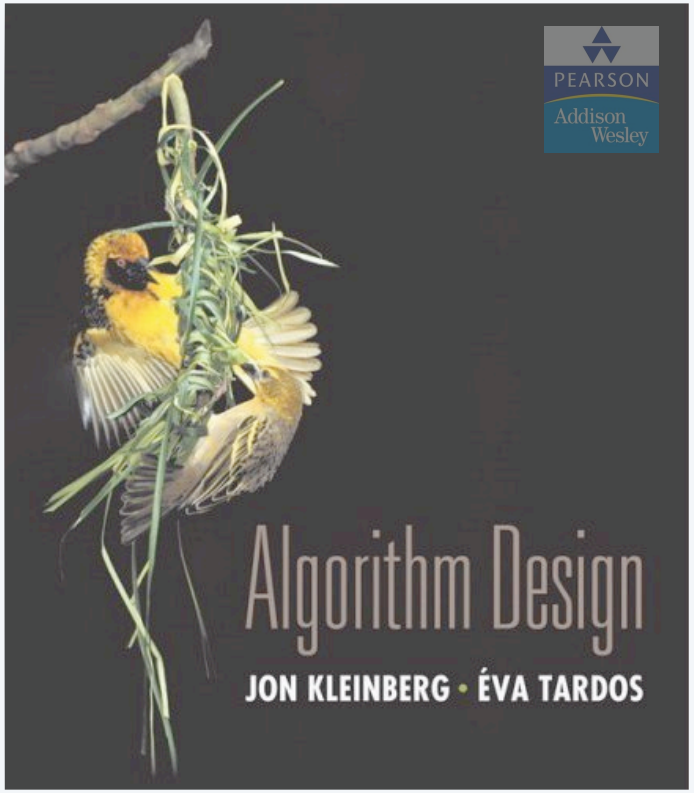


Zur allgemeinen Kurventheorie.

Von

Karl Menger (Amsterdam).

Einleitung.
I. Über die Bedeutung der Ordnungszahl von Kurvenpunkten.
II. Über umfassendste Kurven.
III. Über die Punkte unendlicher Ordnung.



SECTION

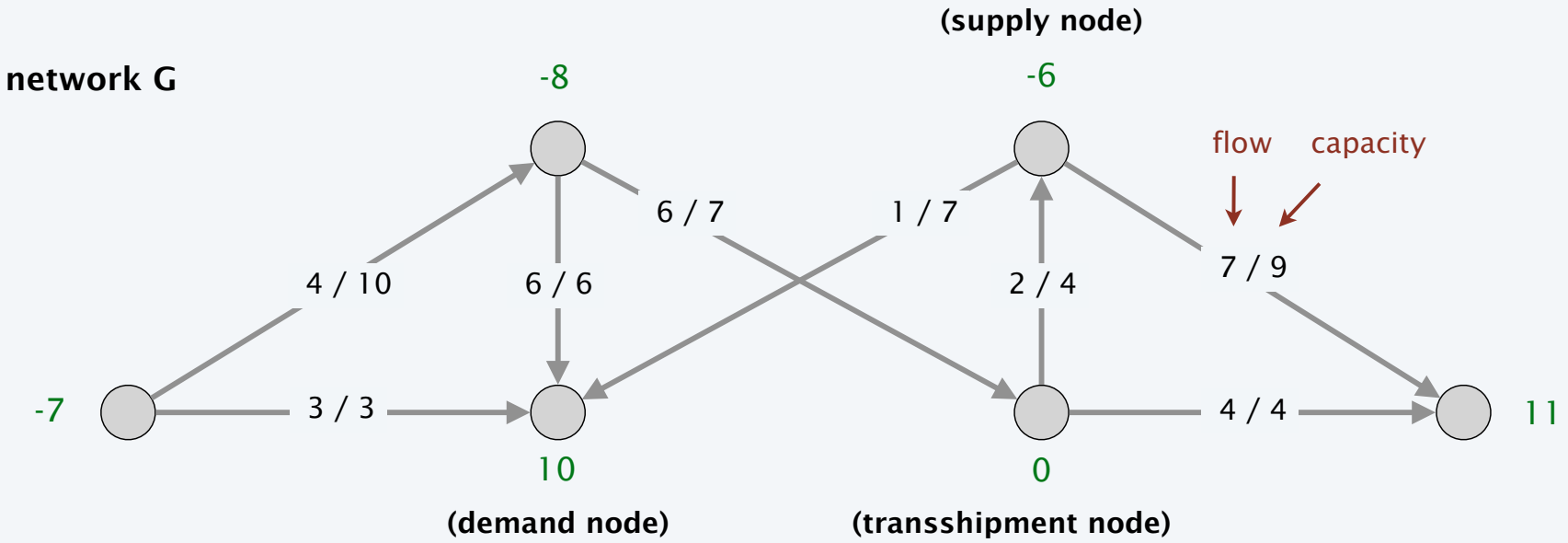
7. NETWORK FLOW II

- *bipartite matching*
- *disjoint paths*
- *extensions to max flow*
- *survey design*
- *airline scheduling*
- *image segmentation*
- *project selection*
- *baseball elimination*

Circulation with demands

Def. Given a digraph $G = (V, E)$ with nonnegative edge capacities $c(e)$ and node supply and demands $d(v)$, a circulation is a function that satisfies:

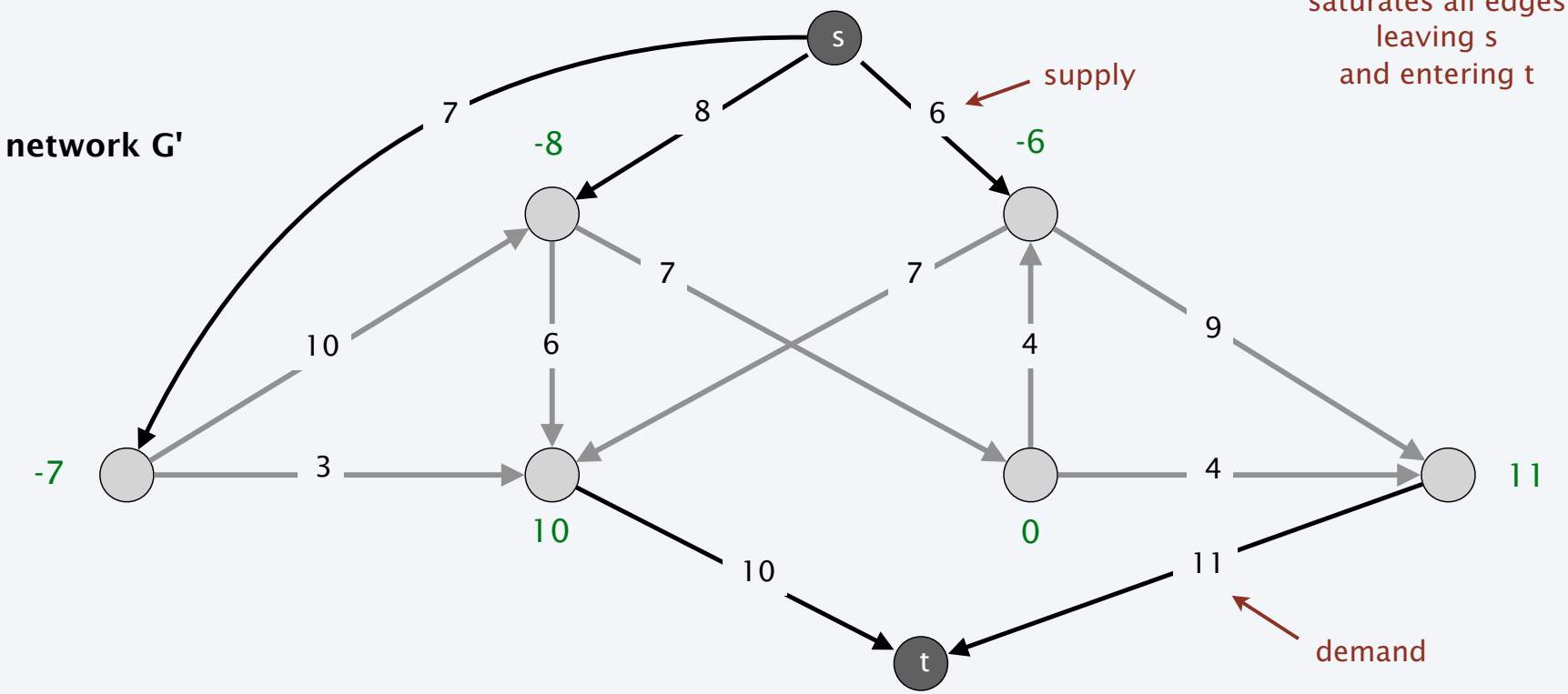
- For each $e \in E$: $0 \le f(e) \le c(e)$ (capacity)
- For each $v \in V$: $\sum_{e \text{ in to } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$ (conservation)



Circulation with demands: max-flow formulation

- Add new source s and sink t.
- For each v with $d(v) < 0$, add edge (s, v) with capacity $-d(v)$.
- For each v with $d(v) > 0$, add edge (v, t) with capacity $d(v)$.

Claim. G has circulation iff G' has max flow of value $D = \sum_{v : d(v) > 0} d(v) = \sum_{v : d(v) < 0} -d(v)$



Circulation with demands

Integrality theorem. If all capacities and demands are integers, and there exists a circulation, then there exists one that is integer-valued.

Pf. Follows from max-flow formulation + integrality theorem for max flow.

Theorem. Given (V, E, c, d), there does not exist a circulation iff there exists a node partition (A, B) such that $\Sigma_{v \in B} d(v) > cap(A, B)$.

Pf sketch. Look at min cut in G'.

demand by nodes in B exceeds
supply of nodes in B plus
max capacity of edges going from A to B

Circulation with demands and lower bounds

Feasible circulation.

- Directed graph $G = (V, E)$.
- Edge capacities $c(e)$ and lower bounds $\ell(e)$ for each edge $e \in E$.
- Node supply and demands $d(v)$ for each node $v \in V$.

Def. A circulation is a function that satisfies:

- For each $e \in E$: $\ell(e) \le f(e) \le c(e)$ (capacity)
- For each $v \in V$: $\sum_{e \text{ in to } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$ (conservation)

Circulation problem with lower bounds. Given (V, E, ℓ, c, d), does there exist a feasible circulation?

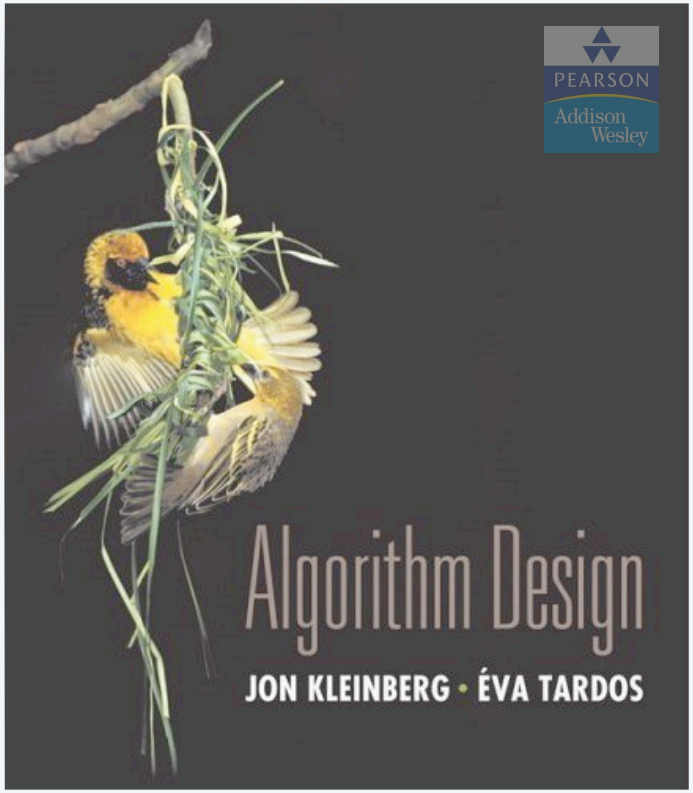
Circulation with demands and lower bounds

Max flow formulation. Model lower bounds as circulation with demands.

- Send $\ell(e)$ units of flow along edge e.
- Update demands of both endpoints.

Theorem. There exists a circulation in G iff there exists a circulation in G'. Moreover, if all demands, capacities, and lower bounds in G are integers, then there is a circulation in G that is integer-valued.

Pf sketch. $f(e)$ is a circulation in G iff $f'(e) = f(e) - \ell(e)$ is a circulation in G'.



SECTION

7. NETWORK FLOW II

- *bipartite matching*
- *disjoint paths*
- *extensions to max flow*
- *survey design*
- *airline scheduling*
- *image segmentation*
- *project selection*
- *baseball elimination*

Survey design

- Design survey asking n_1 consumers about n_2 products. ← one survey question per product
- Can only survey consumer i about product j if they own it.
- Ask consumer i between c_i and c_i' questions.
- Ask between p_j and p_j' consumers about product j.

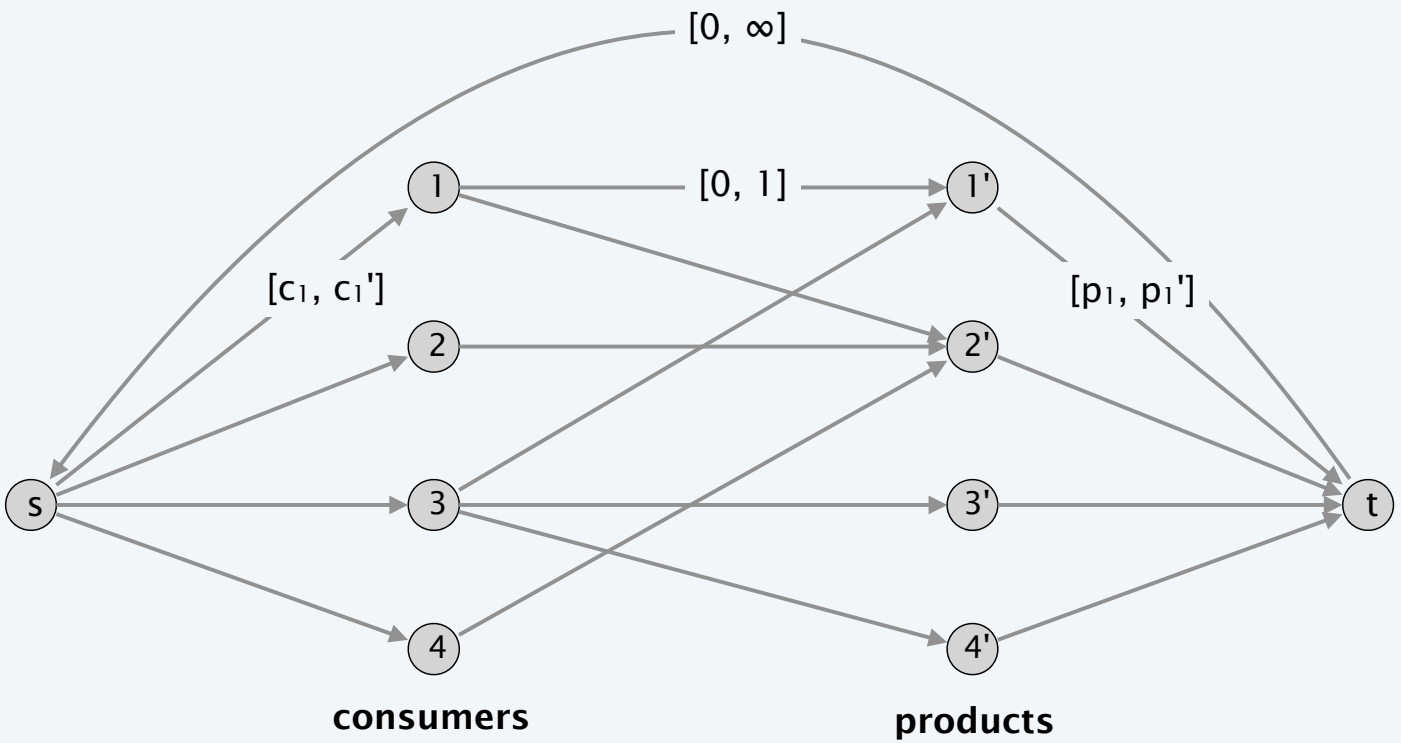
Goal. Design a survey that meets these specs, if possible.

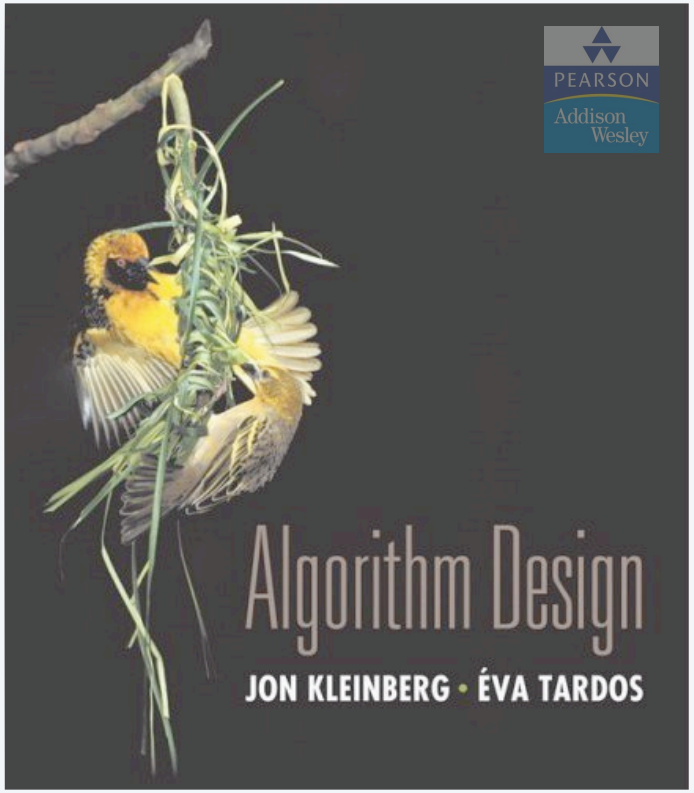
Bipartite perfect matching. Special case when $c_i = c_i' = p_j = p_j' = 1$.

Survey design

Max-flow formulation. Model as circulation problem with lower bounds.

- Add edge (i, j) if consumer j owns product i.
- Add edge from s to consumer j .
- Add edge from product i to t.
- Add edge from t to s.
- Integer circulation $\Leftrightarrow$ feasible survey design.





SECTION

7. NETWORK FLOW II

- *bipartite matching*
- *disjoint paths*
- *extensions to max flow*
- *survey design*
- *airline scheduling*
- *image segmentation*
- *project selection*
- *baseball elimination*

Airline scheduling

Airline scheduling.

- Complex computational problem faced by nation's airline carriers.
- Produces schedules that are efficient in terms of:
  - equipment usage, crew allocation, customer satisfaction
  - in presence of unpredictable issues like weather, breakdowns
- One of largest consumers of high-powered algorithmic techniques.

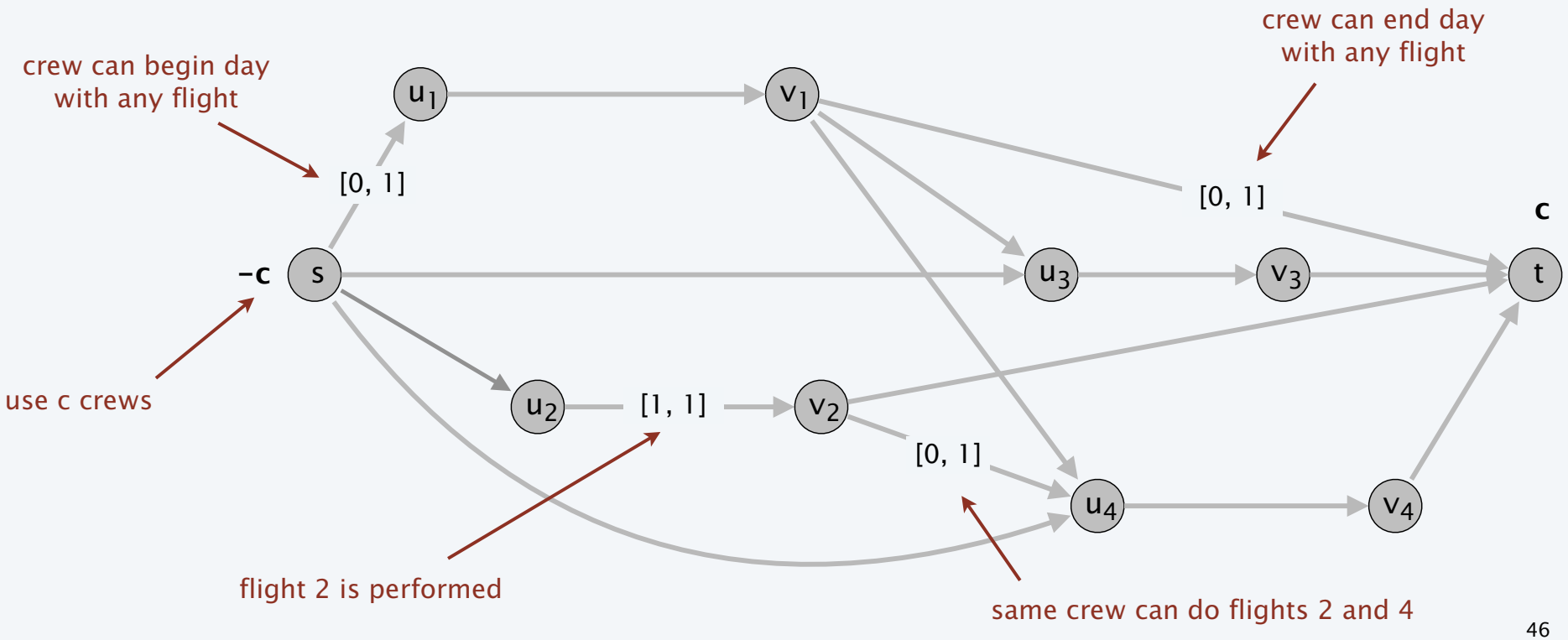
"Toy problem."

- Manage flight crews by reusing them over multiple flights.
- Input: set of k flights for a given day.
- Flight i leaves origin o_i at time s_i and arrives at destination d_i destination at time f_i.
- Minimize number of flight crews.

Airline scheduling

Circulation formulation. [to see if c crews suffice]

- For each flight i, include two nodes u_i and v_i.
- Add source s with demand $-c$, and edges (s, u_i) with capacity 1.
- Add sink t with demand c, and edges (v_i, t) with capacity 1.
- For each i, add edge (u_i, v_i) with lower bound and capacity 1.
- if flight j reachable from i, add edge (v_i, u_j) with capacity 1.



Airline scheduling: running time

Theorem. The airline scheduling problem can be solved in $O(k^3 \log k)$ time.

Pf.

- k = number of flights.
- c = number of crews (unknown).
- $O(k)$ nodes, $O(k^2)$ edges.
- At most k crews needed.

  $\Rightarrow$ solve $\lg k$ circulation problems. ← binary search for optimal value c*
- Value of the flow is between 0 and k.

  $\Rightarrow$ at most k augmentations per circulation problem.
- Overall time = $O(k^3 \log k)$.

Remark. Can solve in $O(k^3)$ time by formulating as minimum flow problem.

Airline scheduling: postmortem

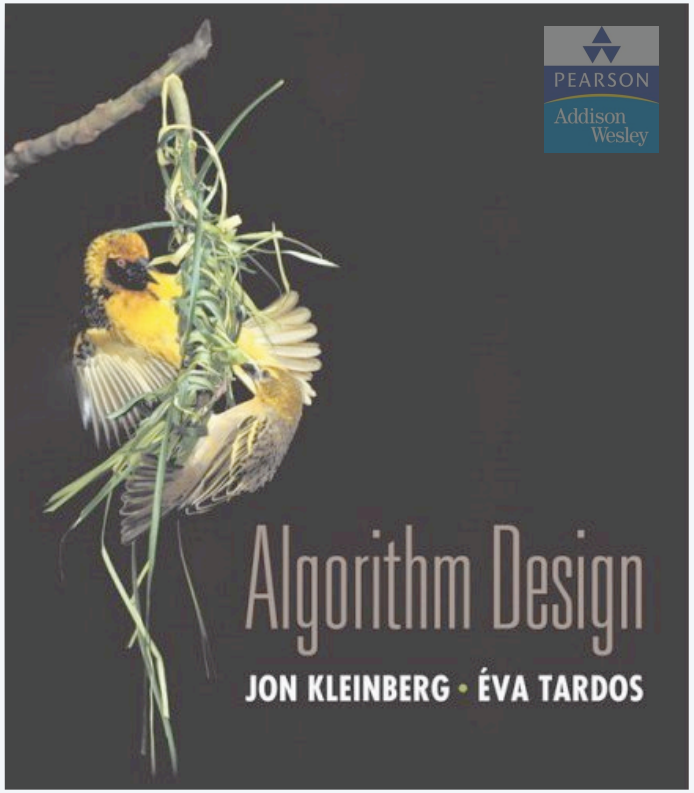
Remark. We solved a toy problem.

Real-world problem models countless other factors:

- Union regulations: e.g., flight crews can only fly certain number of hours in given interval.
- Need optimal schedule over planning horizon, not just one day.
- Deadheading has a cost.
- Flights don't always leave or arrive on schedule.
- Simultaneously optimize both flight schedule and fare structure.

Message.

- Our solution is a generally useful technique for efficient reuse of limited resources but trivializes real airline scheduling problem.
- Flow techniques useful for solving airline scheduling problems (and are widely used in practice).
- Running an airline efficiently is a very difficult problem.



SECTION

7. NETWORK FLOW II

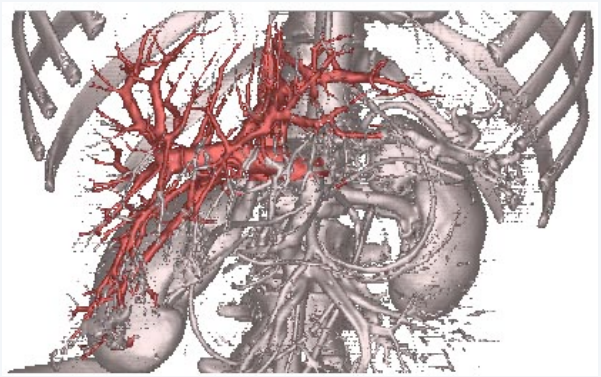
- *bipartite matching*
- *disjoint paths*
- *extensions to max flow*
- *survey design*
- *airline scheduling*
- *image segmentation*
- *project selection*
- *baseball elimination*

Image segmentation

Image segmentation.

- Central problem in image processing.
- Divide image into coherent regions.

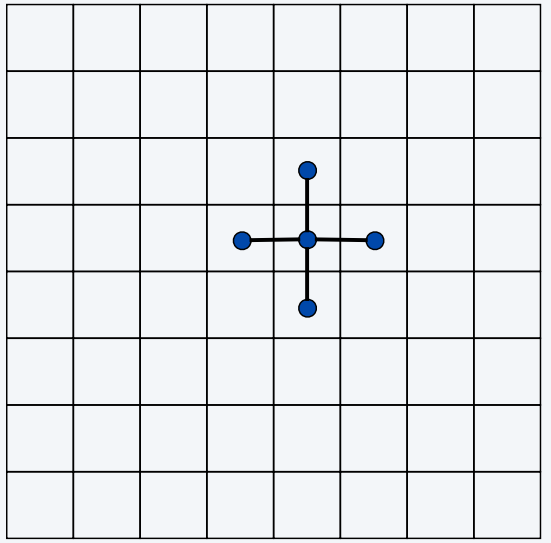
Ex. Three people standing in front of complex background scene.
Identify each person as a coherent object.



liver and hepatic vascularization segmentation

Image segmentation

Foreground / background segmentation.



- Label each pixel in picture as belonging to foreground or background.
- V = set of pixels, E = pairs of neighboring pixels.
- $a_i \geq 0$ is likelihood pixel i in foreground.
- $b_i \geq 0$ is likelihood pixel i in background.
- $p_{ij} \geq 0$ is separation penalty for labeling one of i and j as foreground, and the other as background.

Goals.

- Accuracy: if $a_i > b_i$ in isolation, prefer to label i in foreground.
- Smoothness: if many neighbors of i are labeled foreground, we should be inclined to label i as foreground.
- Find partition (A, B) that maximizes: $\sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij}$

foreground background

Image segmentation

Formulate as min cut problem.

- Maximization.
- No source or sink.
- Undirected graph.

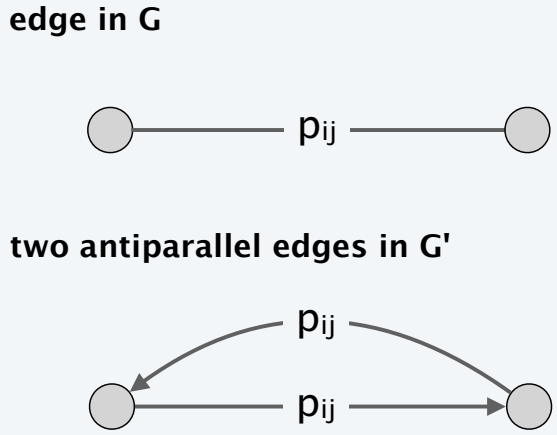
Turn into minimization problem.

- Maximizing $\sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij}$
- is equivalent to minimizing $\underbrace{\left(\sum_{i \in V} a_i + \sum_{j \in V} b_j \right)}_{\text{a constant}} - \sum_{i \in A} a_i - \sum_{j \in B} b_j + \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij}$
- or alternatively $\sum_{j \in B} a_j + \sum_{i \in A} b_i + \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij}$

Image segmentation

Formulate as min cut problem $G' = (V', E')$.

- Include node for each pixel.
- Use two antiparallel edges instead of undirected edge.
- Add source s to correspond to foreground.
- Add sink t to correspond to background.



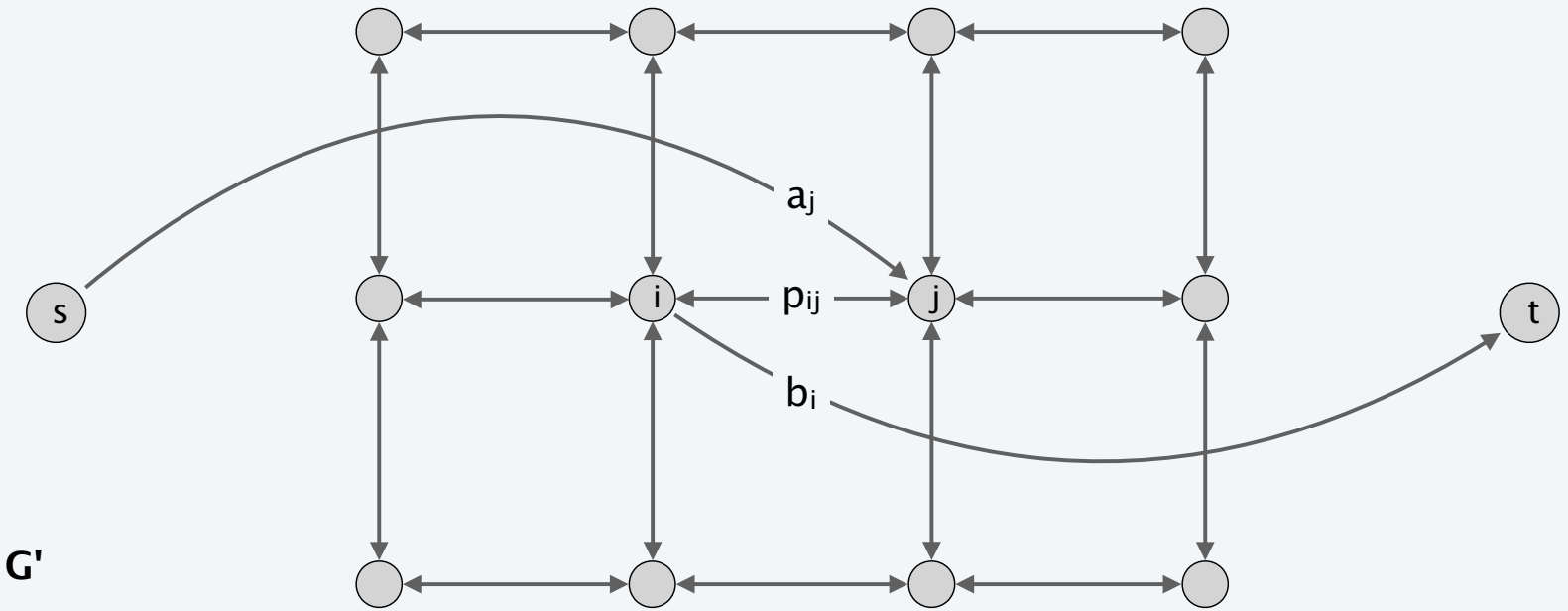


Image segmentation

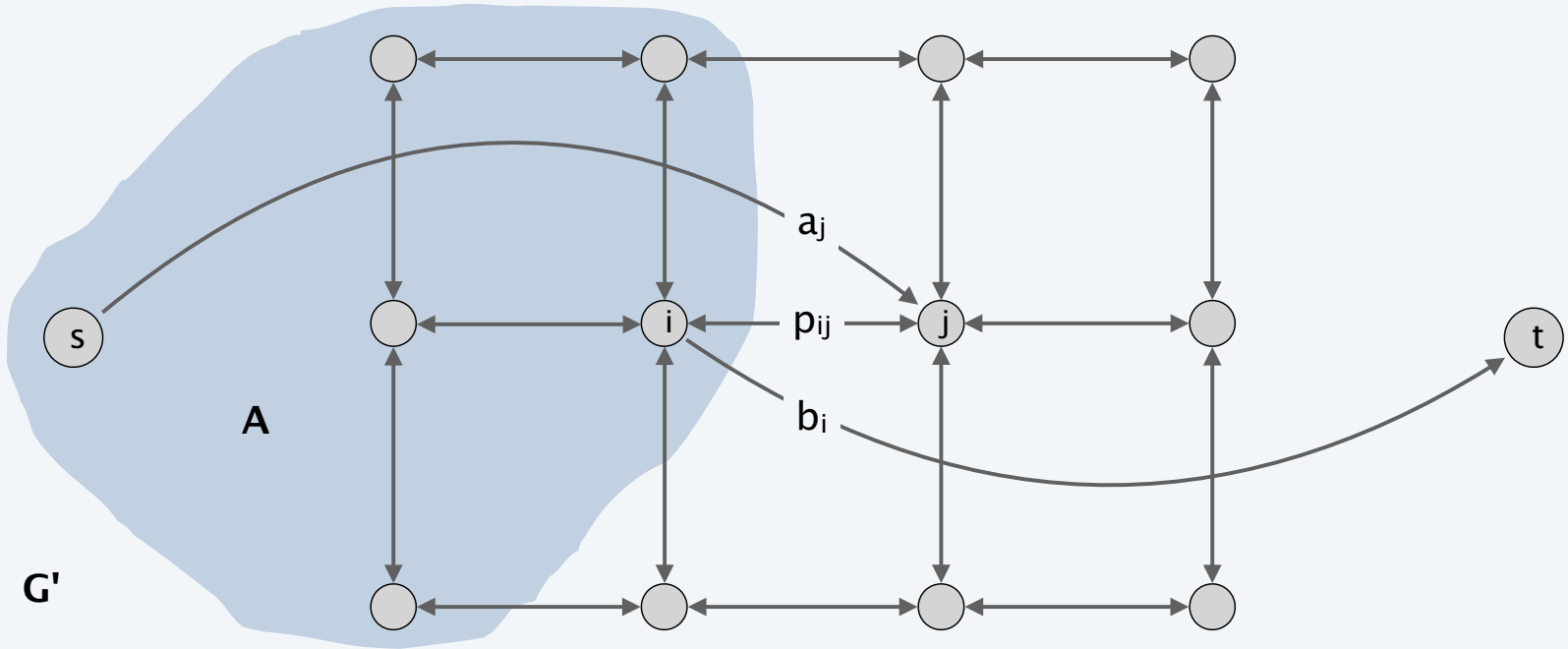
Consider min cut (A, B) in G'.

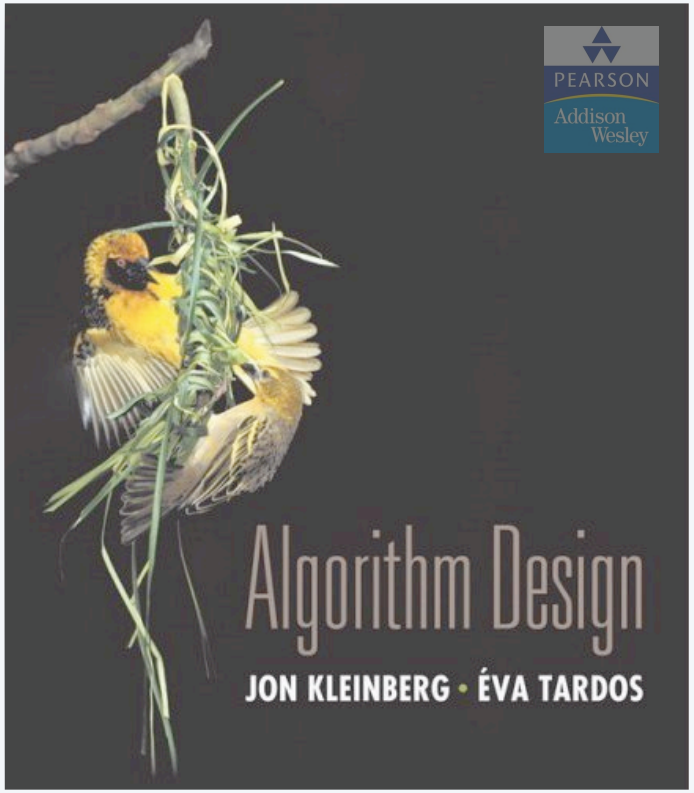
- A = foreground.

$$cap(A,B) = \sum_{j \in B} a_j + \sum_{i \in A} b_i + \sum_{\substack{(i,j) \in E \\ i \in A,\ j \in B}} p_{ij}$$

← if i and j on different sides, p_{ij} counted exactly once

- Precisely the quantity we want to minimize.





SECTION

7. NETWORK FLOW II

- *bipartite matching*
- *disjoint paths*
- *extensions to max flow*
- *survey design*
- *airline scheduling*
- *image segmentation*
- *project selection*
- *baseball elimination*

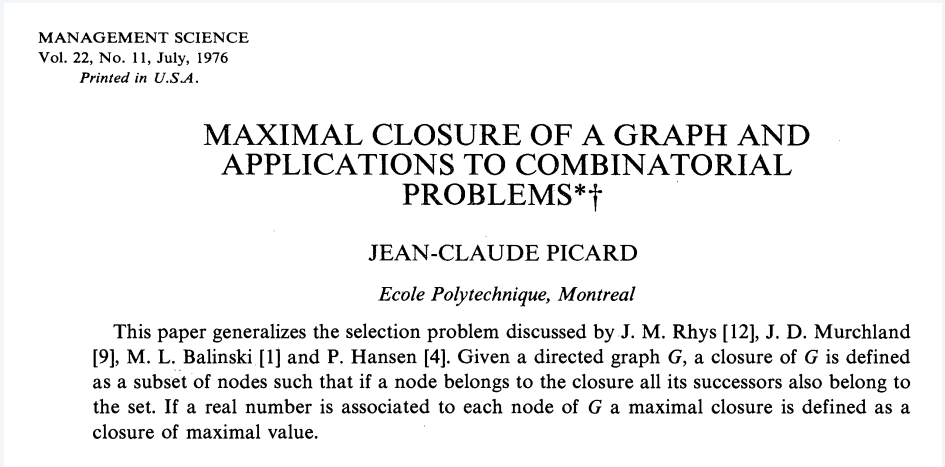
Project selection (maximum weight closure problem)

Projects with prerequisites.

can be positive or negative

- Set of possible projects P: project v has associated revenue p_v.
- Set of prerequisites E: if $(v, w) \in E$, cannot do project v unless also do project w.
- A subset of projects $A \subseteq P$ is feasible if the prerequisite of every project in A also belongs to A.

Project selection problem. Given a set of projects P and prerequisites E, choose a feasible subset of projects to maximize revenue.



MANAGEMENT SCIENCE
Vol. 22, No. 11, July, 1976
Printed in U.S.A.

MAXIMAL CLOSURE OF A GRAPH AND APPLICATIONS TO COMBINATORIAL PROBLEMS*†

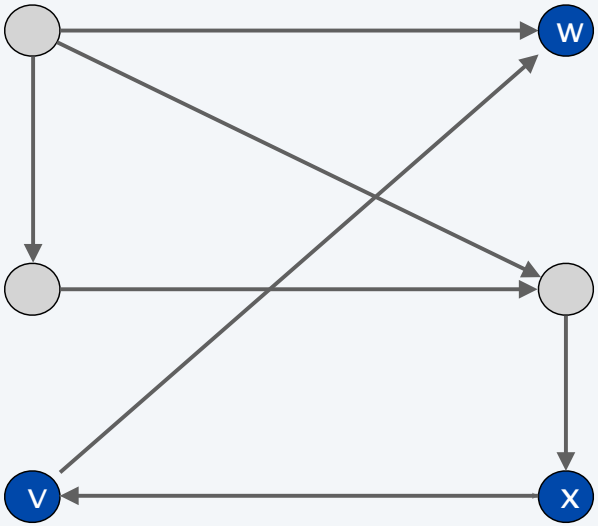
JEAN-CLAUDE PICARD

Ecole Polytechnique, Montreal

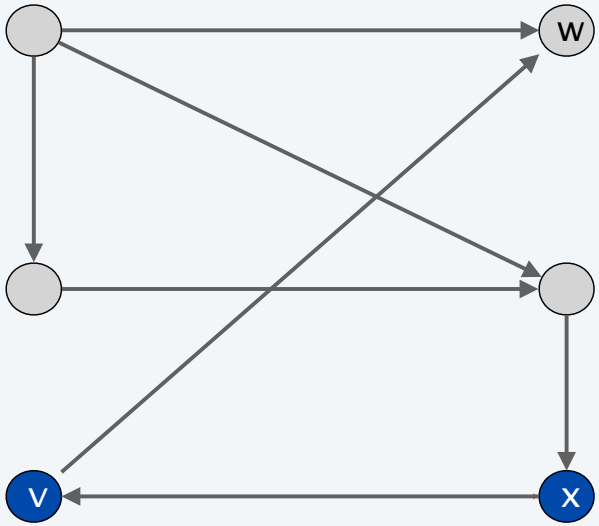
This paper generalizes the selection problem discussed by J. M. Rhys [12], J. D. Murchland [9], M. L. Balinski [1] and P. Hansen [4]. Given a directed graph G, a closure of G is defined as a subset of nodes such that if a node belongs to the closure all its successors also belong to the set. If a real number is associated to each node of G a maximal closure is defined as a closure of maximal value.

Project selection: prerequisite graph

Prerequisite graph. Add edge (v, w) if can't do v without also doing w.



{ v, w, x } is feasible

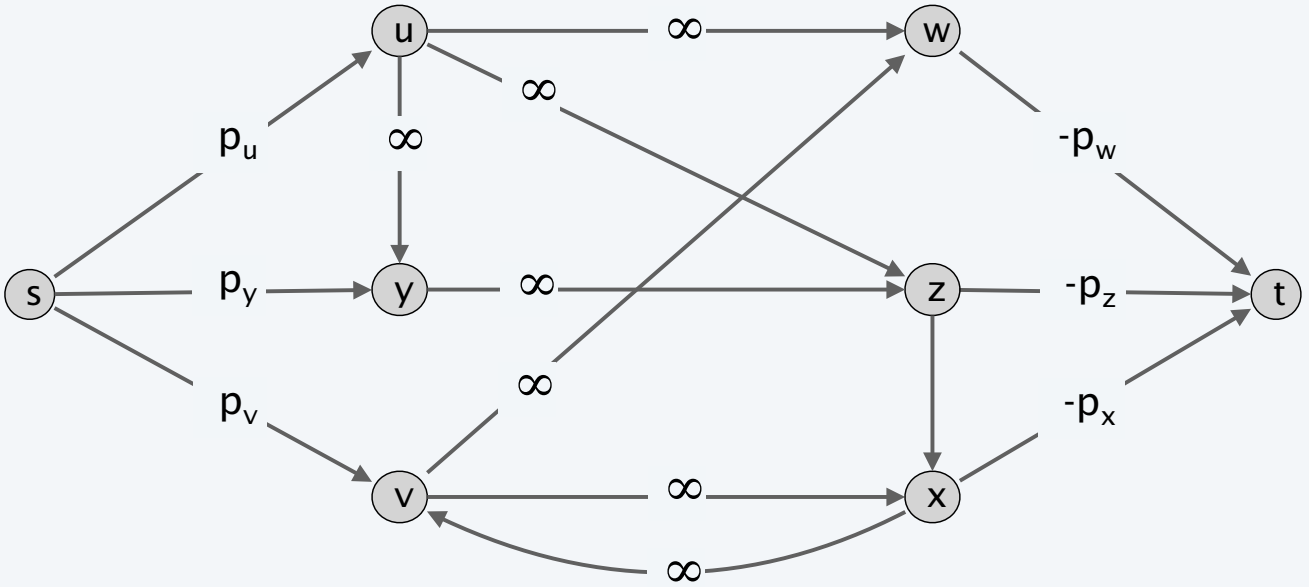


{ v, x } is infeasible

Project selection: min-cut formulation

Min-cut formulation.

- Assign capacity ∞ to all prerequisite edge.
- Add edge (s, v) with capacity p_v if $p_v > 0$.
- Add edge (v, t) with capacity $-p_v$ if $p_v < 0$.
- For notational convenience, define $p_s = p_t = 0$.



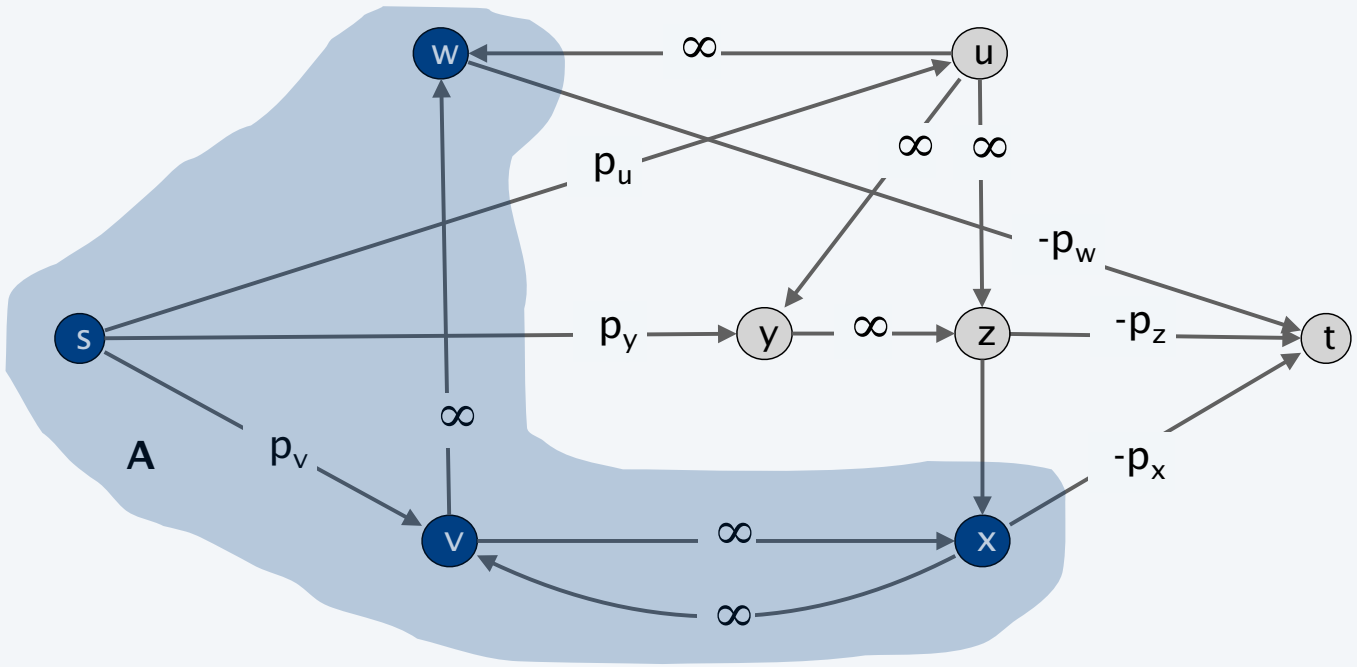
Project selection: min-cut formulation

Claim. (A, B) is min cut iff $A - \{ s \}$ is optimal set of projects.

- Infinite capacity edges ensure $A - \{ s \}$ is feasible.
- Max revenue because:

$$cap(A, B) = \sum_{v \in B:\, p_v > 0} p_v + \sum_{v \in A:\, p_v < 0} (-p_v)$$

$$= \underbrace{\sum_{v:\, p_v > 0} p_v}_{\text{constant}} - \sum_{v \in A} p_v$$



Open-pit mining

Open-pit mining. (studied since early 1960s)

- Blocks of earth are extracted from surface to retrieve ore.
- Each block v has net value p_v = value of ore − processing cost.
- Can't remove block v before w or x.