http://algs4.cs.princeton.edu

6.4 Maximum Flow

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

http://algs4.cs.princeton.edu

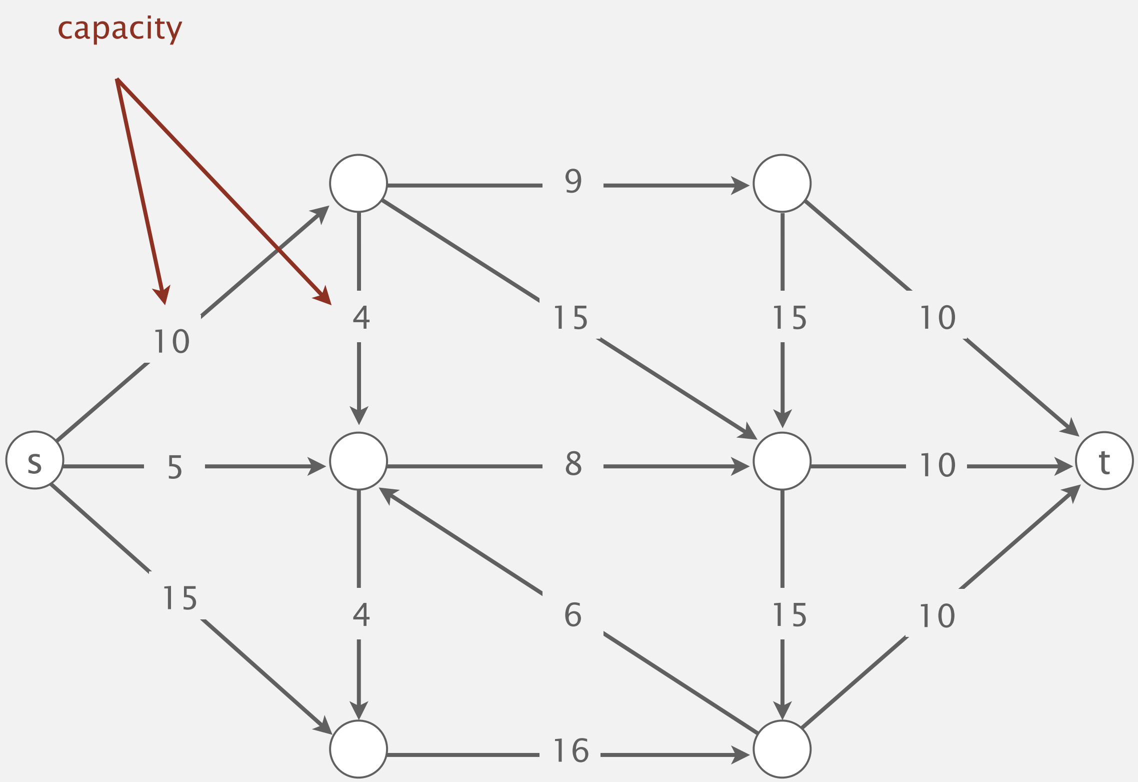
6.4 Maximum Flow

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

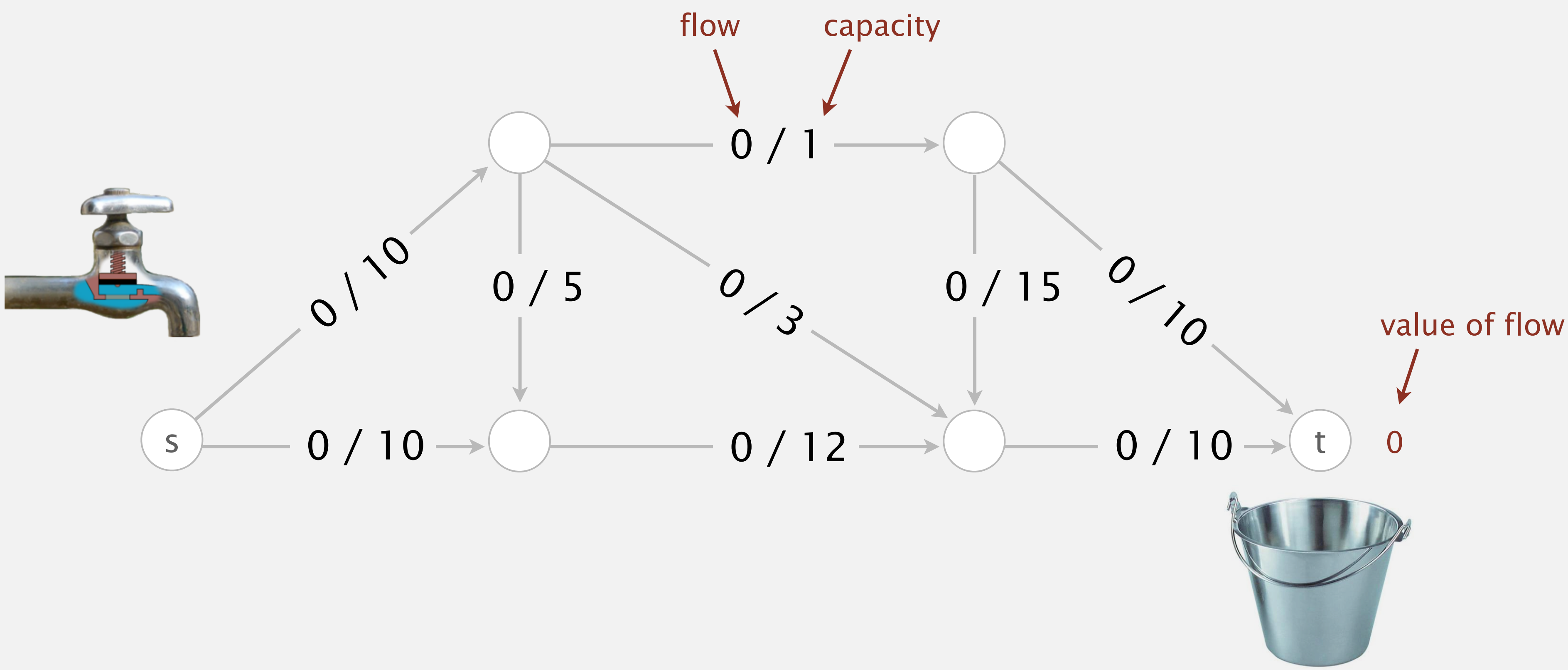
Mincut problem

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

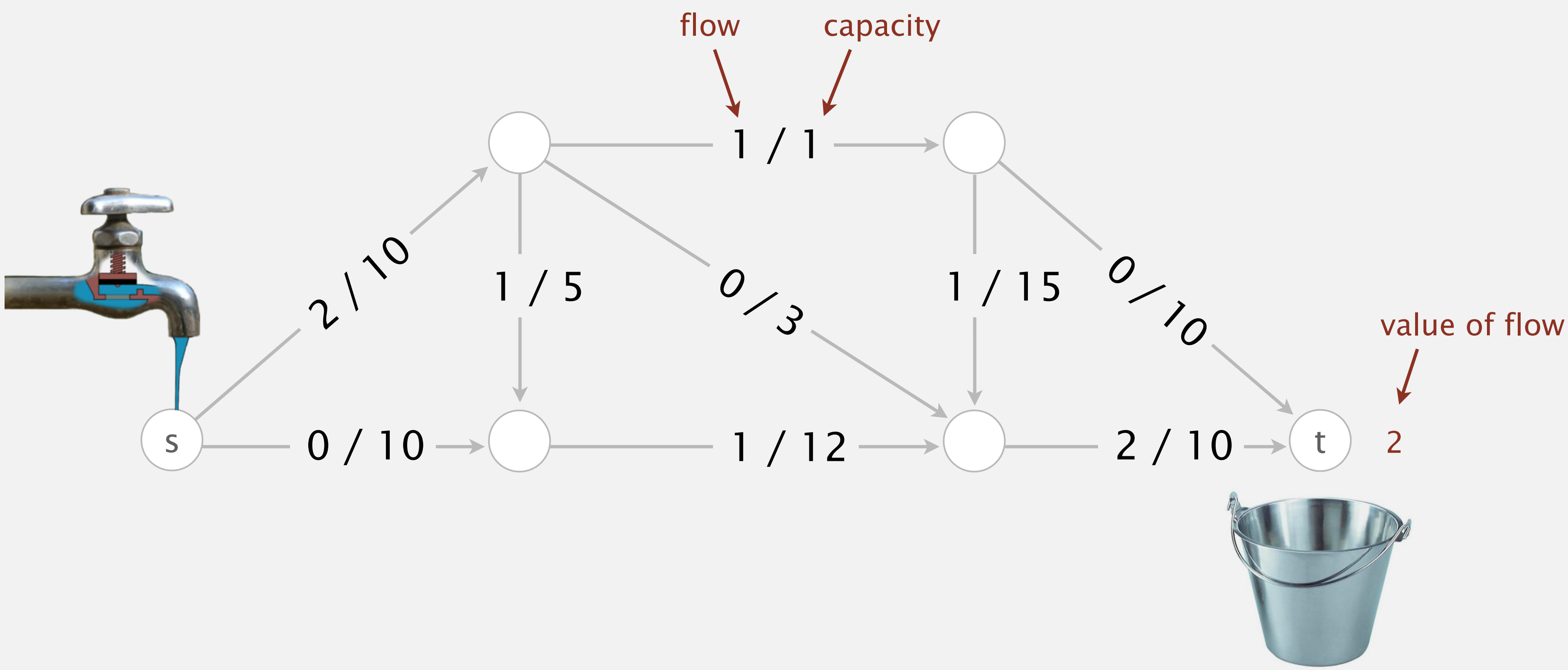
each edge has a positive capacity



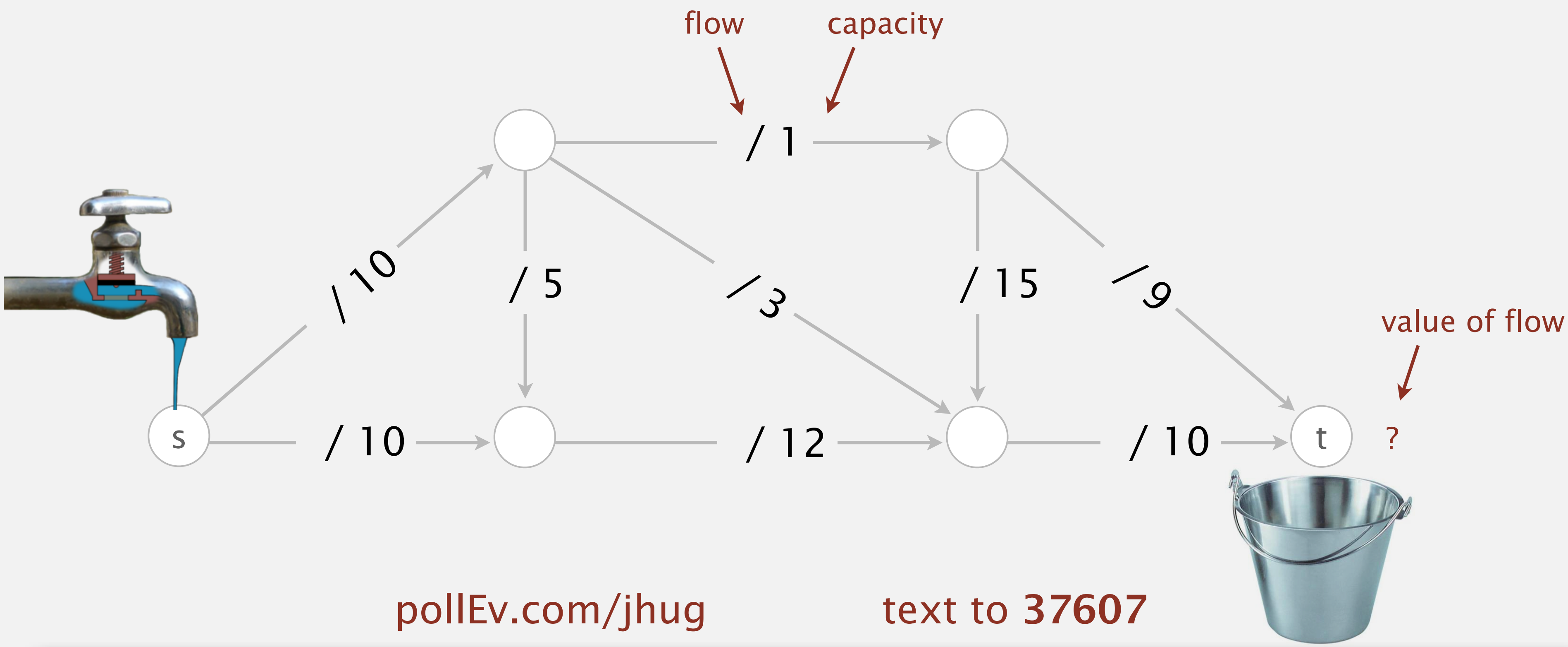
Max flow / min cut problem



Max flow / min cut problem



Max flow / min cut problem

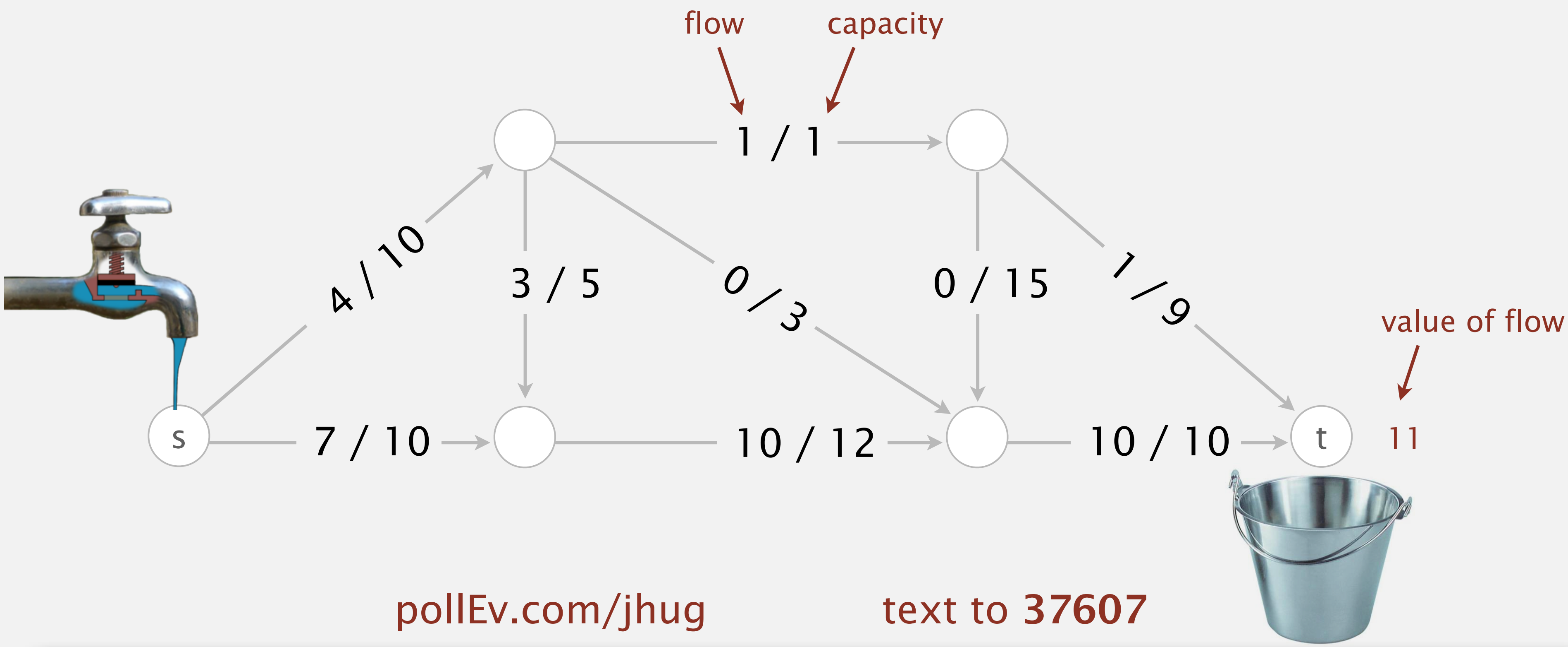


pollEv.com/jhug text to **37607**

Q: What is the value of the max flow?

```
A. 20        [191071]        D. 11        [170148]
B. 19        [175058]        E. 10        [170215]
C. 16        [170059]

Extra: Find a cut whose total capacity equals the max flow.
```

Max flow / min cut problem

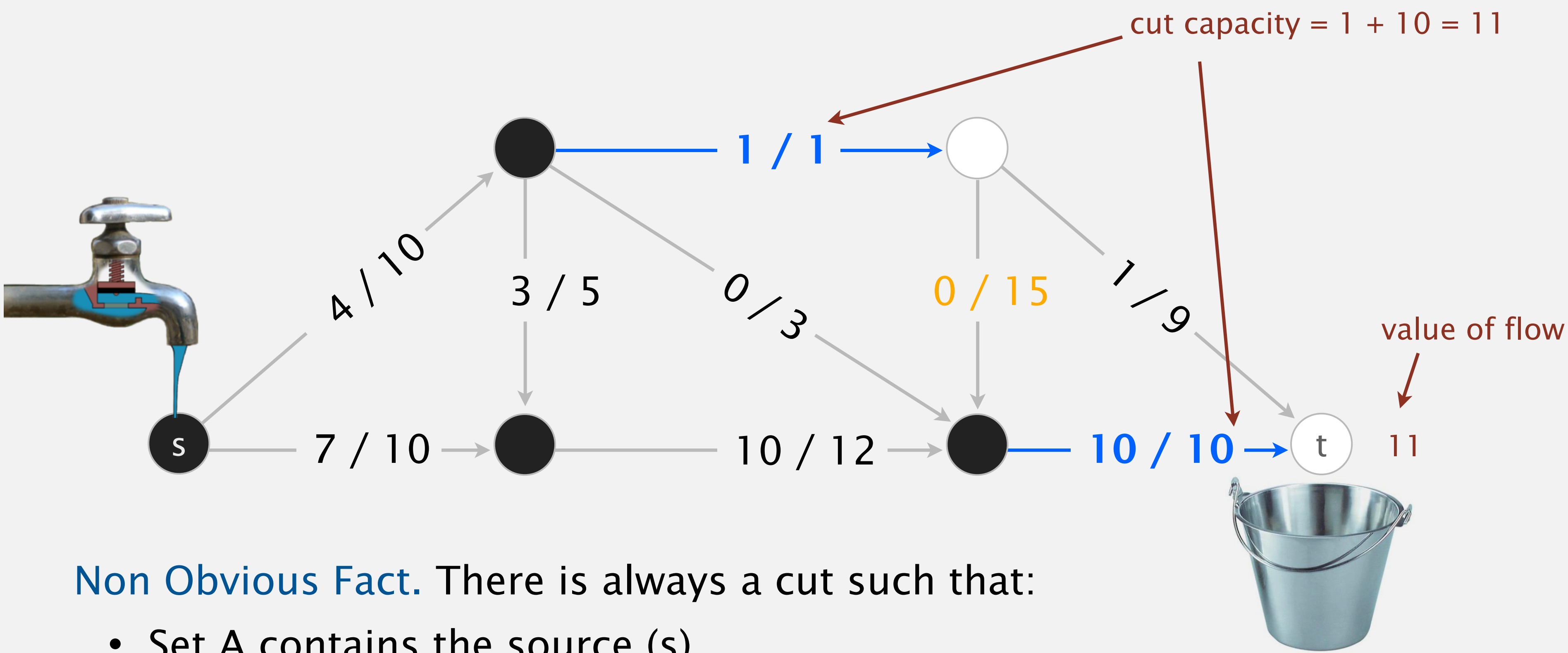


pollEv.com/jhug text to **37607**

Q: What is the value of the max flow?

```
D. 11

Extra: Find a cut whose total capacity equals the max flow.
```

Max flow / min cut problem



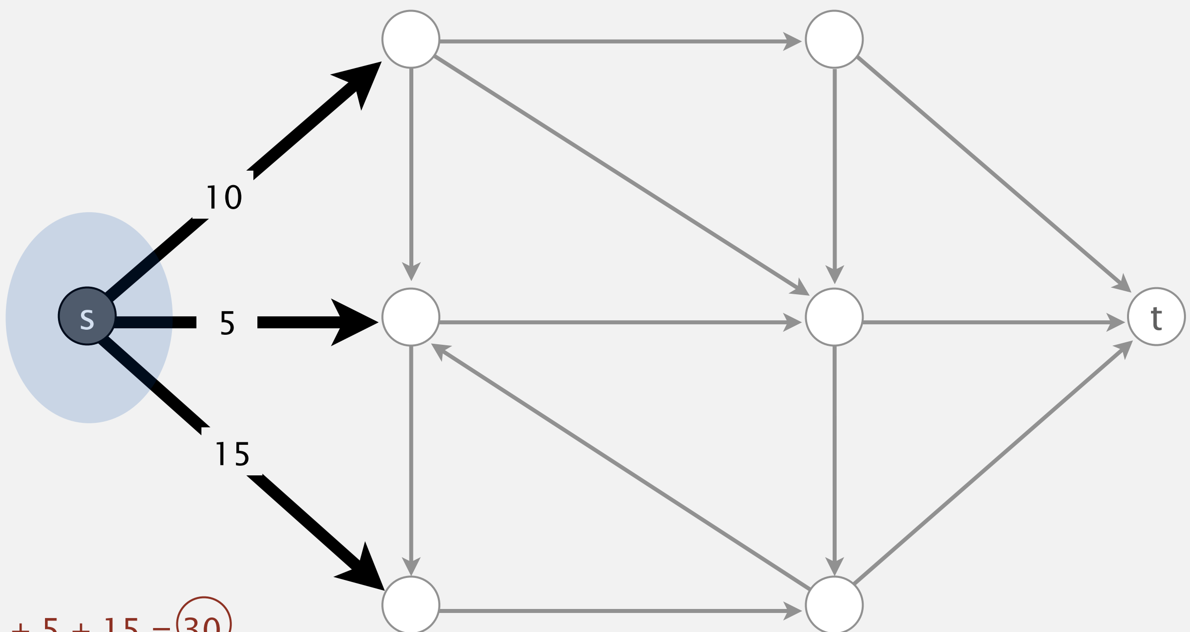
Non Obvious Fact. There is always a cut such that:

- Set A contains the source (s).
- Set B contains the sink (t).
- The capacity of this cut is equal to the value of the max flow.
- All edges from A to B are **full**.
- All edges from B to A are **empty**.

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.

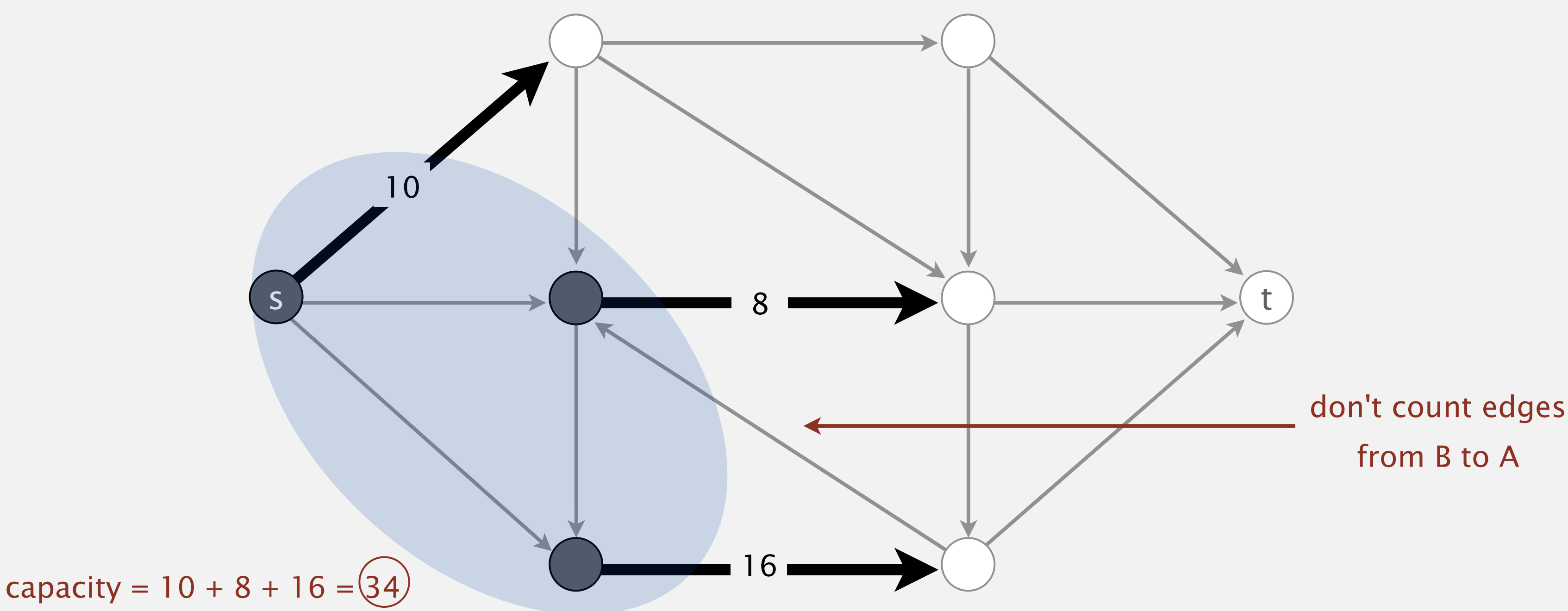


capacity = 10 + 5 + 15 = 30

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.

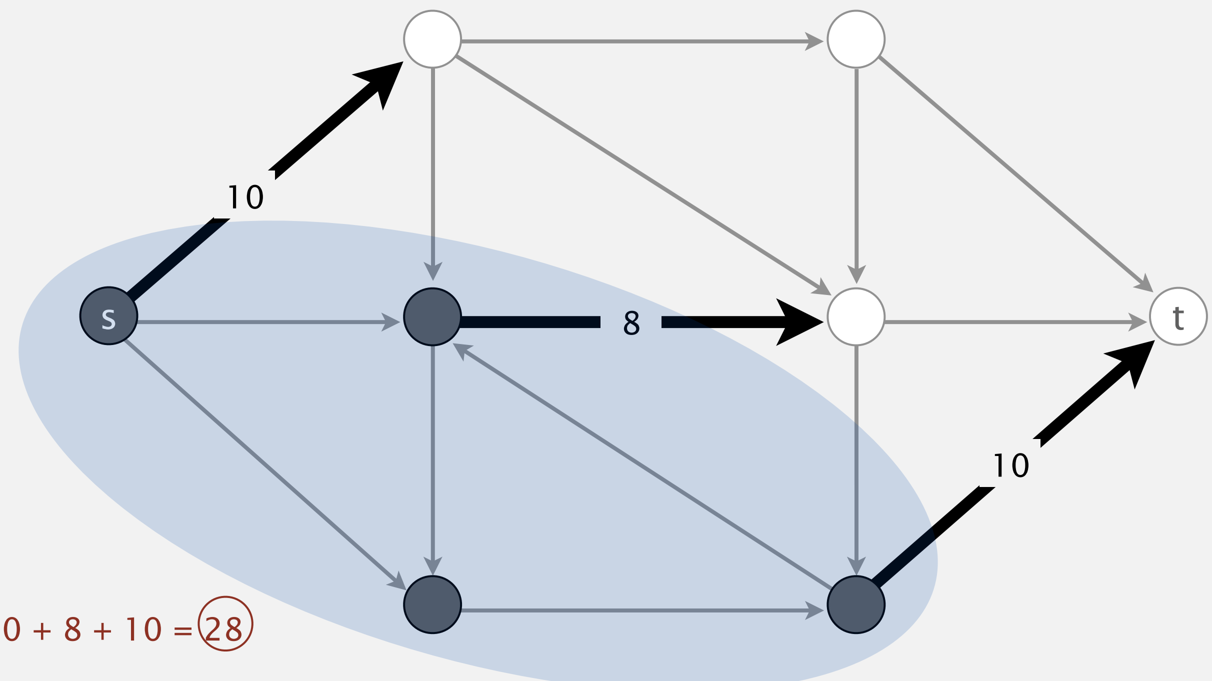


Mincut problem

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.

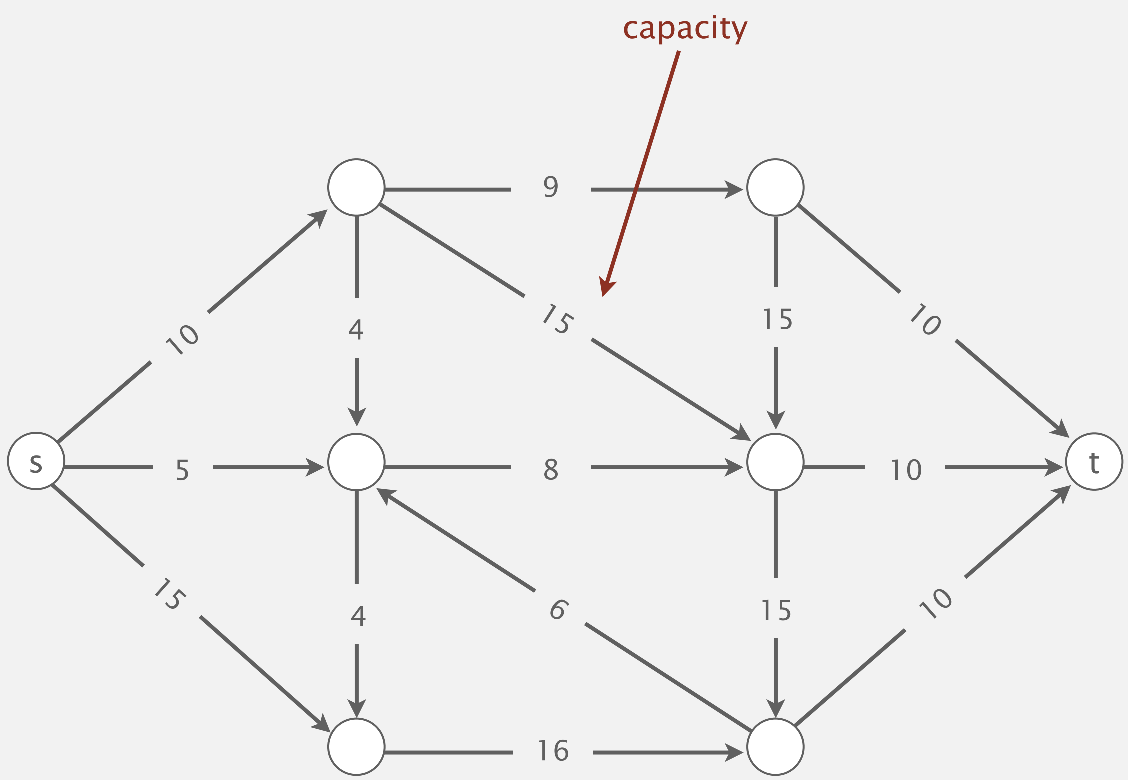


capacity = 10 + 8 + 10 = 28

Maxflow problem

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

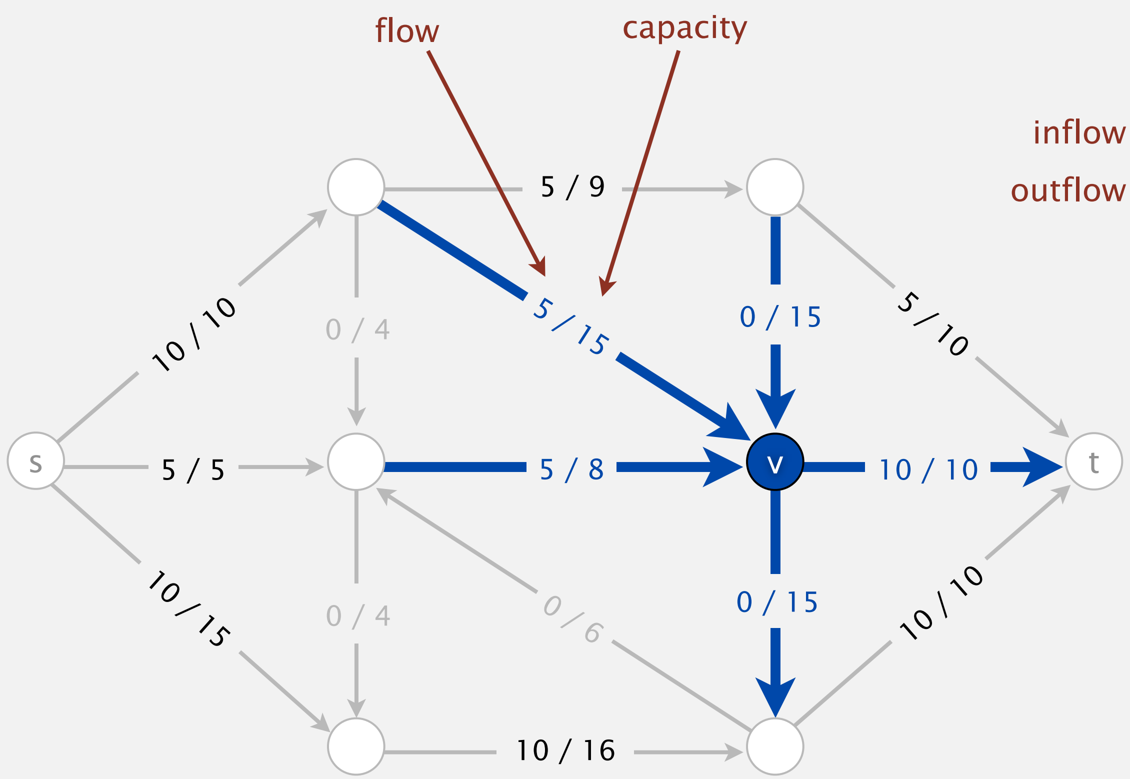
each edge has a
positive capacity



Maxflow problem

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

- Capacity constraint: $0 \leq$ edge's flow $\leq$ 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

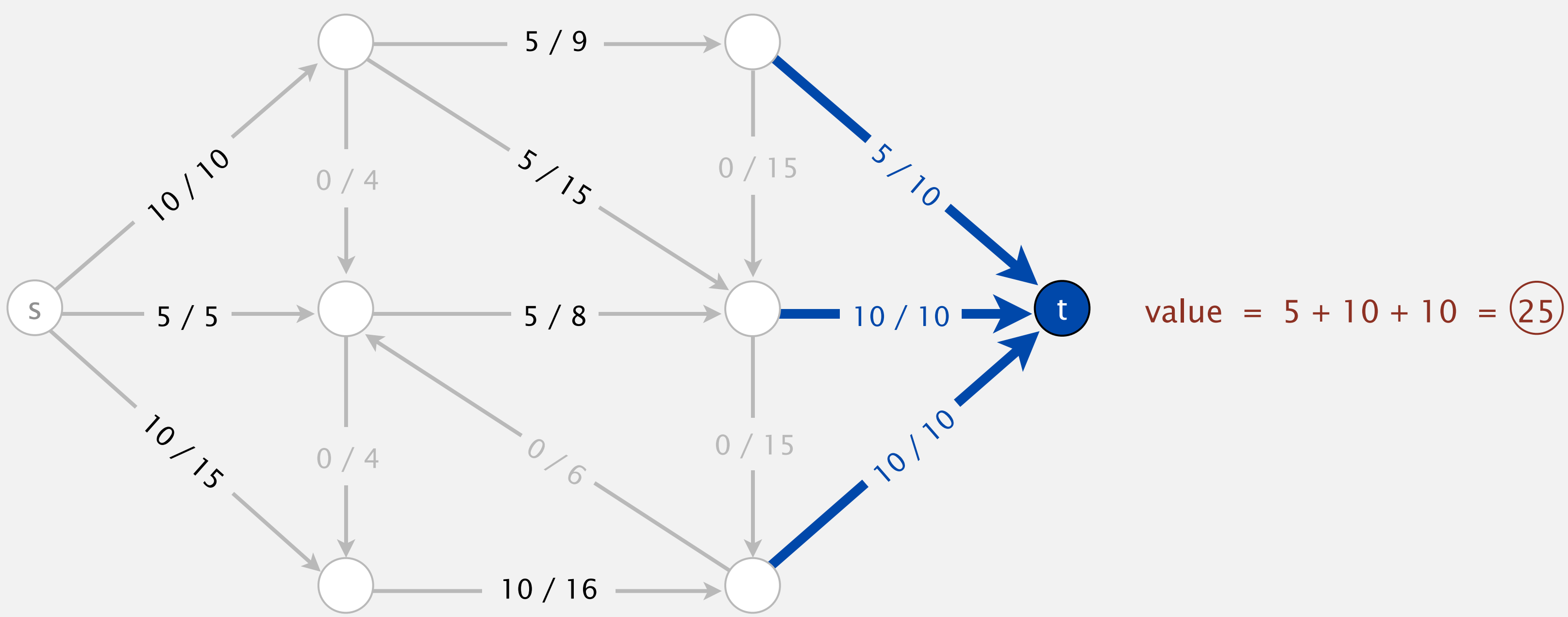
Maxflow problem

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



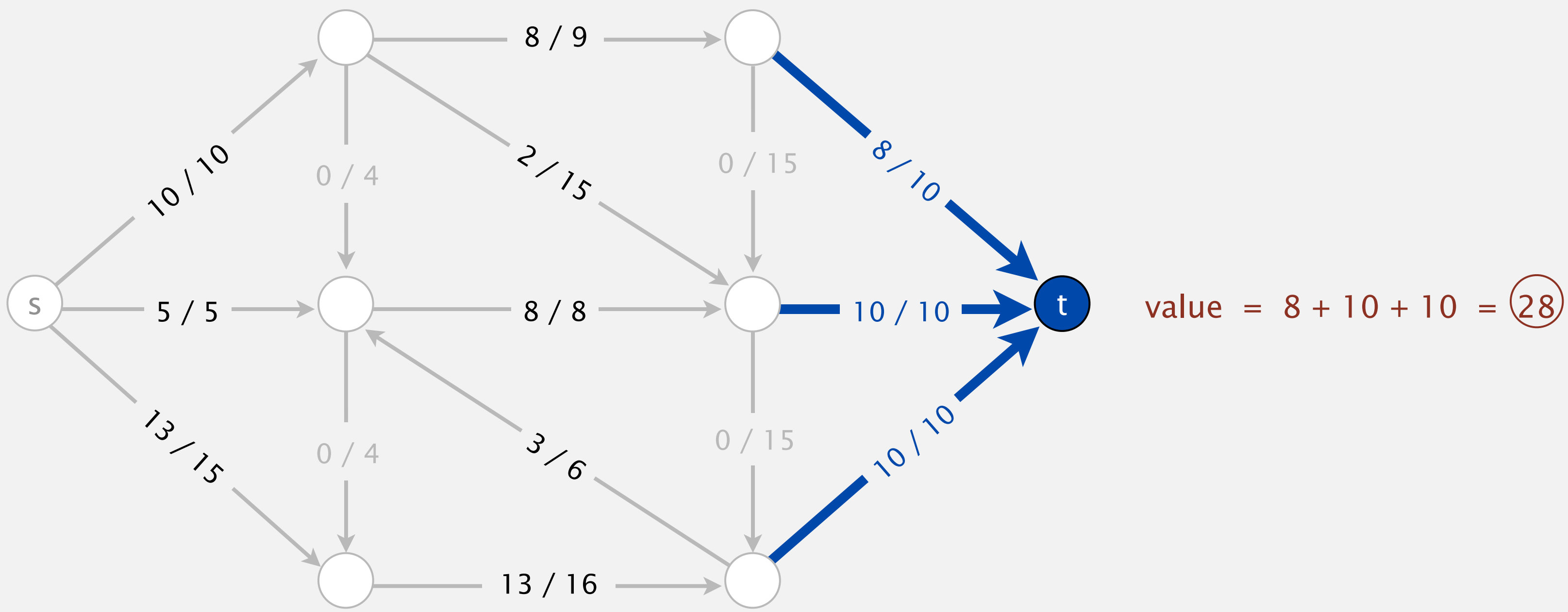
Maxflow problem

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

- Capacity constraint: $0 \leq$ edge's flow $\leq$ 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.

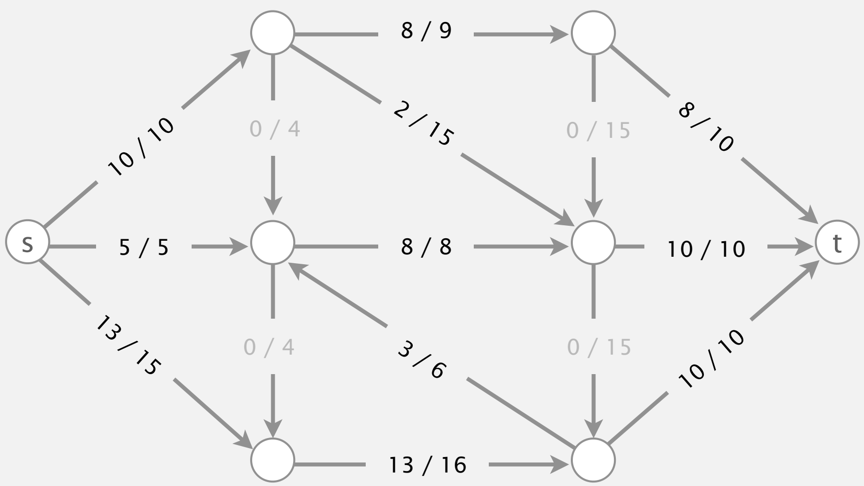


Summary

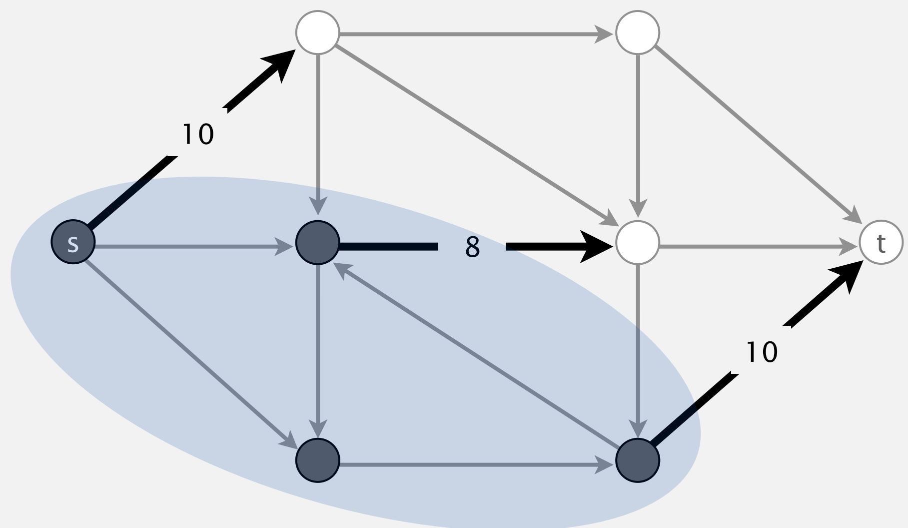
Input. A weighted digraph, source vertex s, and target vertex t.

Mincut problem. Find a cut of minimum capacity.

Maxflow problem. Find a flow of maximum value.



value of flow = 28



capacity of cut = 28

Remarkable fact. These two problems are dual!

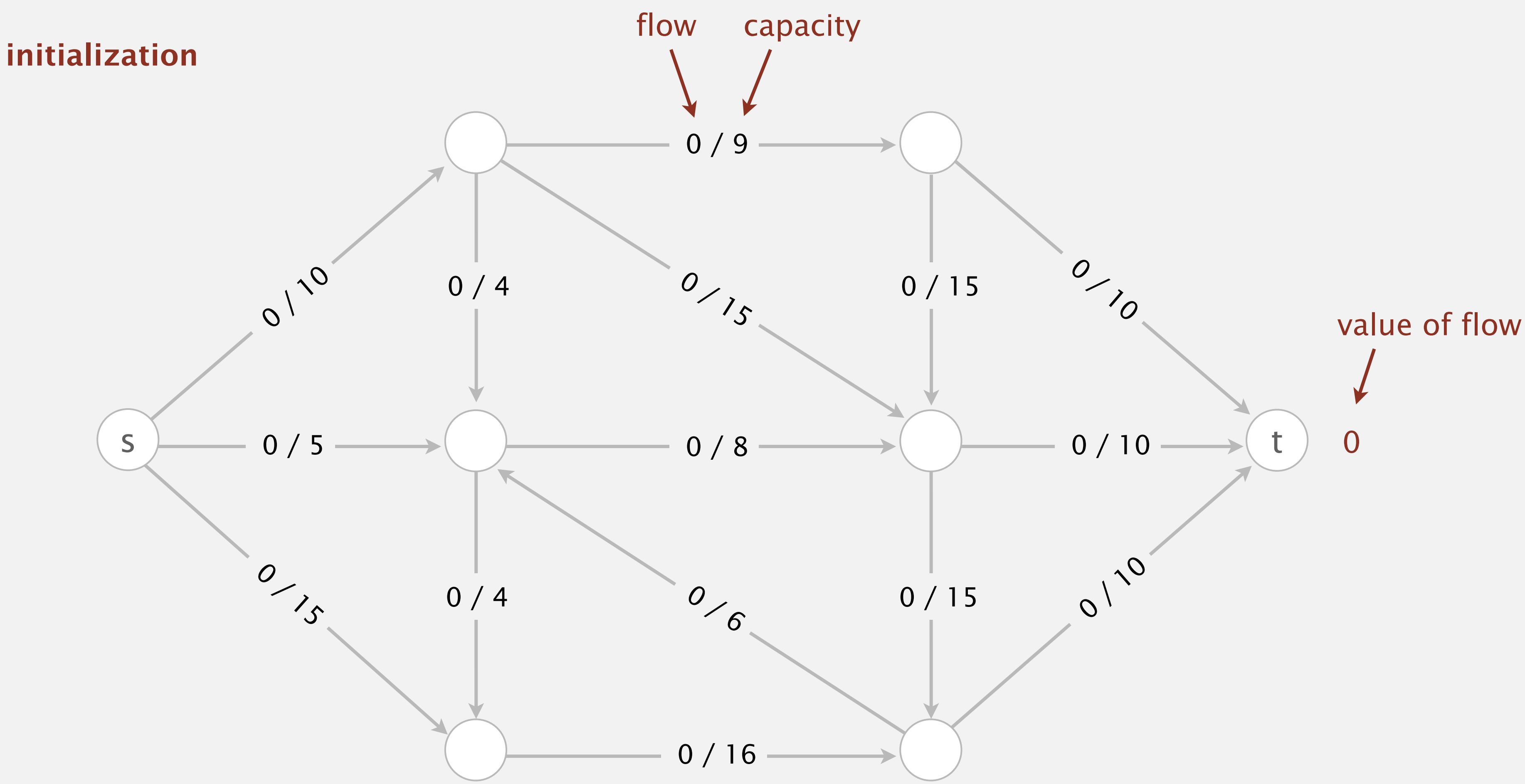
http://algs4.cs.princeton.edu

6.4 Maximum Flow

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

Ford-Fulkerson algorithm

Initialization. Start with 0 flow.

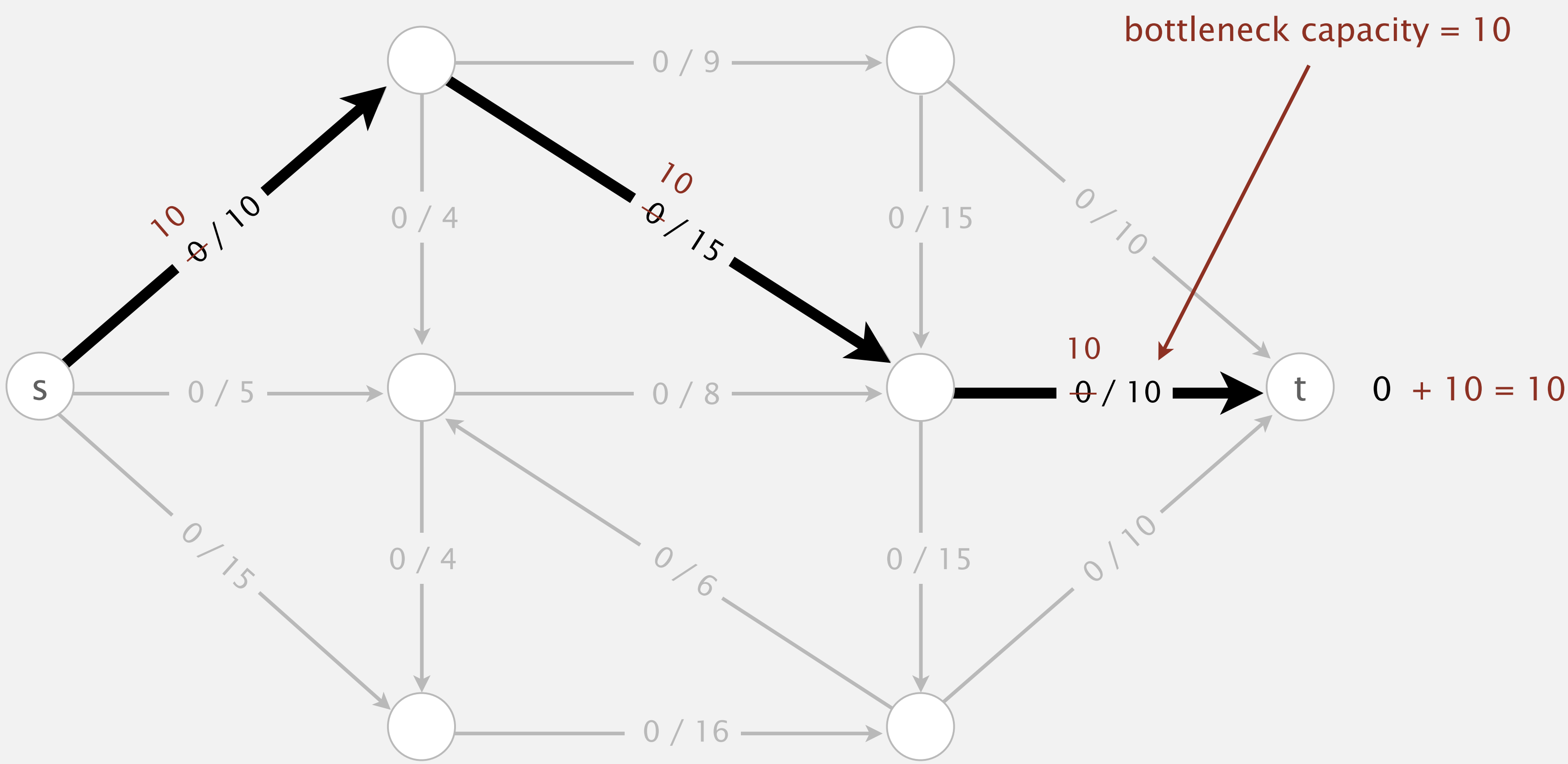


Idea: increase flow along augmenting paths

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

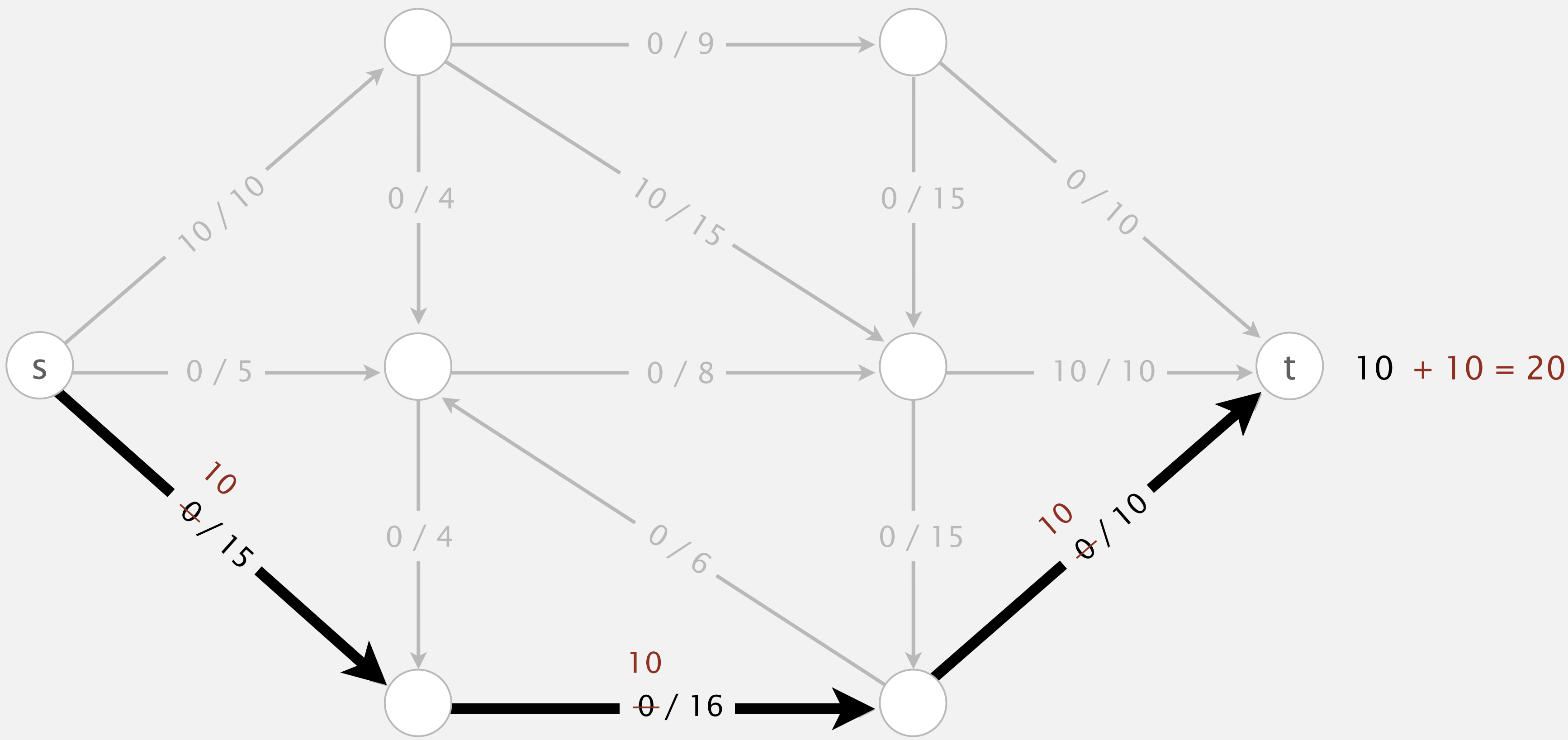


Idea: increase flow along augmenting paths

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

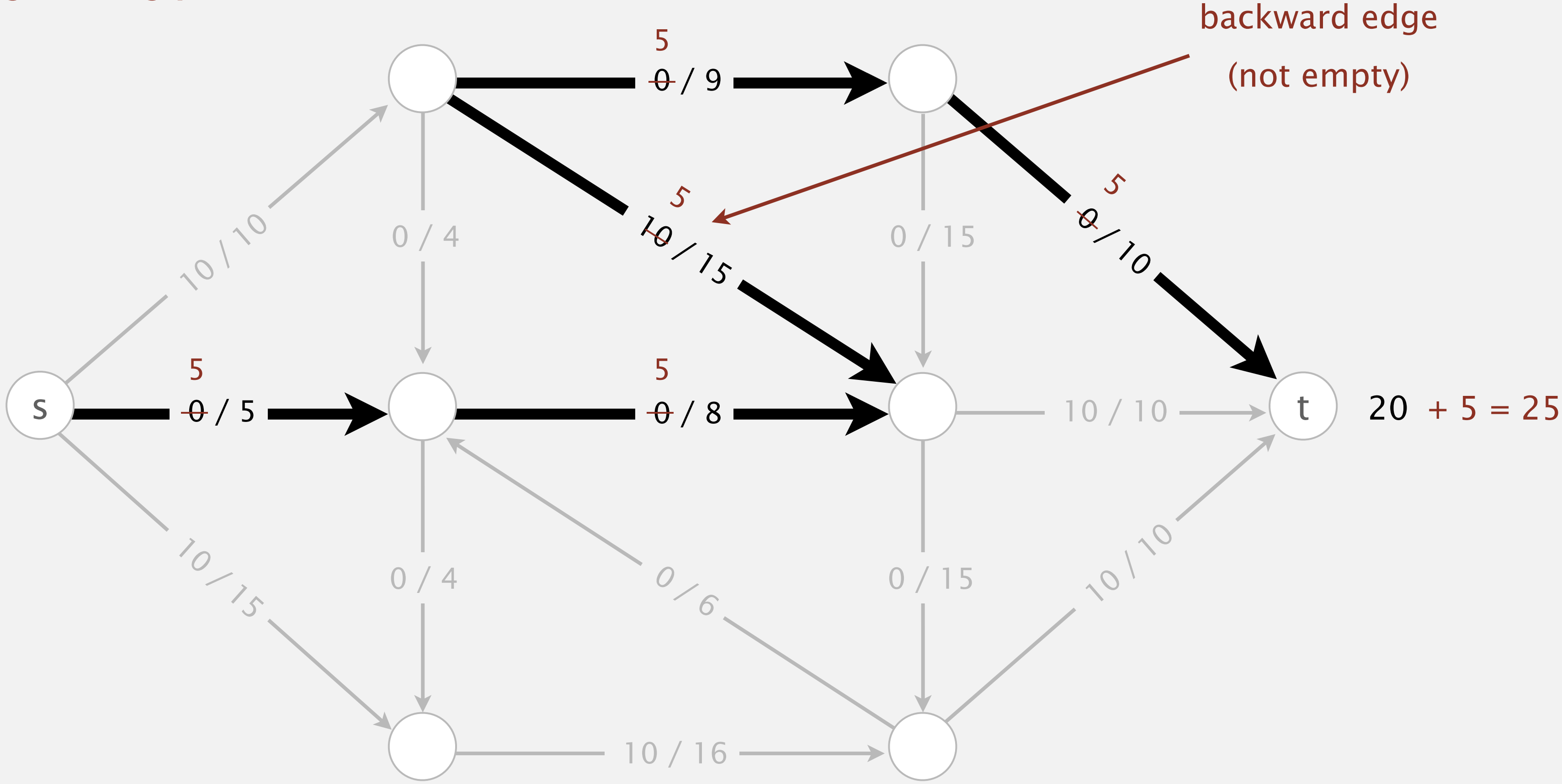


Idea: increase flow along augmenting paths

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).

3rd augmenting path

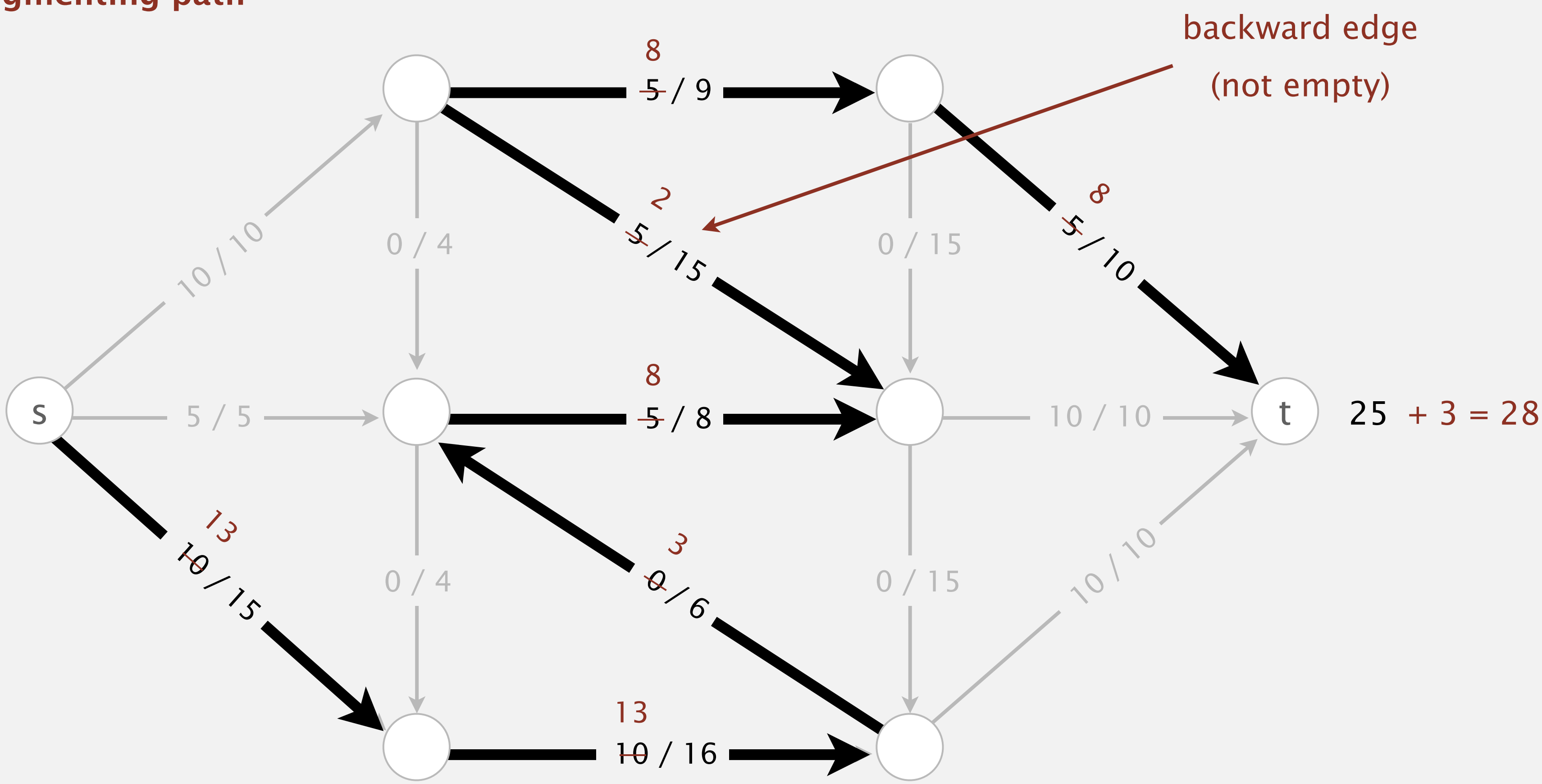


Idea: increase flow along augmenting paths

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).

4th augmenting path

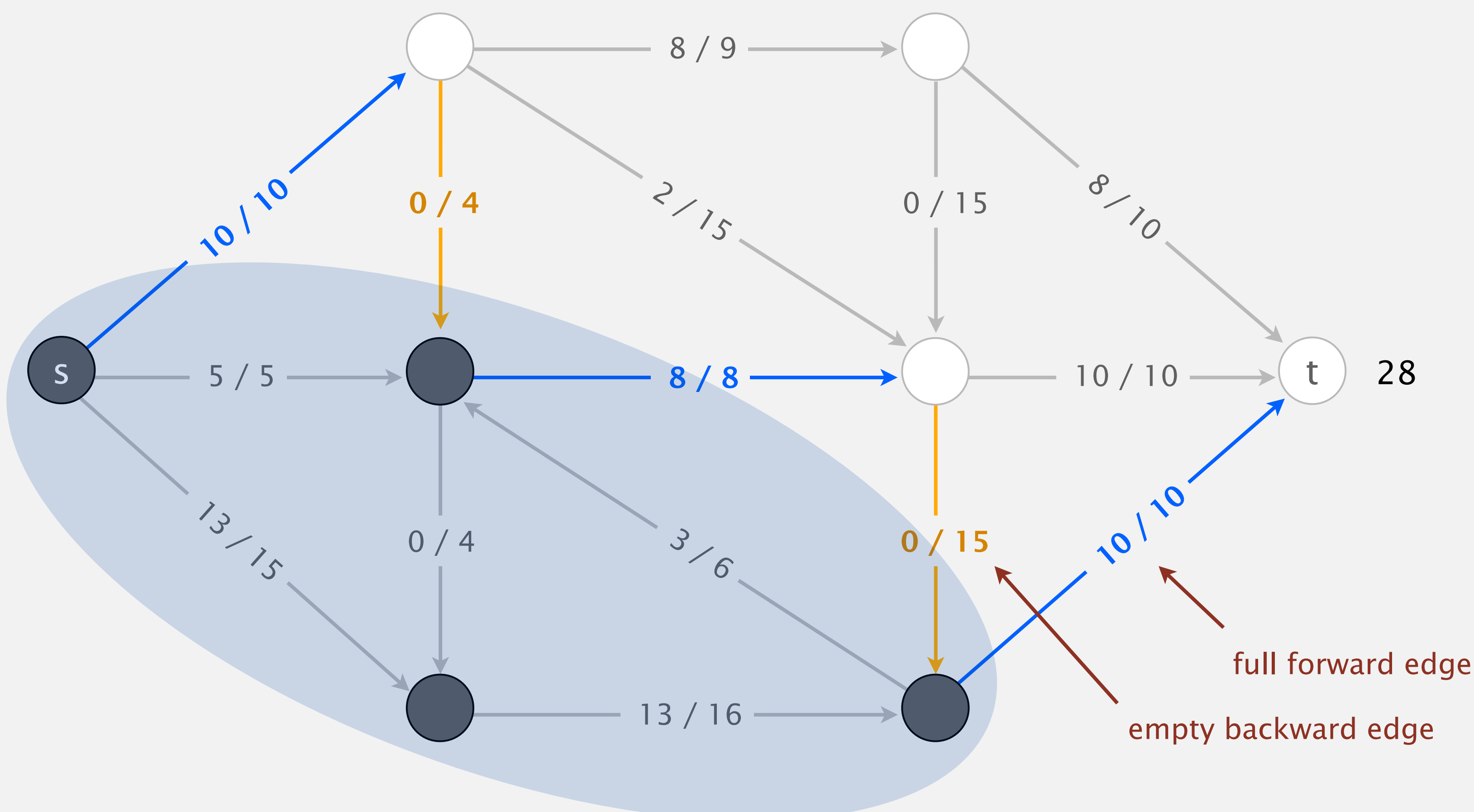


Idea: increase flow along augmenting paths

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

Algorithms

Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

6.4 Maximum Flow

- introduction
- Ford-Fulkerson algorithm
- **Java implementation**
- maxflow-mincut theorem
- running time analysis
- 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 find an augmenting path?
- If FF terminates, does it always compute a maxflow?
- How to compute a mincut?
- Does FF always terminate? If so, after how many augmentations?

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

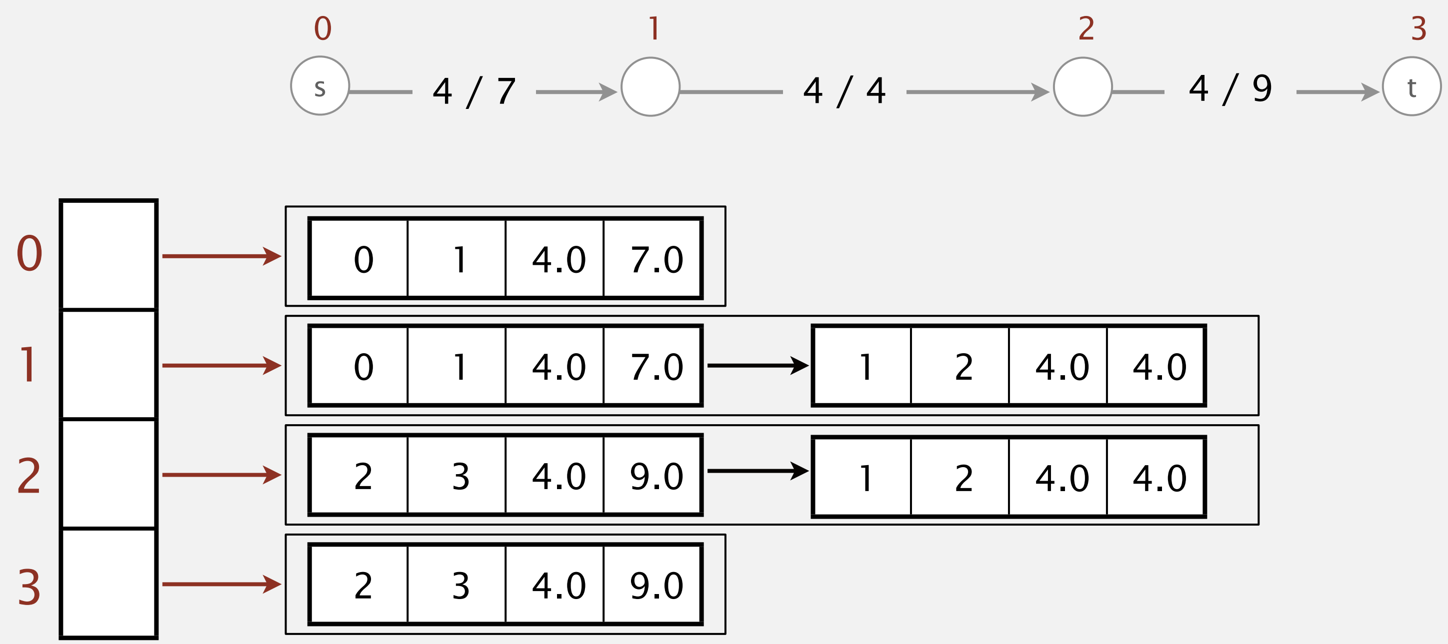
add forward edge

add backward edge

Flow network: Java implementation

Ford-Fulkerson inspired details

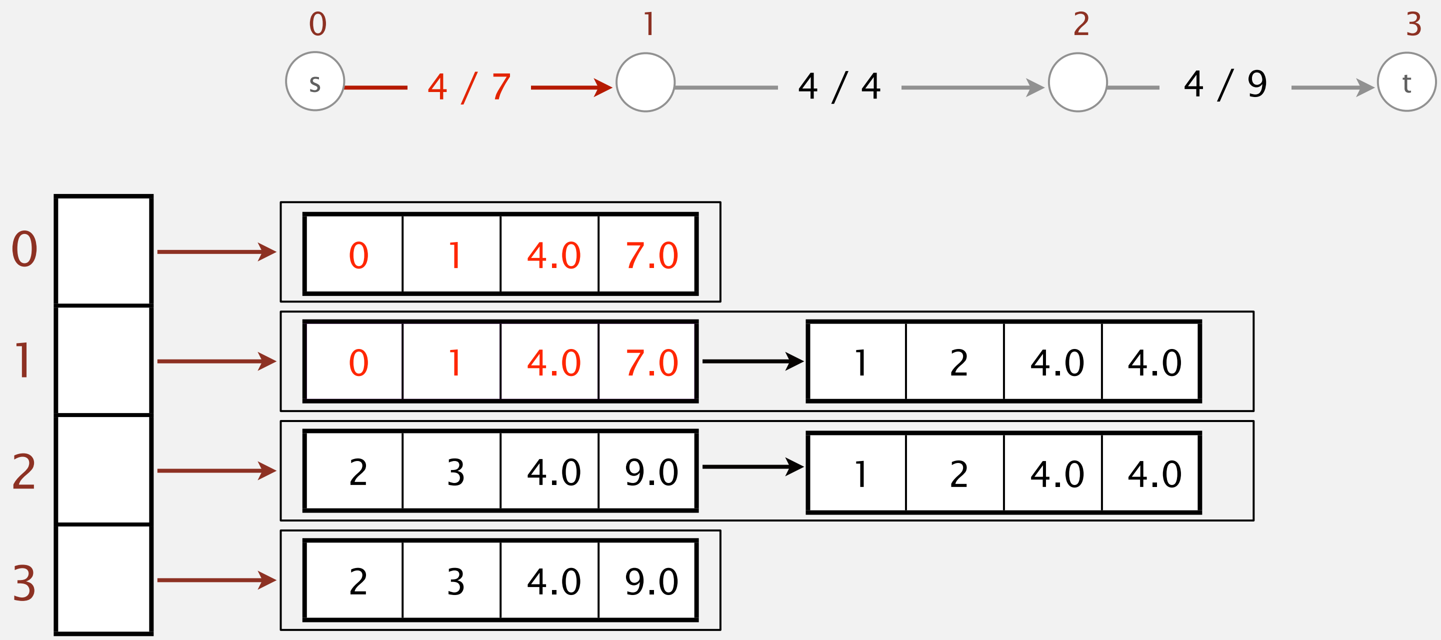
- Both forward and backward edges are provided.



Flow network: Java implementation

Ford-Fulkerson inspired details

- Both forward and backward edges are provided.
- Edges can report their **residual capacity**.

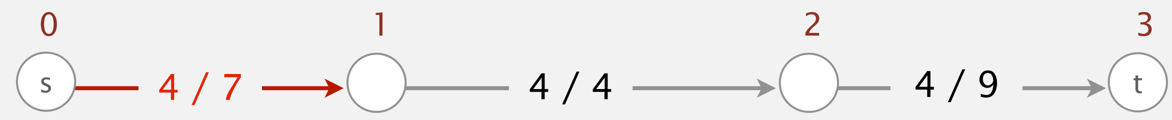


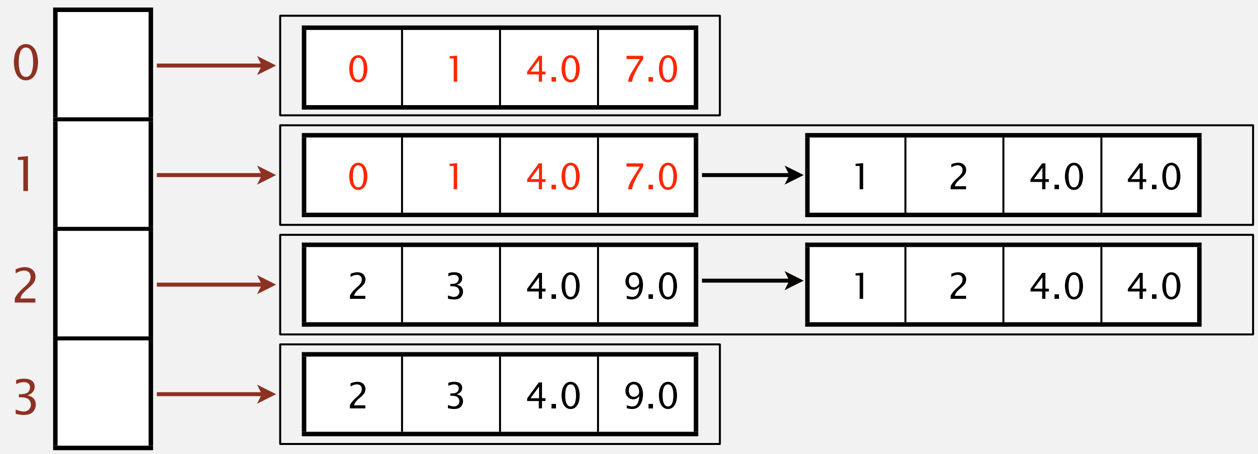

```
e.edgeFrom(): 0    e.edgeTo(): 1
e.residualCapacityTo(1): 3
e.residualCapacityTo(0): 4
```

Forward edge, residual capacity = capacity - flow

Backward edge, residual capacity = flow

Flow network: Java implementation



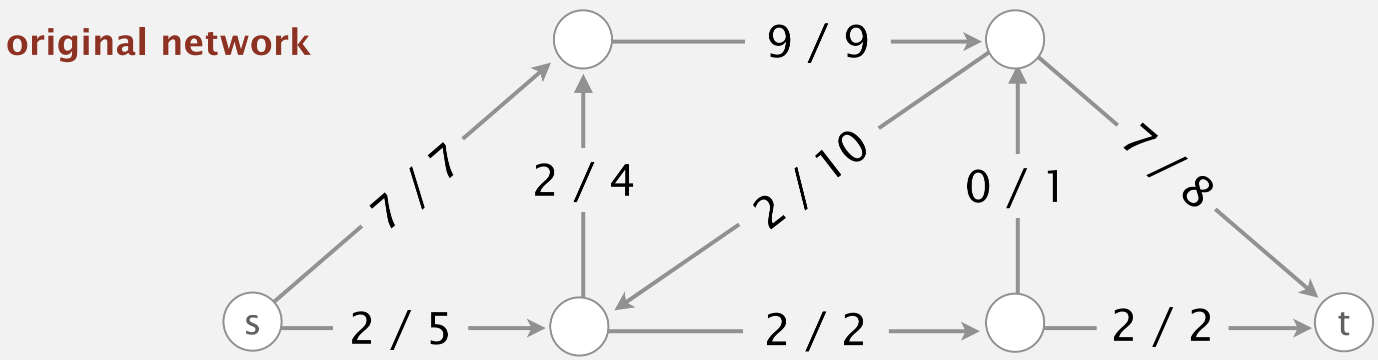


Residual Network

- Edge weighted digraph representing how much spare (used) capacity is available on a forward (backward) edge. If none, no edge.
- Represented ***IMPLICITLY*** by e.residualCapacityTo().

Residual Networks - Questions to ponder

Draw the residual network corresponding to the graph below.



What is the result of the code below?

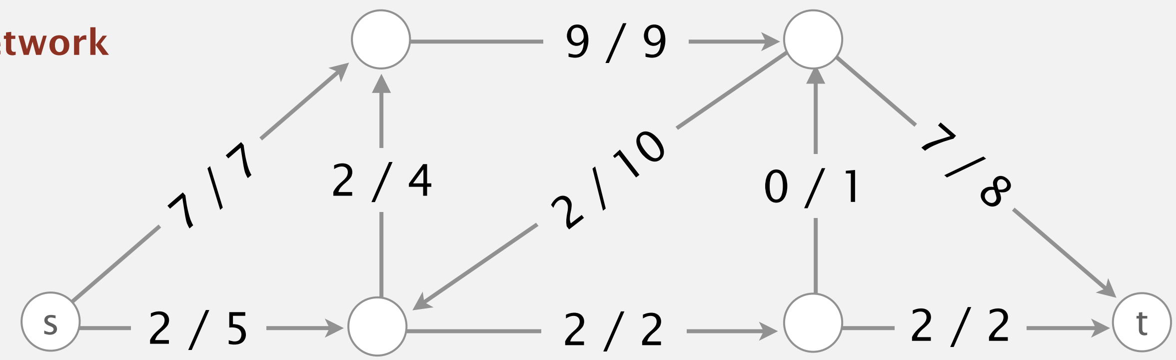
```
for (FlowEdge e : G.adj(s)) {
   int v = e.from(); int w = e.to();
   System.out.println(e.residualCapacityTo(w));
}
```

How can you find an augmenting path using the residual network graph?

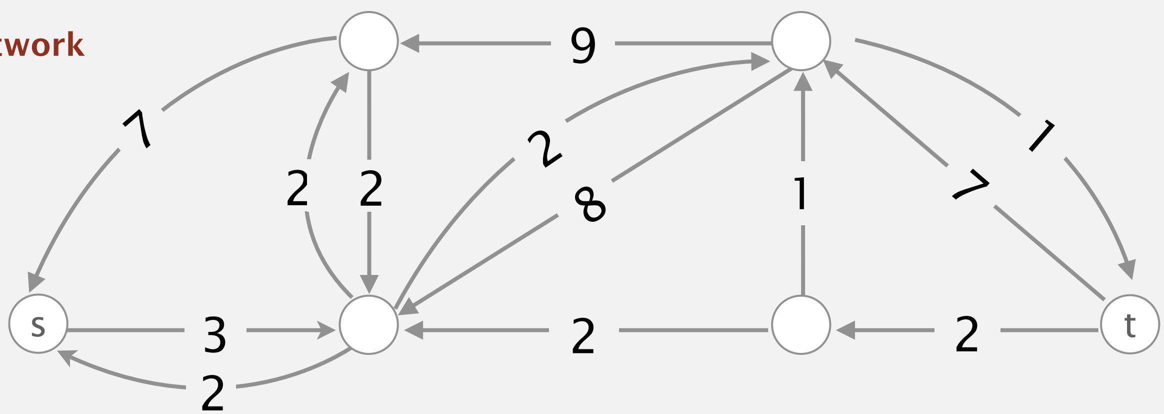
Residual Networks

Draw the residual network corresponding to the graph below.

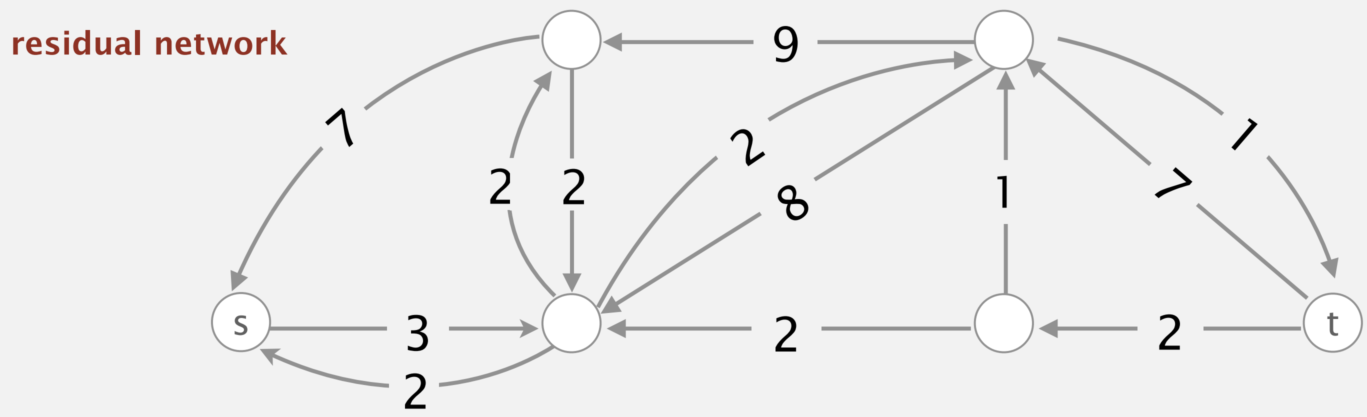
original network



residual network



Residual Network



What is the result of the code below (s is the source vertex)?

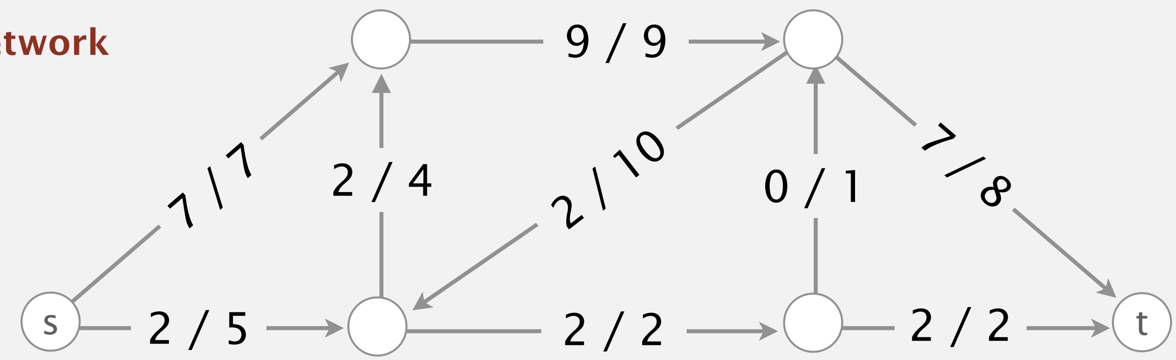
```
for (FlowEdge e : G.adj(s)) {
    int v = e.from(); int w = e.to();
    System.out.println(e.residualCapacityTo(w));
}
```

- The two FlowEdges adjacent to e have residual capacity of 0 and 3 when examined in the forward direction.
  - Prints 0 on a new line, and 3 on a new line.

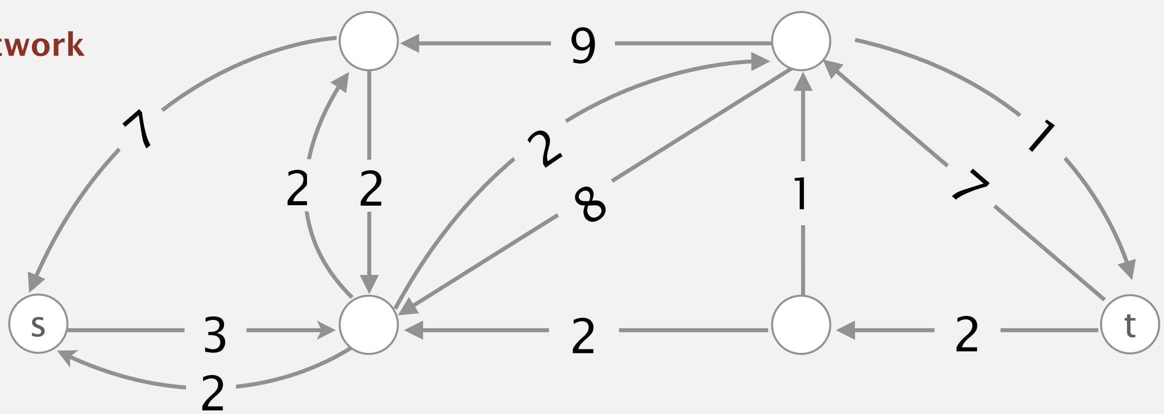
Residual Networks

Draw the residual network corresponding to the graph below.

original network



residual network



How can you find an augmenting path using the residual network graph?

- Find any path from s to t. Edge only exists if weight > 0.

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);
  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) {
          marked[w] = true;
          queue.enqueue(w);
          edgeTo[w] = e;
      }
    }
    //how do we know if a path exists to t?
    return marked[t];
  }
}
```

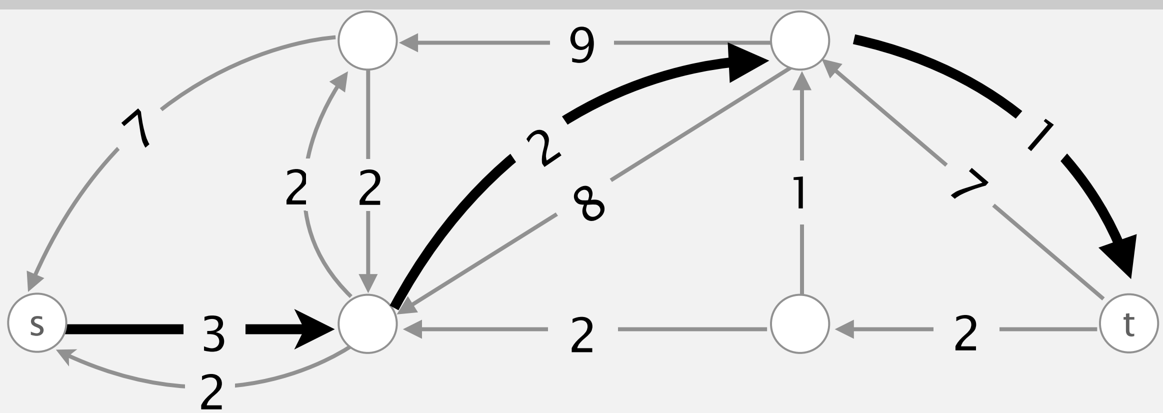
Ford-Fulkerson: Java implementation

```
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;
      }
   }
   ...
```

creates edgeTo[]

walk backwards from t and compute bottleneck capacity

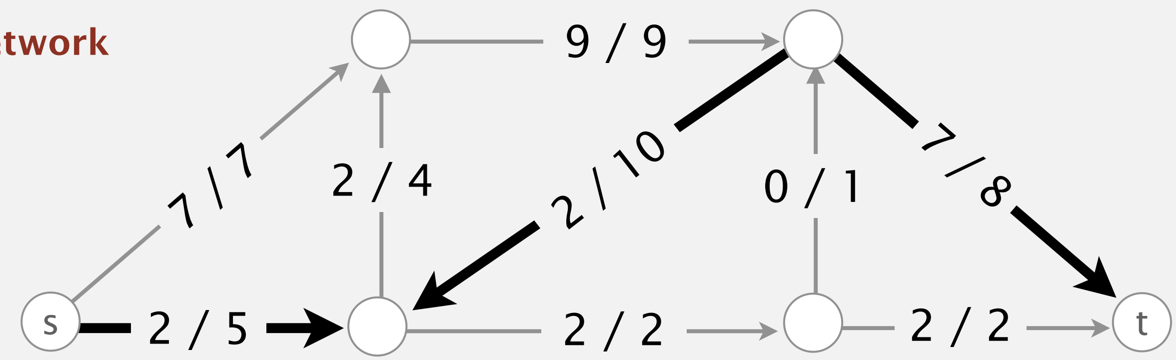
walk backwards from t and augment flow



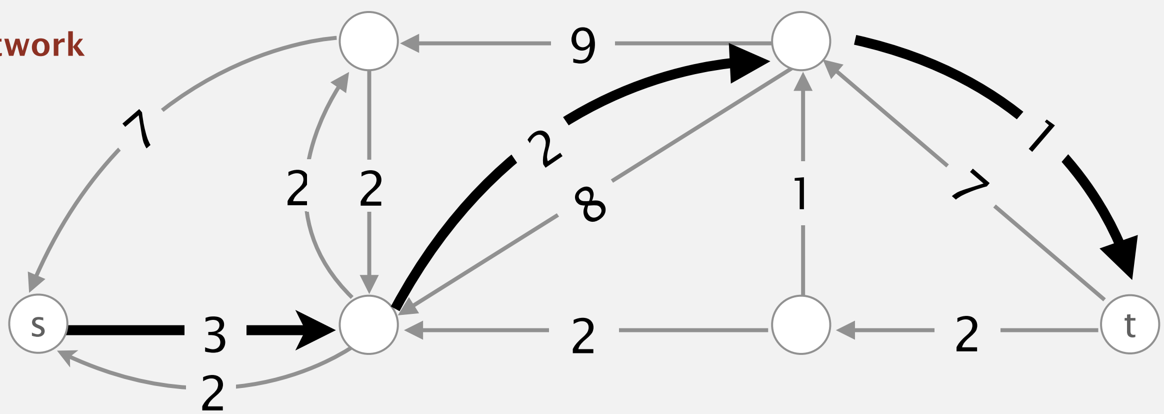
Residual Networks

Any path in residual network is an augmenting path in original network

original network



residual network



http://algs4.cs.princeton.edu

6.4 Maximum Flow

- introduction
- Ford-Fulkerson algorithm
- Java implementation
- maxflow-mincut theorem
- running time analysis
- 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 find an augmenting path? BFS (or other).
- If FF terminates, does it always compute a maxflow?
- How to compute a mincut?
- Does FF always terminate? If so, after how many augmentations?

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 an st-cut 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.

Overall Goal:

- Prove that i ⇒ ii. [Trivial]
- Prove that ii ⇒ iii. [Trivial]
- Prove that iii ⇒ i. [A little work]

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.

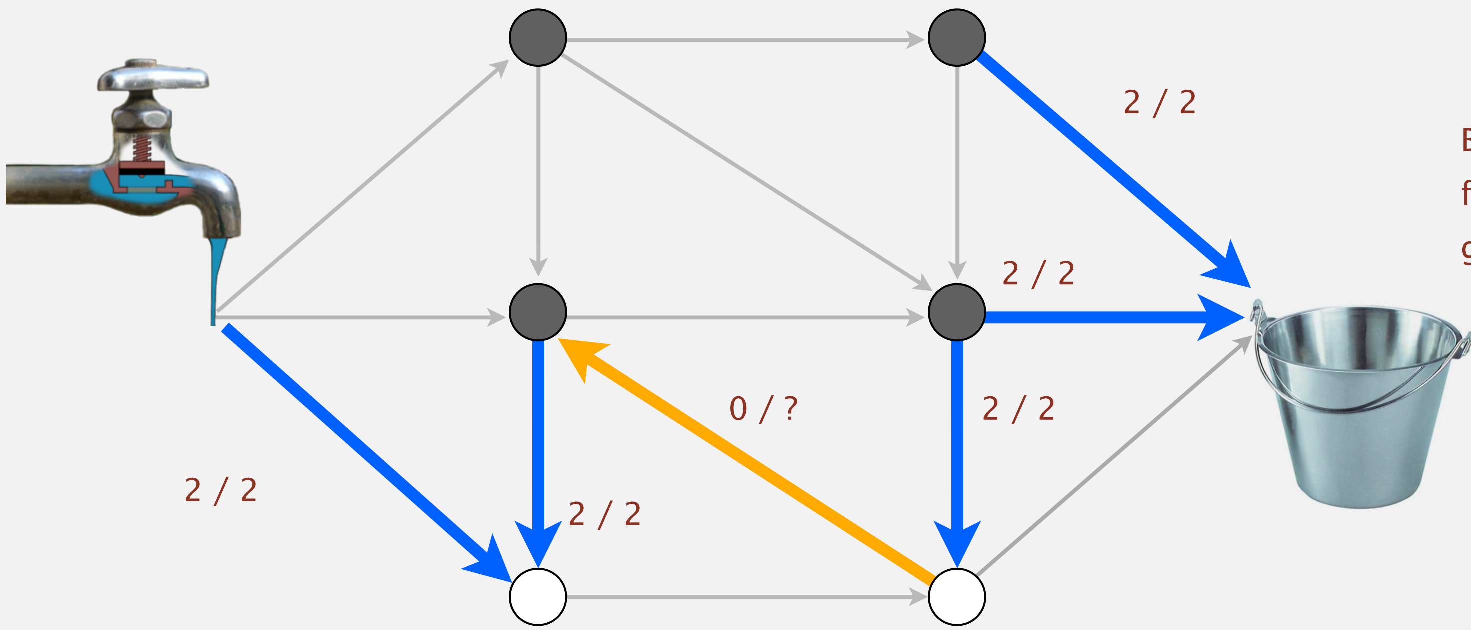
Pf. The following three conditions are equivalent for any flow f:

i. There exists an st-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 ⇒ ii]: Trivial by analogy with water flow [see book for technical proof].



Bucket can fill at no faster than 10 gallons per second.

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 an st-cut cut whose capacity equals the value of the flow f.

ii. f is a maxflow.

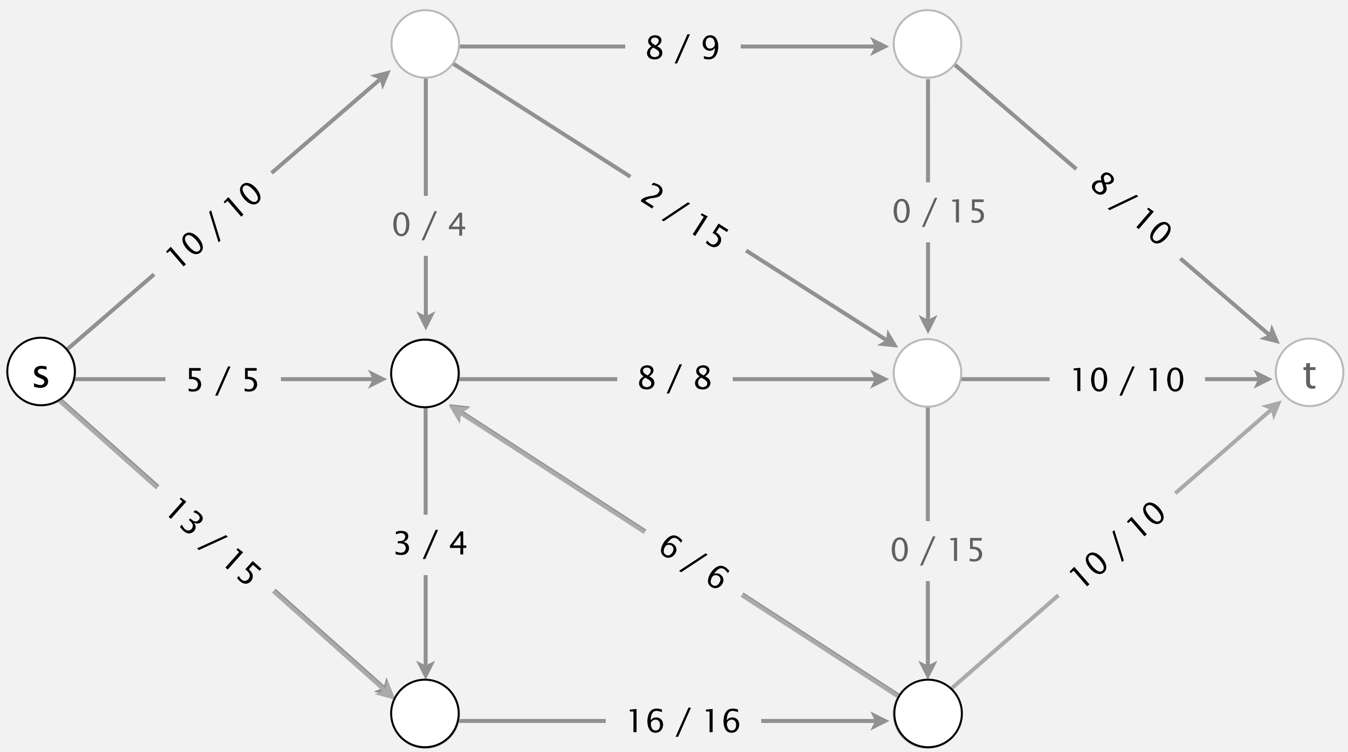
iii. There is no augmenting path with respect to f.

[ii ⇒ iii] Trivial, 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.

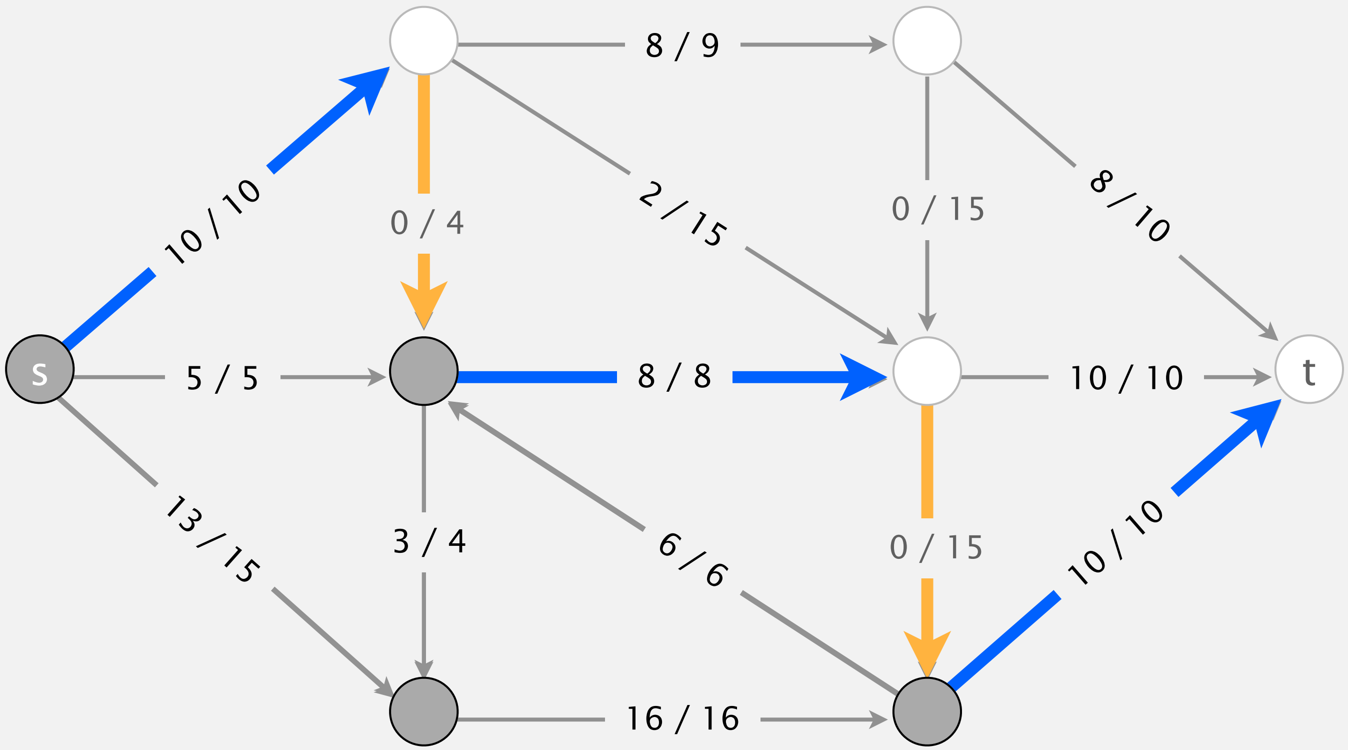
Computing a mincut from a maxflow

Find an augmenting path.



Computing a mincut from a maxflow

- We've found a cut whose capacity equals the value of the flow.



Find an augmenting path.

- Couldn't find an augmenting path (some edges block us).
  - These edges form a cut.
  - There is no backward flow from t to s.
  - All edges from s to t are full.

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.

Overall Goal:

- Prove that i ⇒ ii. [Analogy with water, see book for technical proof]
- Prove that ii ⇒ iii. [Trivial by proving contrapositive]
- Prove that iii ⇒ i. [Constructive proof, see book for technical proof]

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**

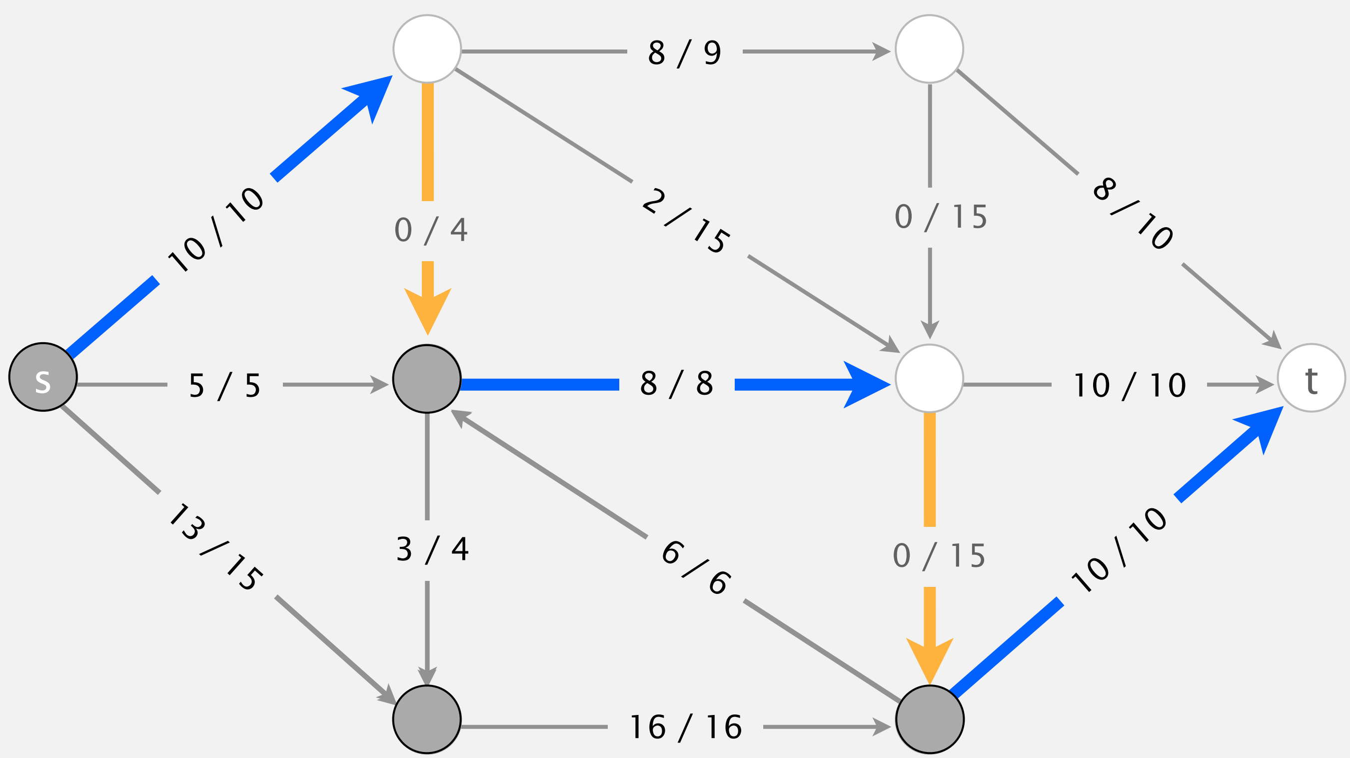
Questions.

- How to find an augmenting path? BFS (or other).
- If FF terminates, does it always compute a maxflow?
  - Yes, because non-existence of augmenting path implies max flow.
  - iii ⇒ i ⇒ ii
- How to compute a mincut?
- Does FF always terminate? If so, after how many augmentations?

Computing a mincut from a maxflow

Find an augmenting path.

- Couldn't find an augmenting path (some edges block us).
  - These edges form a cut.
  - There is no backward flow from t to s.
  - All edges from s to t are full.
- We've found a cut whose capacity equals the value of the flow.



Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

http://algs4.cs.princeton.edu

6.4 MAXIMUM FLOW

- introduction
- Ford-Fulkerson algorithm
- Java implementation
- maxflow-mincut theorem
- **running time analysis**
- 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 find an augmenting path? BFS (or other).
- If FF terminates, does it always compute a maxflow? Yes. ✔
- How to compute a mincut? Easy. ✔
- 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.

important for some applications (stay tuned)

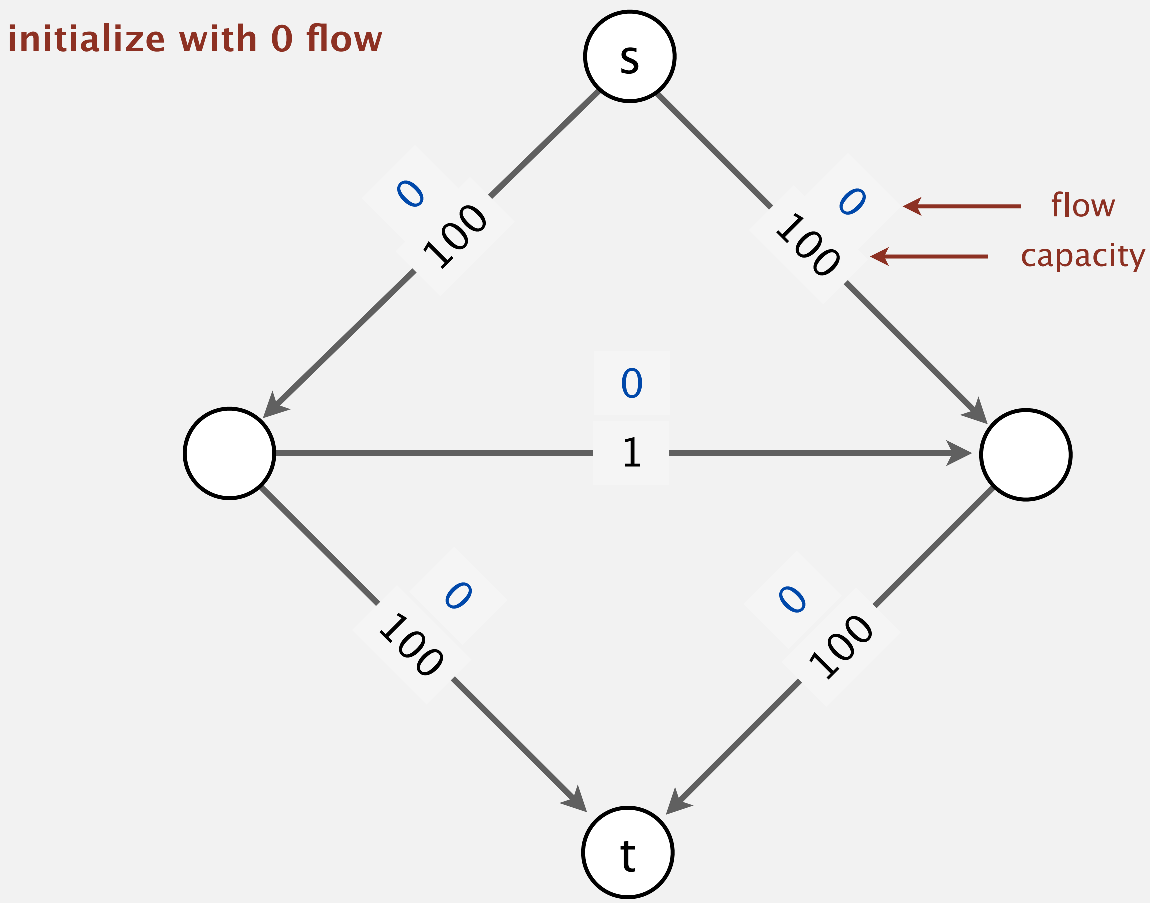
and FF finds one!

Integrality theorem. There exists an integer-valued maxflow.

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

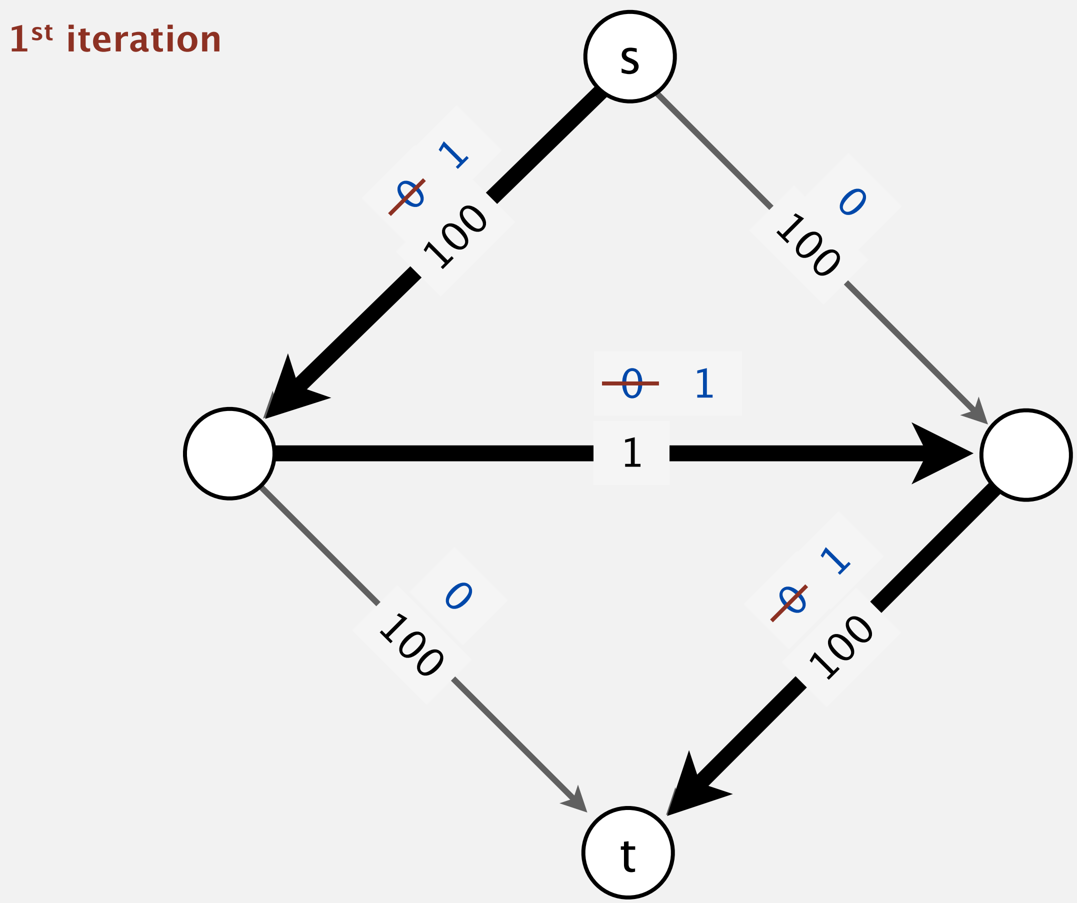
Bad case for Ford-Fulkerson

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



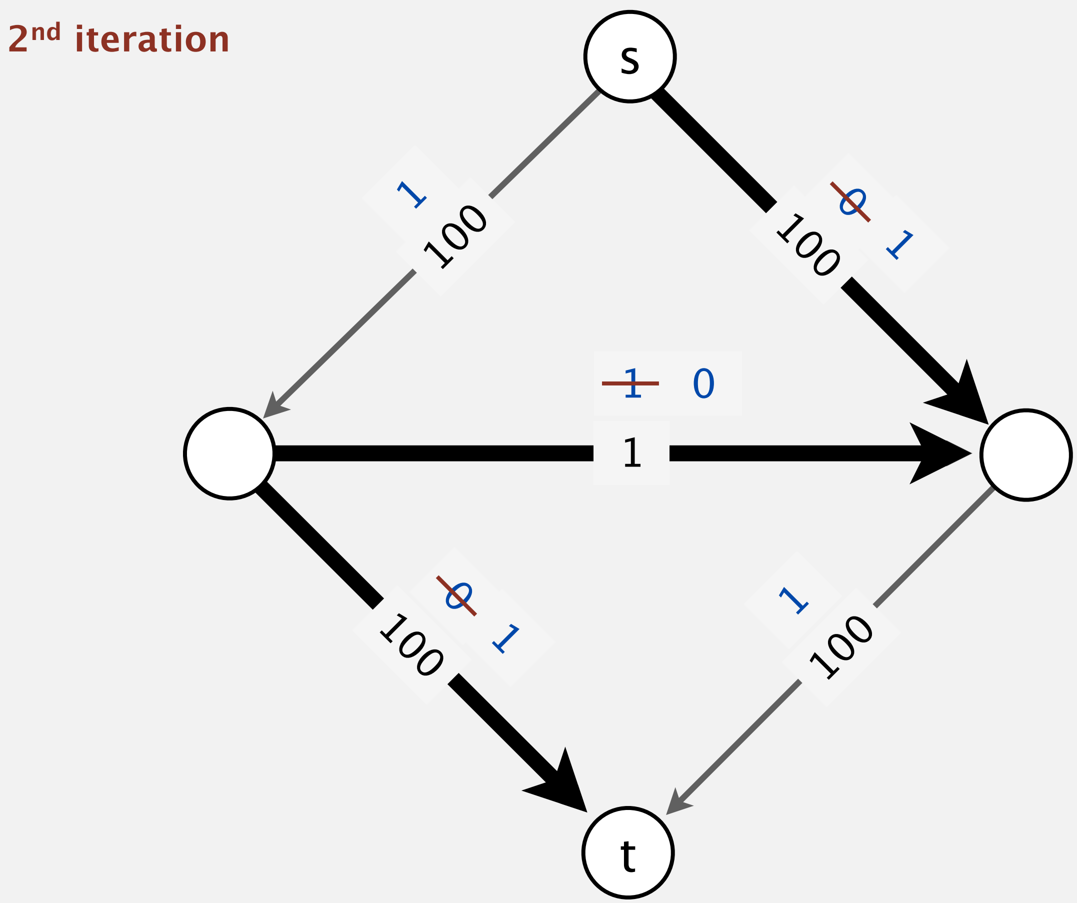
Bad case for Ford-Fulkerson

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



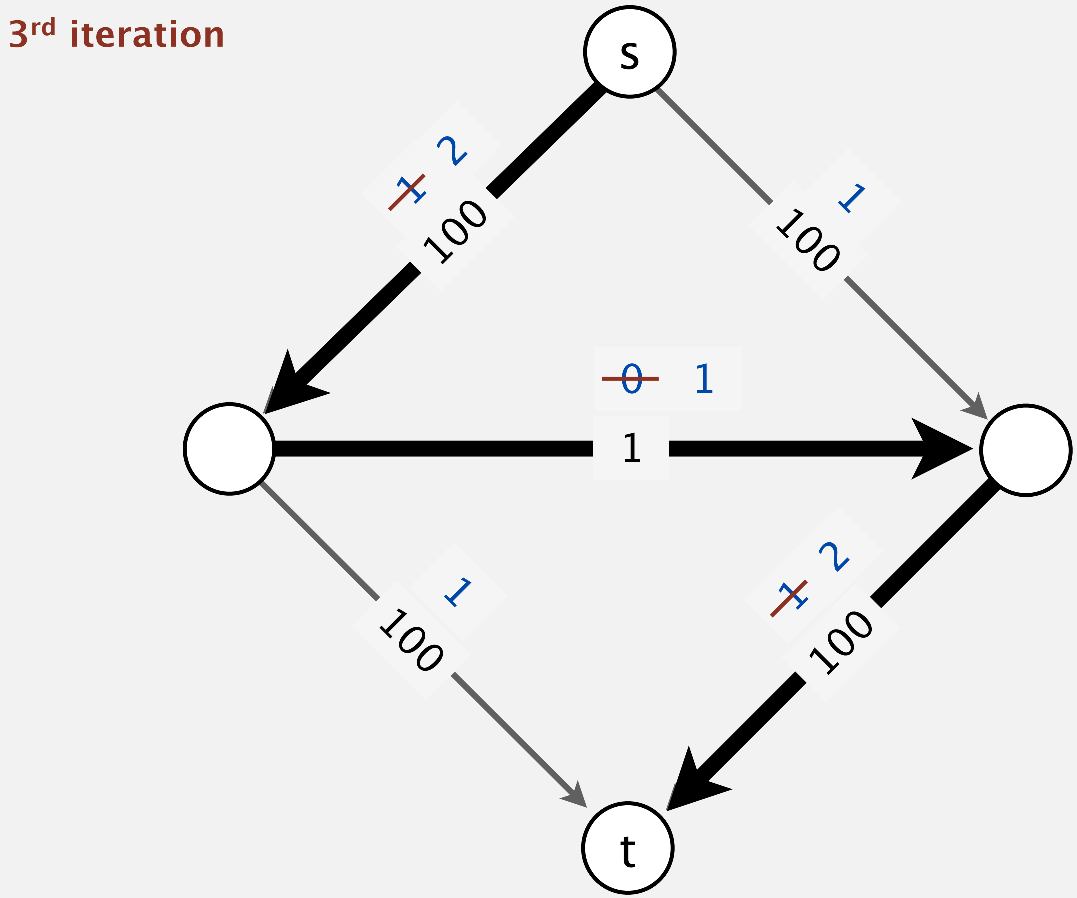
Bad case for Ford-Fulkerson

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

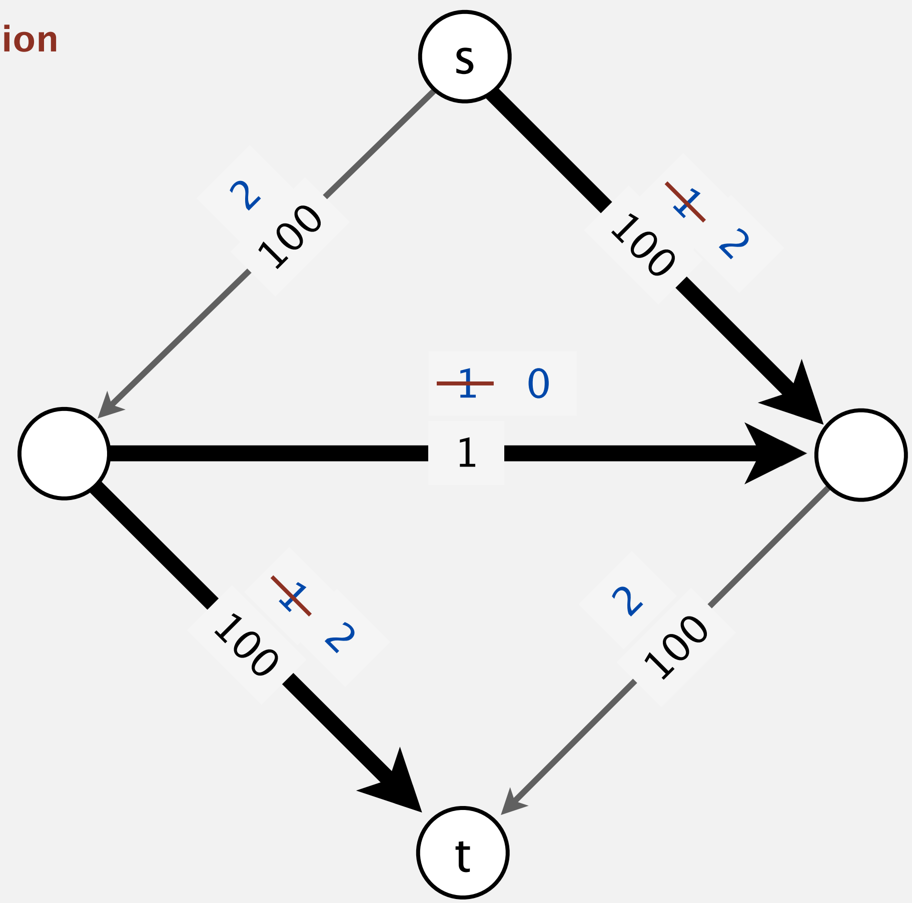
Bad news. Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maxflow.



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.

4th iteration



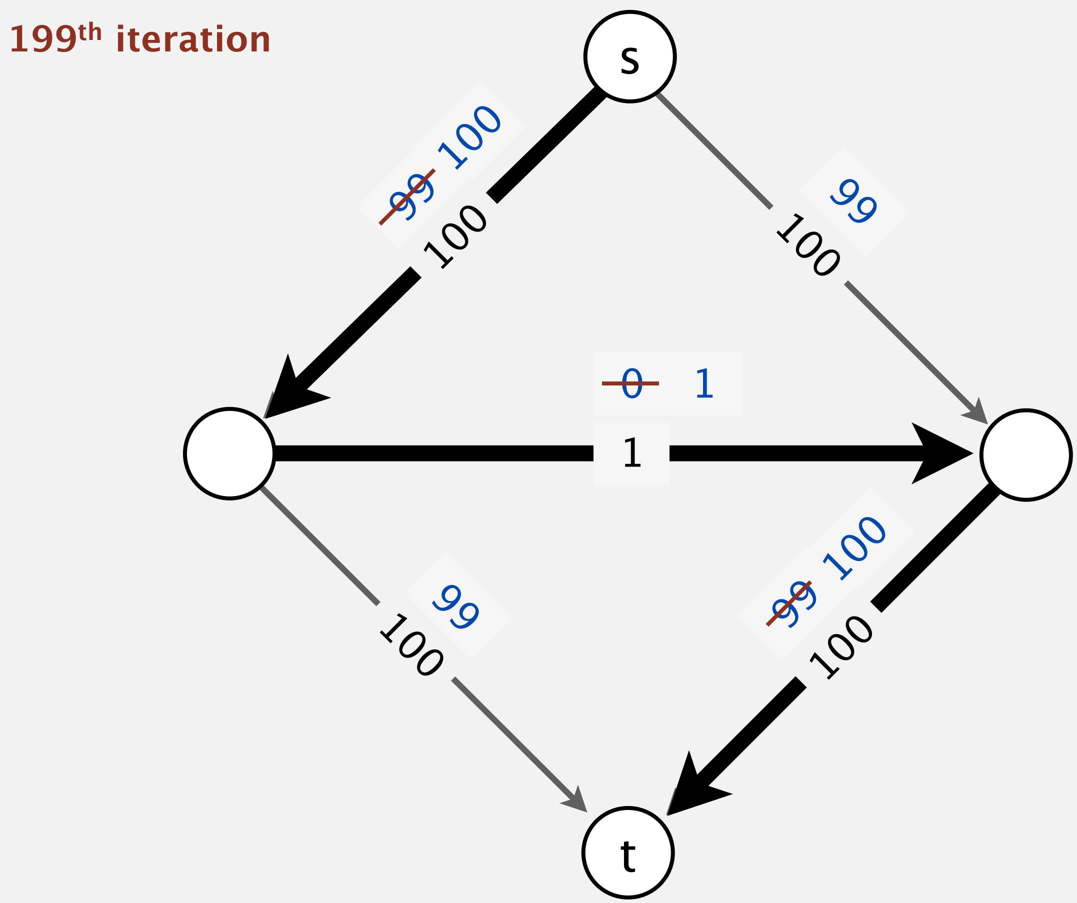
Bad case for Ford-Fulkerson

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

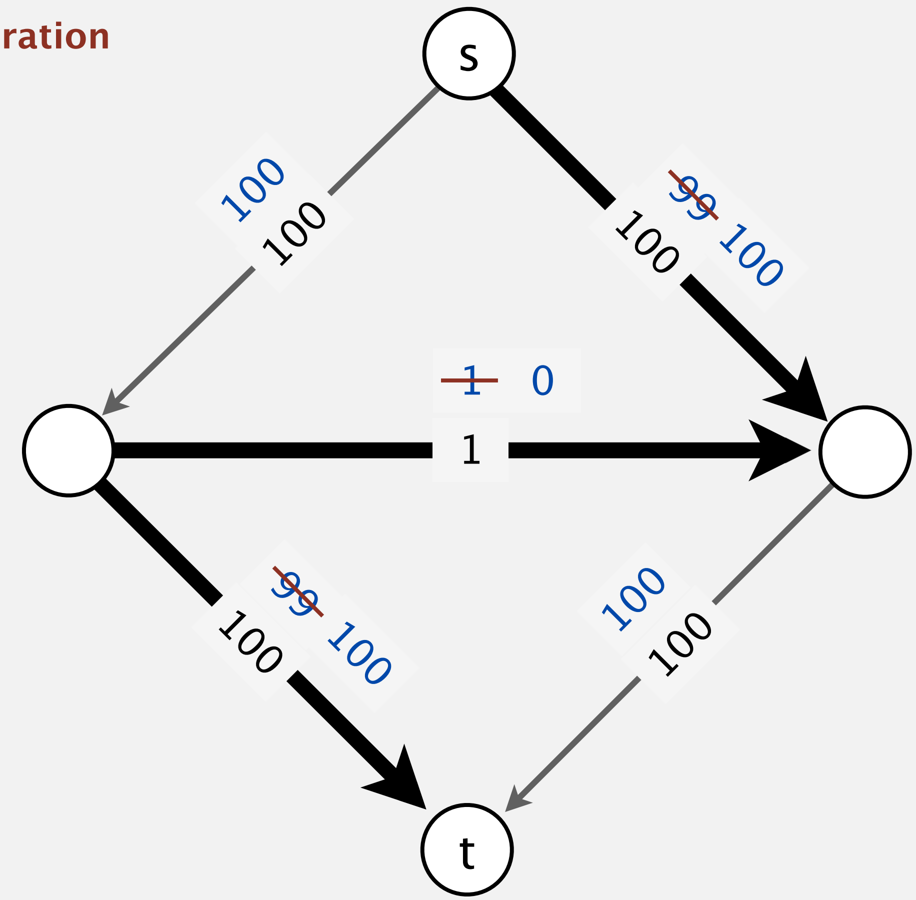
Bad news. Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maxflow.



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.

200th iteration

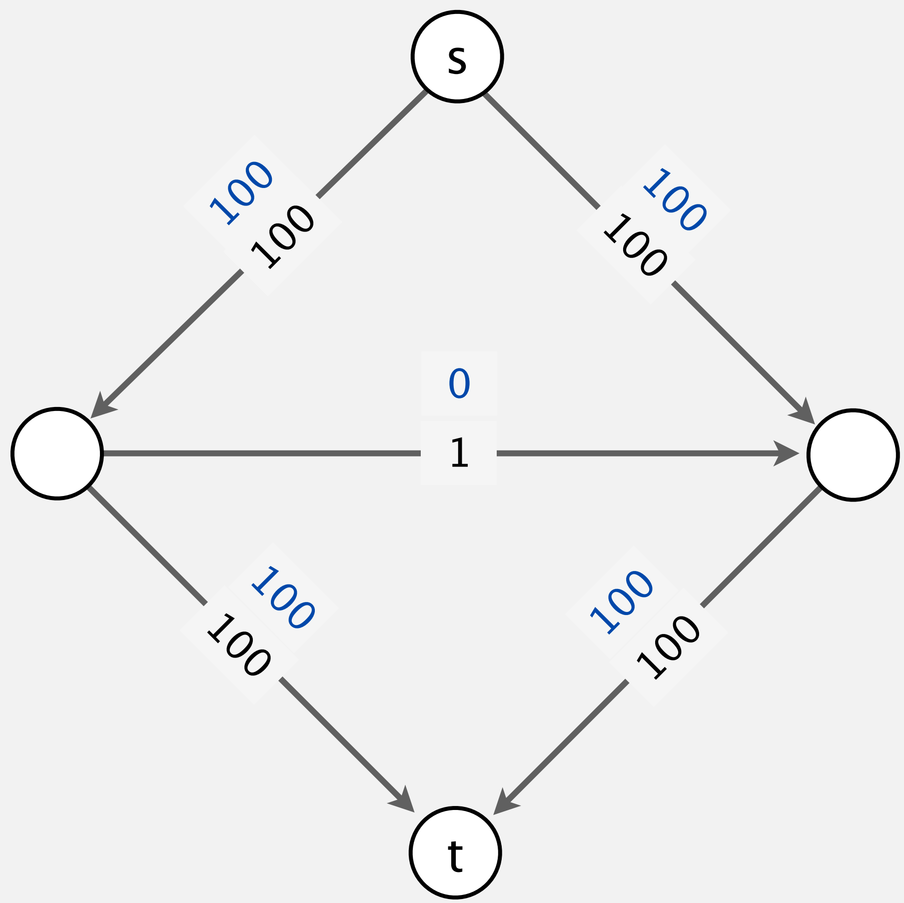


Bad case for Ford-Fulkerson

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?

FF performance depends on choice of augmenting paths.

augmenting path	number of paths	implementation
shortest path	≤ ½ E V	queue (BFS)
fattest path	≤ E ln(E U)	priority queue
random path	≤ E U	randomized queue
DFS path	≤ E U	stack (DFS)

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

augmenting path
with fewest
number of edges

shortest path

augmenting path
with maximum
bottleneck capacity

fattest path

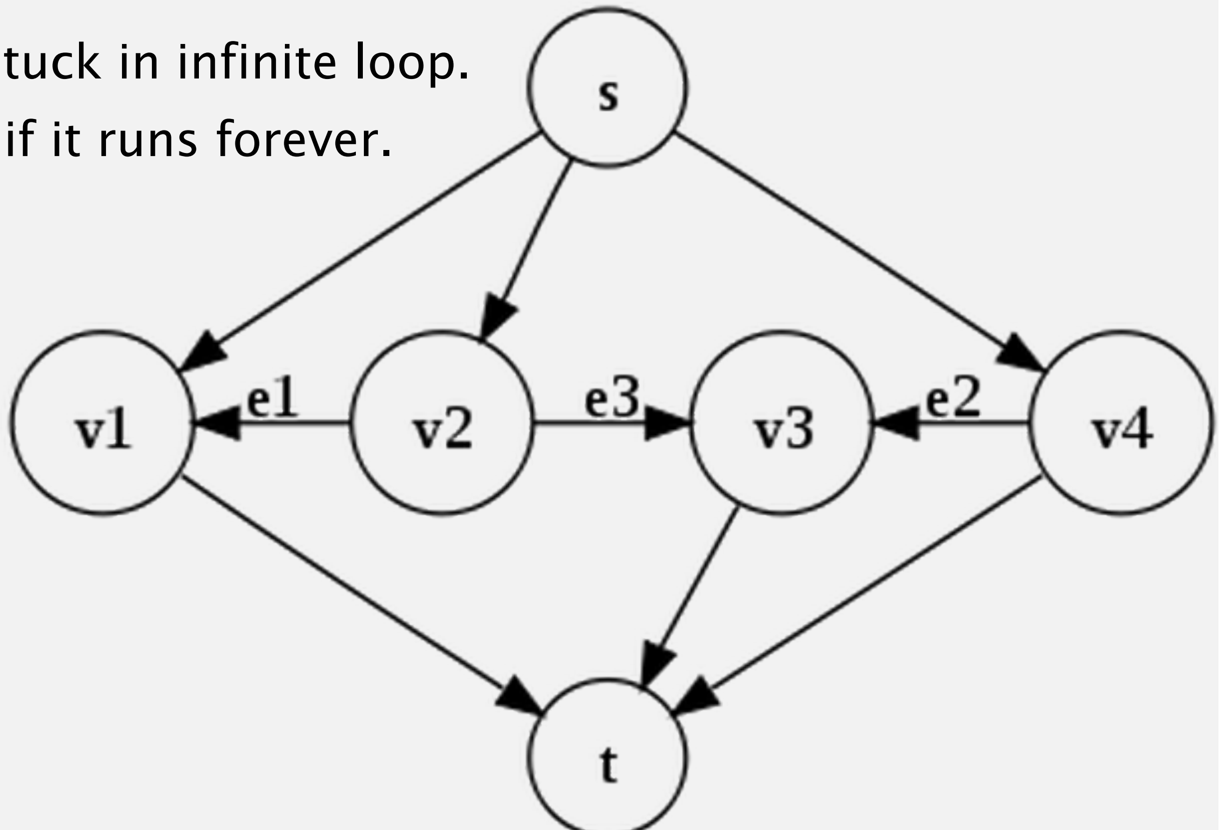
Non-integer weights (beyond scope of course)

If:

- e1 = 1
- e2 = (sqrt(5) - 1)/2
- e3 = 1
- Other edges of weight 2 or greater

Can show:

- Ford Fulkerson can get stuck in infinite loop.
- Does not converge even if it runs forever.



http://algs4.cs.princeton.edu

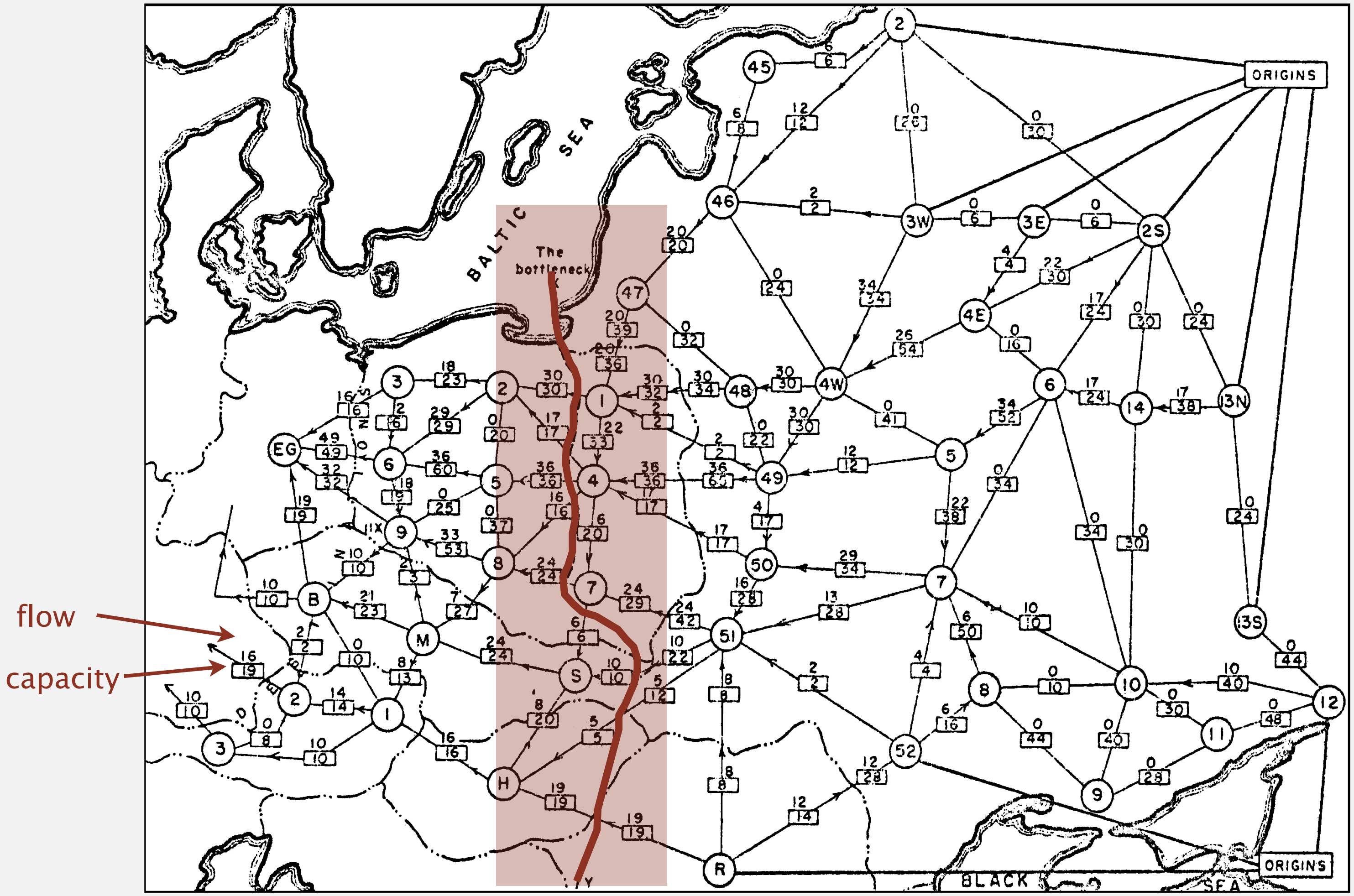
6.4 Maximum Flow

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

Mincut application (RAND Corporation - 1950s)

"Free world" goals. Understand peak Soviet supply rate.

Cut supplies (if cold war turns into real war).

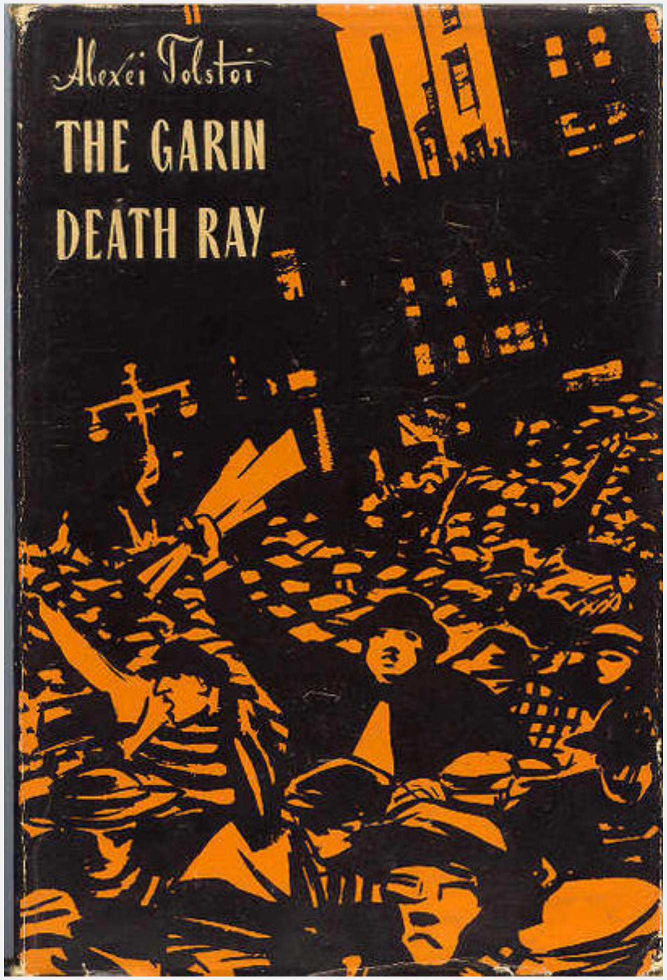


rail network connecting Soviet Union with Eastern European countries
(map declassified by Pentagon in 1999)

Maxflow application (1950s)

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

- Originally studied by writer Alexei Tolstoi in the 1930s (ad hoc approach).
- Later considered by Ford & Fulkerson via min cut approach.

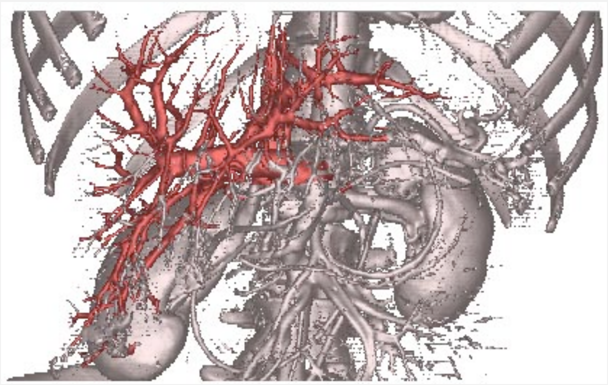


Гиперболоид инженера Гарина

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

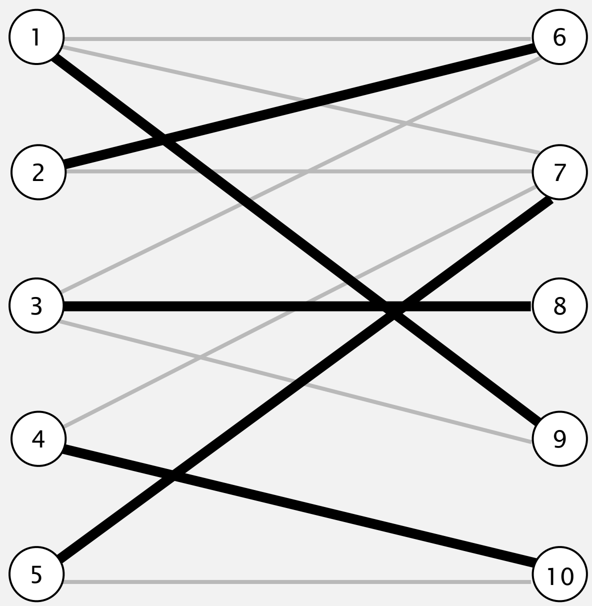
Bipartite matching problem

Given a bipartite graph, find a perfect matching.

perfect matching (solution)

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

bipartite graph



N students **N companies**

bipartite matching problem

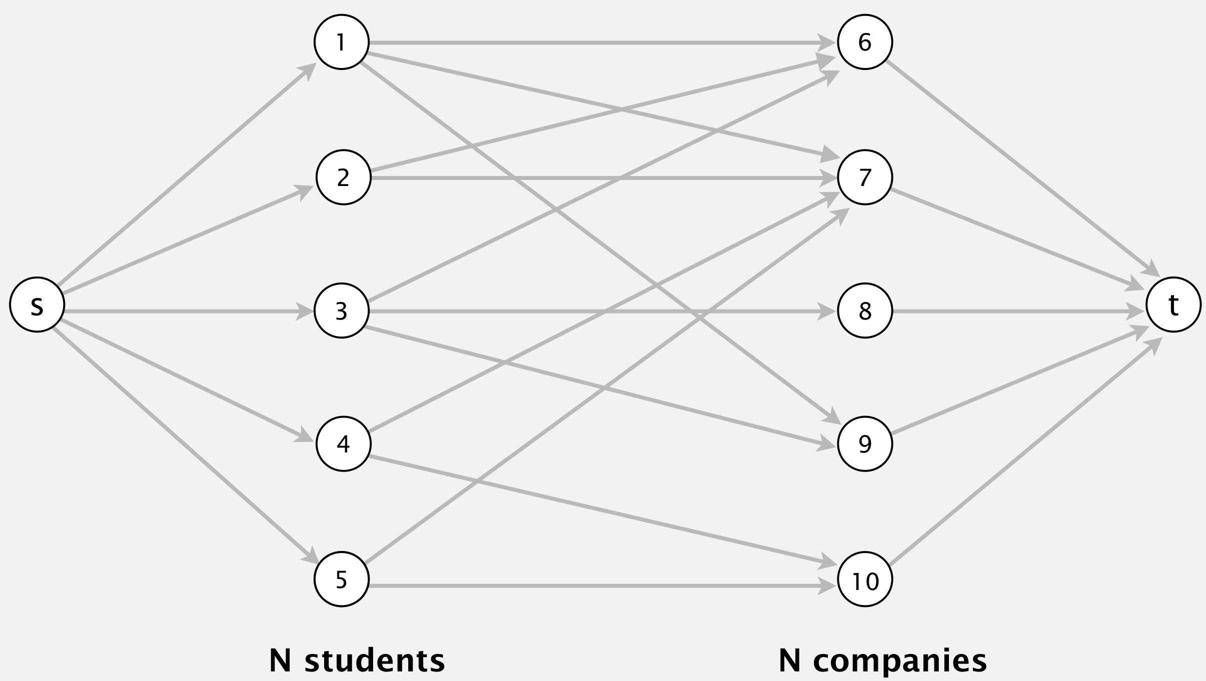
1 Alice	6 Adobe
Adobe	Alice
Amazon	Bob
Google	Carol
2 Bob	7 Amazon
Adobe	Alice
Amazon	Bob
3 Carol	Dave
Adobe	Eliza
Facebook	8 Facebook
Google	Carol
4 Dave	9 Google
Amazon	Alice
Yahoo	Carol
5 Eliza	10 Yahoo
Amazon	Dave
Yahoo	Eliza

Network flow formulation of bipartite matching

- 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).

also works if just 1

flow network



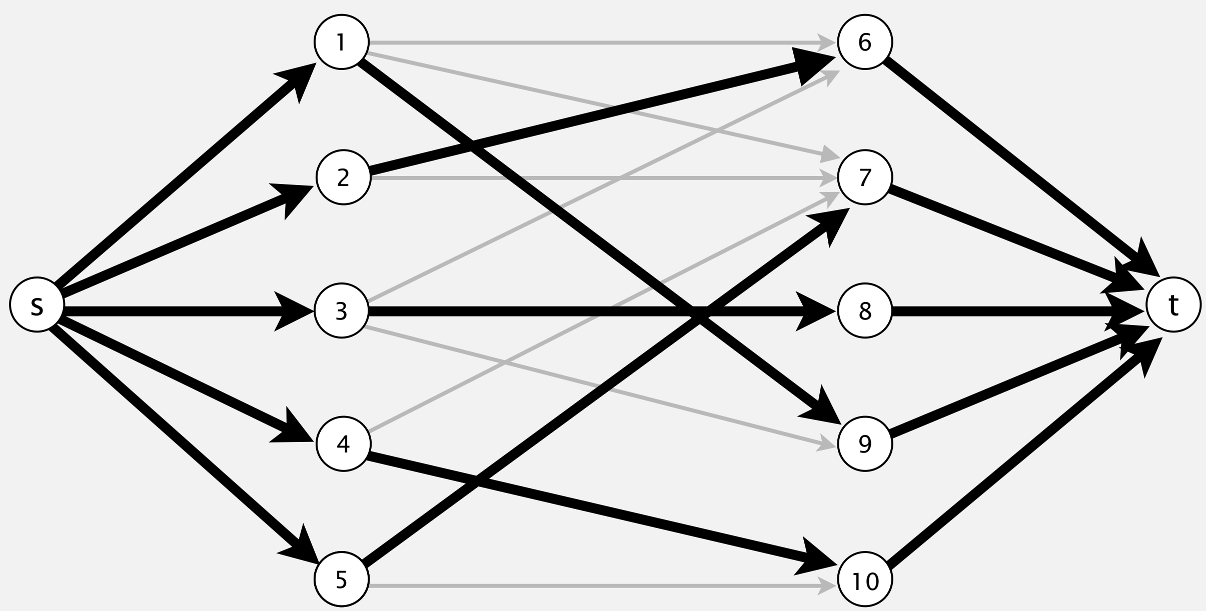
bipartite matching problem

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



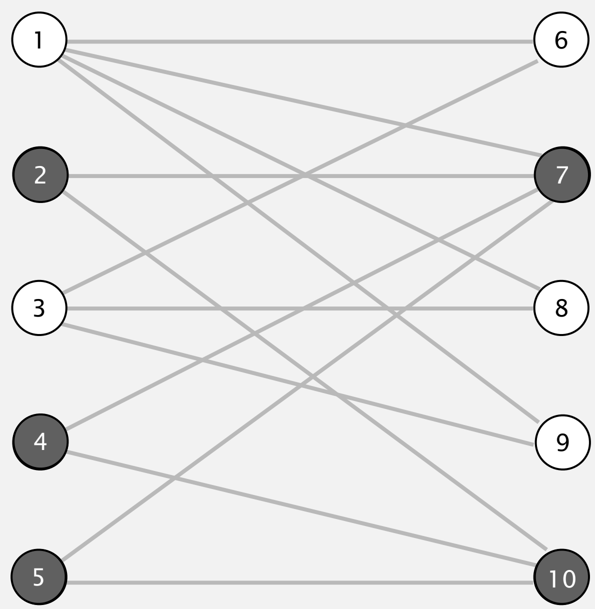
N students **N companies**

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 }

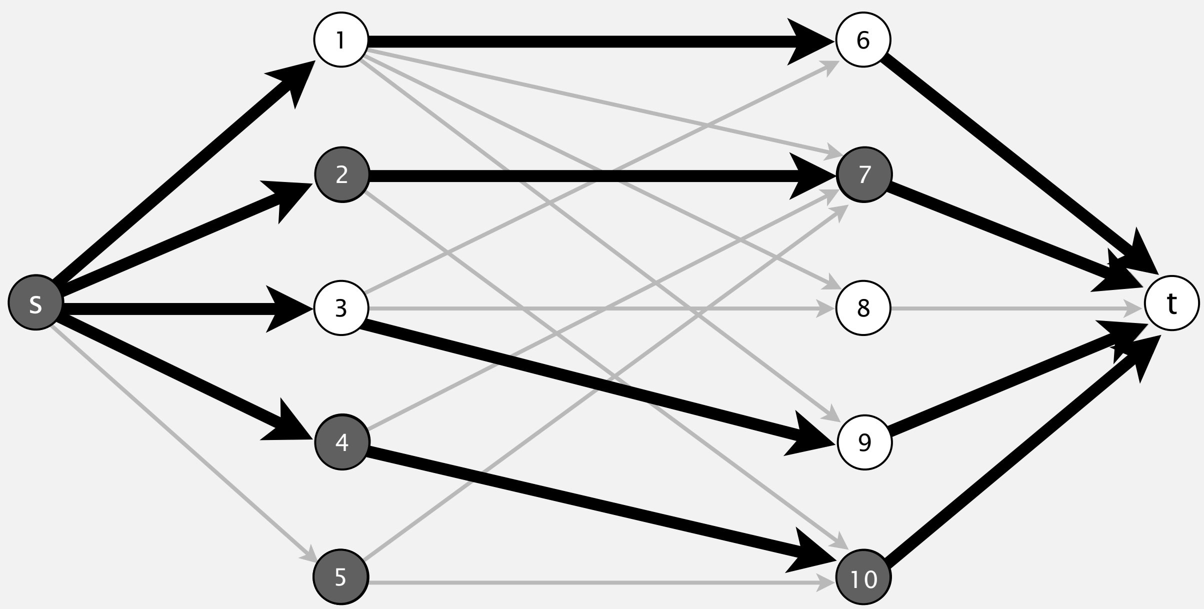
student in S
can be matched
only to
companies in T

| 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.

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)

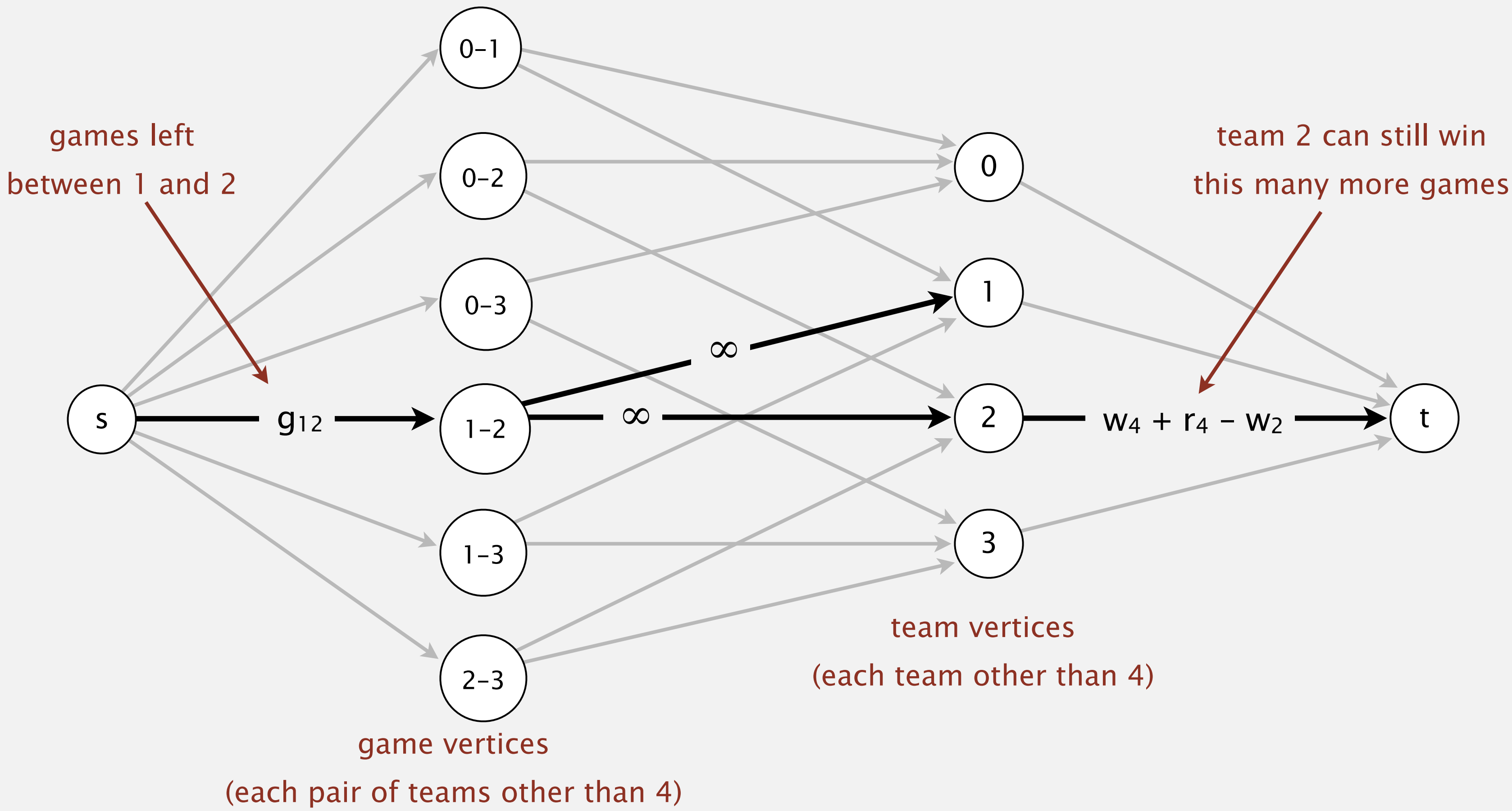
Detroit is mathematically eliminated.

- 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.

Build a flow network for EACH team. Below is graph for 4.



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	$E^2 \log E \log(EU)$	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-word maxflow/mincut problems in linear time.
- Theory: prove it for worst-case inputs.
- Still much to be learned!

http://algs4.cs.princeton.edu

6.4 Maximum Flow

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