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

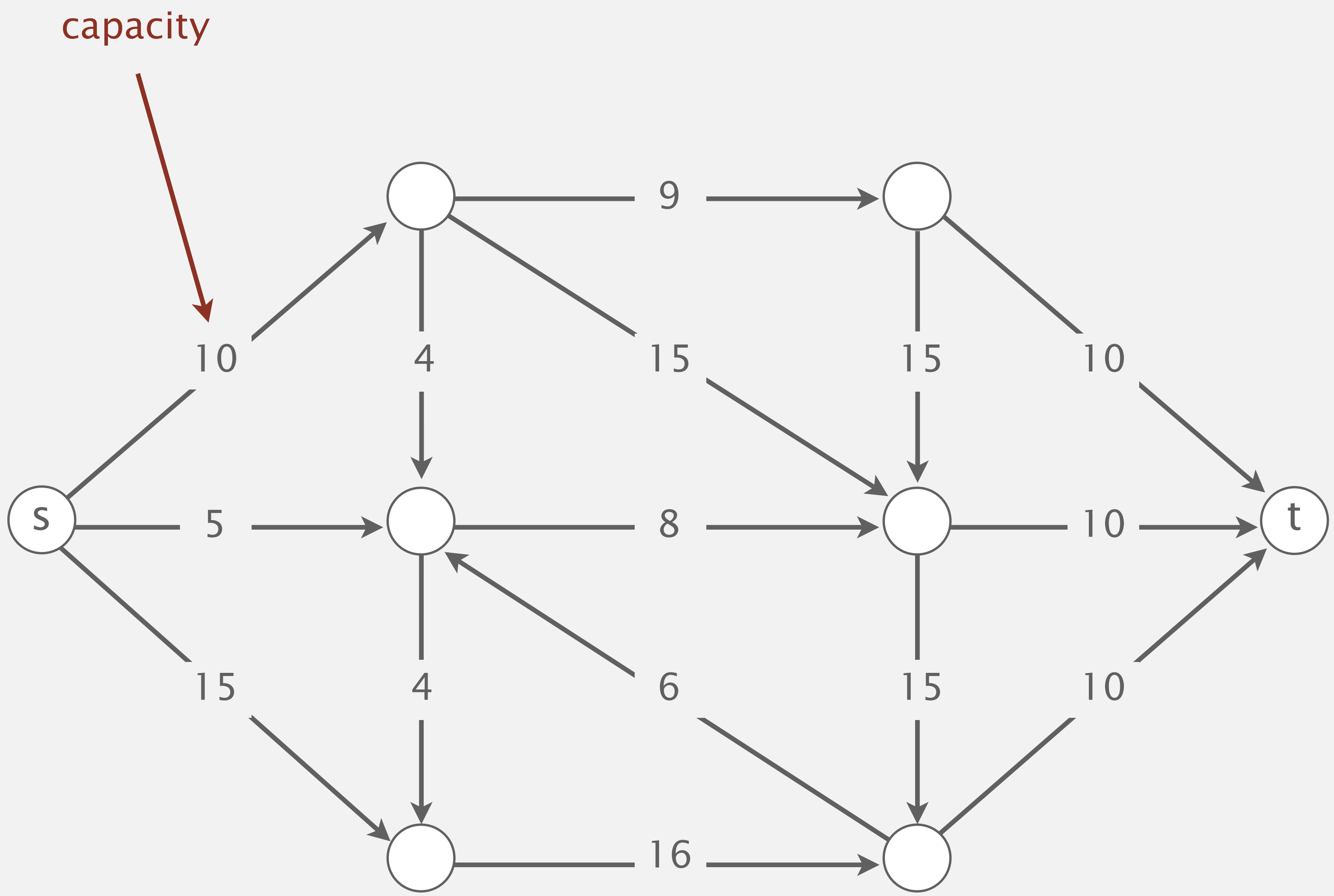
6.4 Maximum Flow

- *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.

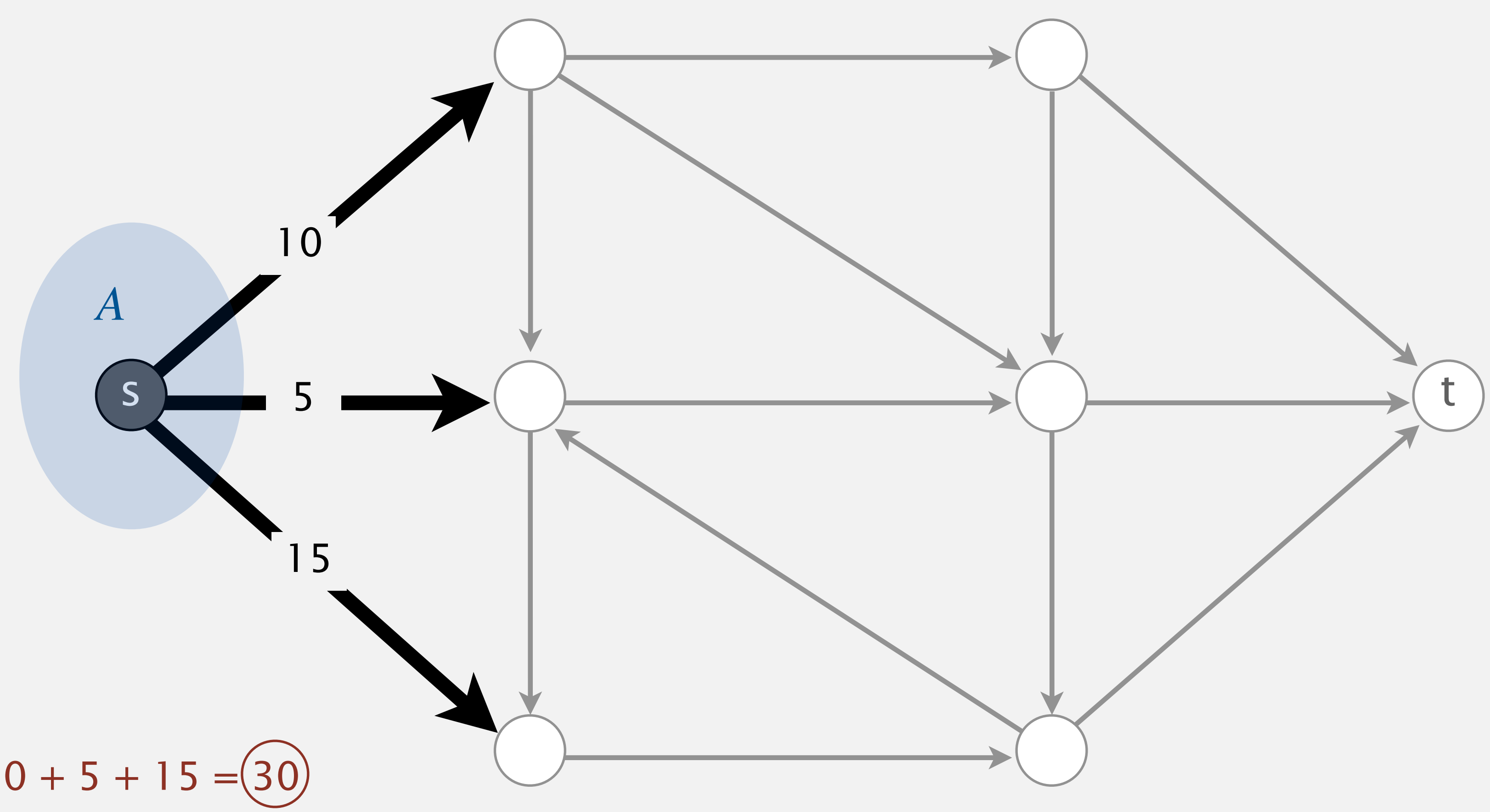
each edge has a positive capacity



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.

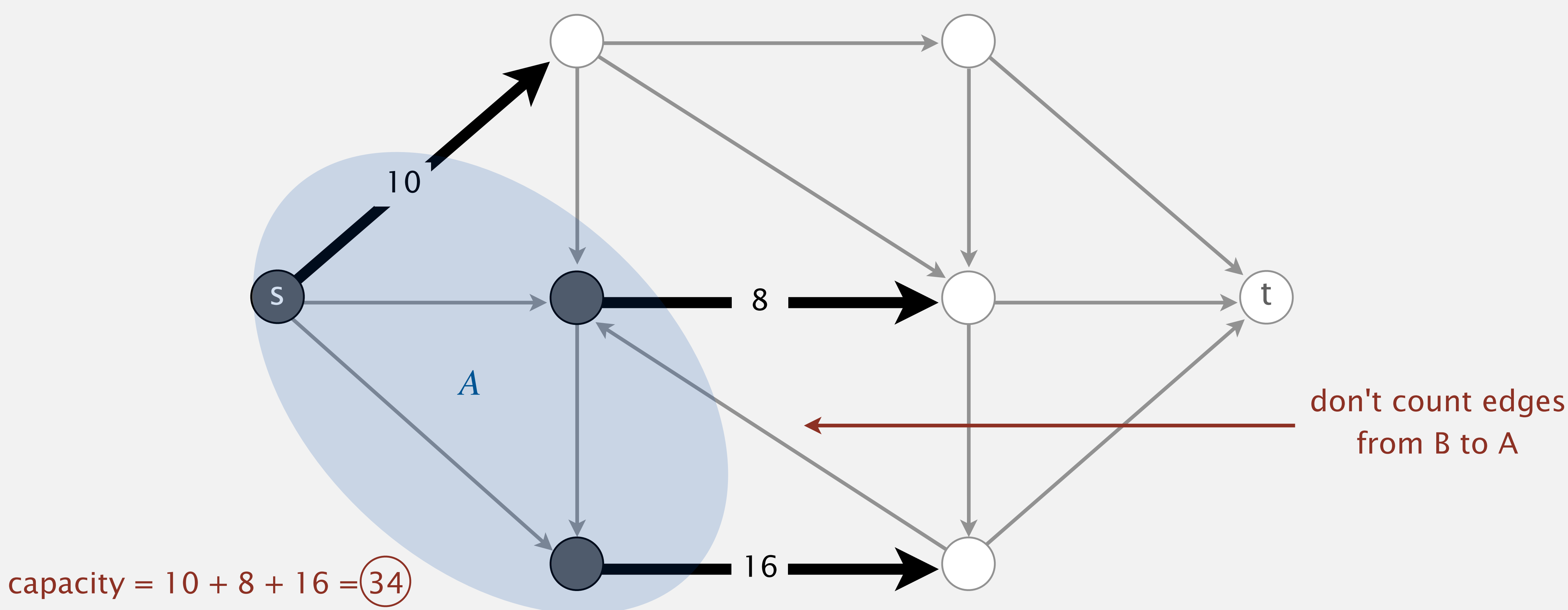


capacity = 10 + 5 + 15 = 30

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.

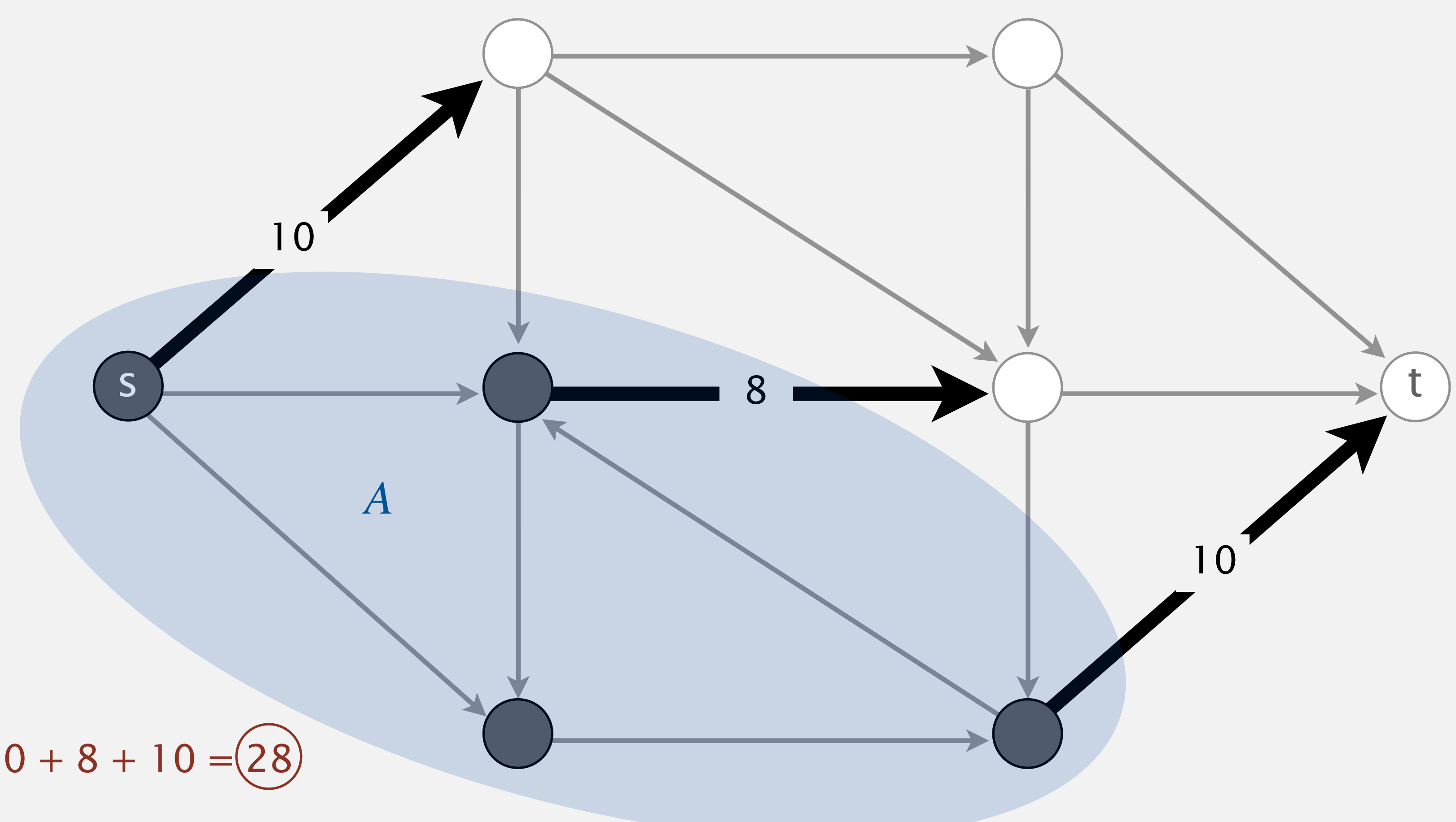


Mincut problem

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: quiz 1

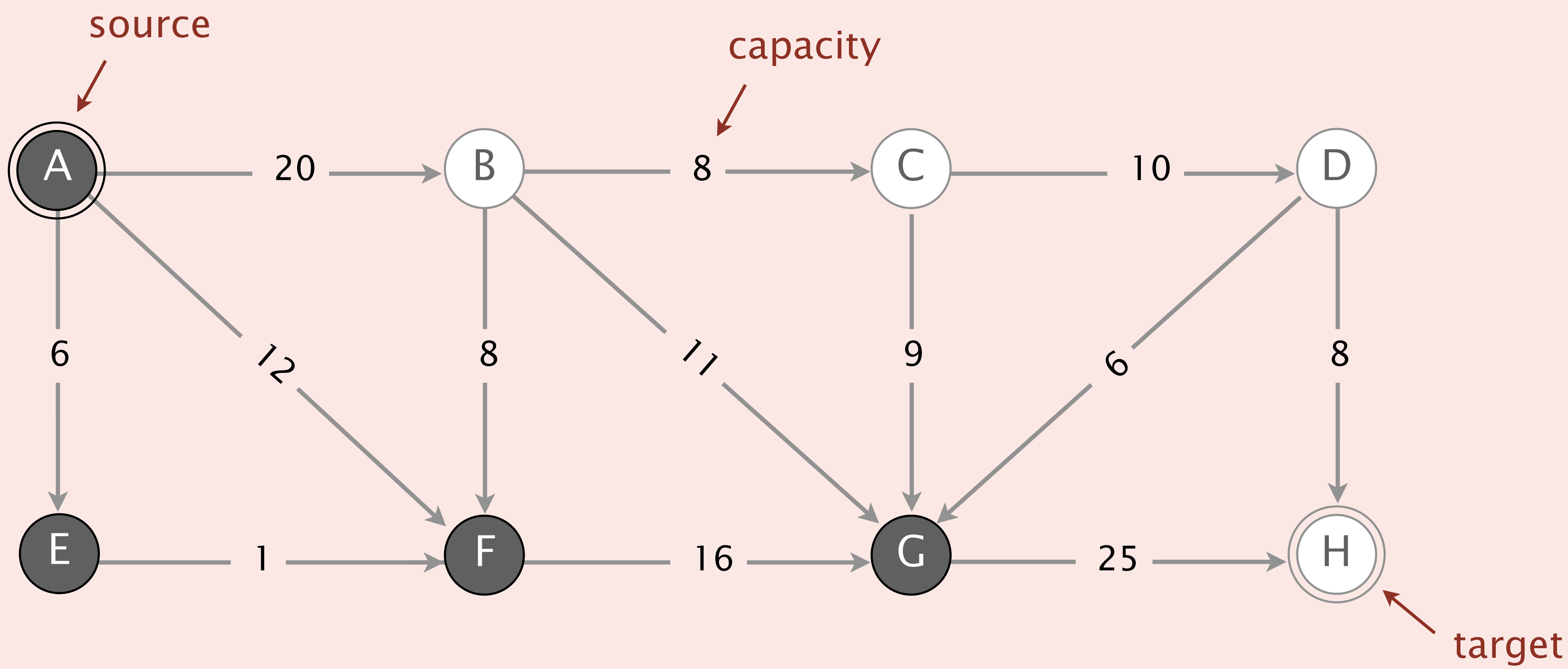
What is the capacity of the st-cut $\{A, E, F, G\}$?

A. 11 (20 + 25 – 8 – 11 – 9 – 6)

B. 34 (8 + 11 + 9 + 6)

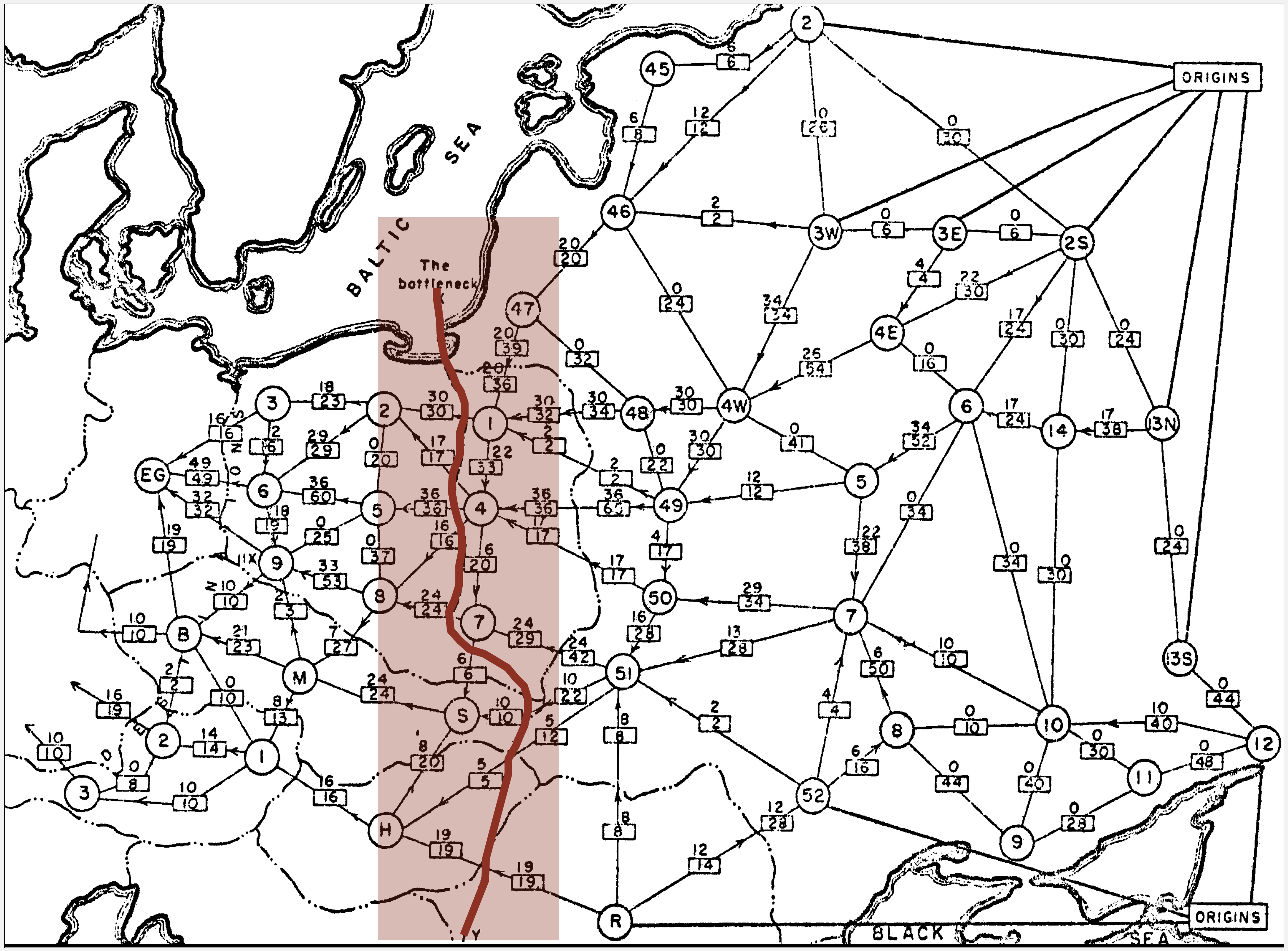
C. 45 (20 + 25)

D. 79 (20 + 25 + 8 + 11 + 9 + 6)



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


Though maximum flow algorithms have a long history, revolutionary progress is still being made.

BY ANDREW V. GOLDBERG AND ROBERT E. TARJAN

Efficient Maximum Flow Algorithms

gorithms in more detail. We restrict ourselves to basic maximum flow algorithms and do not cover interesting special cases (such as undirected graphs, planar graphs, and bipartite matchings) or generalizations (such as minimum-cost and multi-commodity flow problems).

Before formally defining the maximum flow and the minimum cut problems, we give a simple example of each problem: For the maximum flow example, suppose we have a graph that represents an oil pipeline network from an oil well to an oil depot. Each arc has a capacity, or maximum number of liters per second that can flow through the corresponding pipe. The goal is to find the maximum number of liters per second (maximum flow) that can be shipped from well to depot. For the minimum cut problem, we want to find the set of pipes of the smallest total capacity such that removing the pipes disconnects the oil well from the oil depot (minimum cut).

The maximum flow, minimum cut

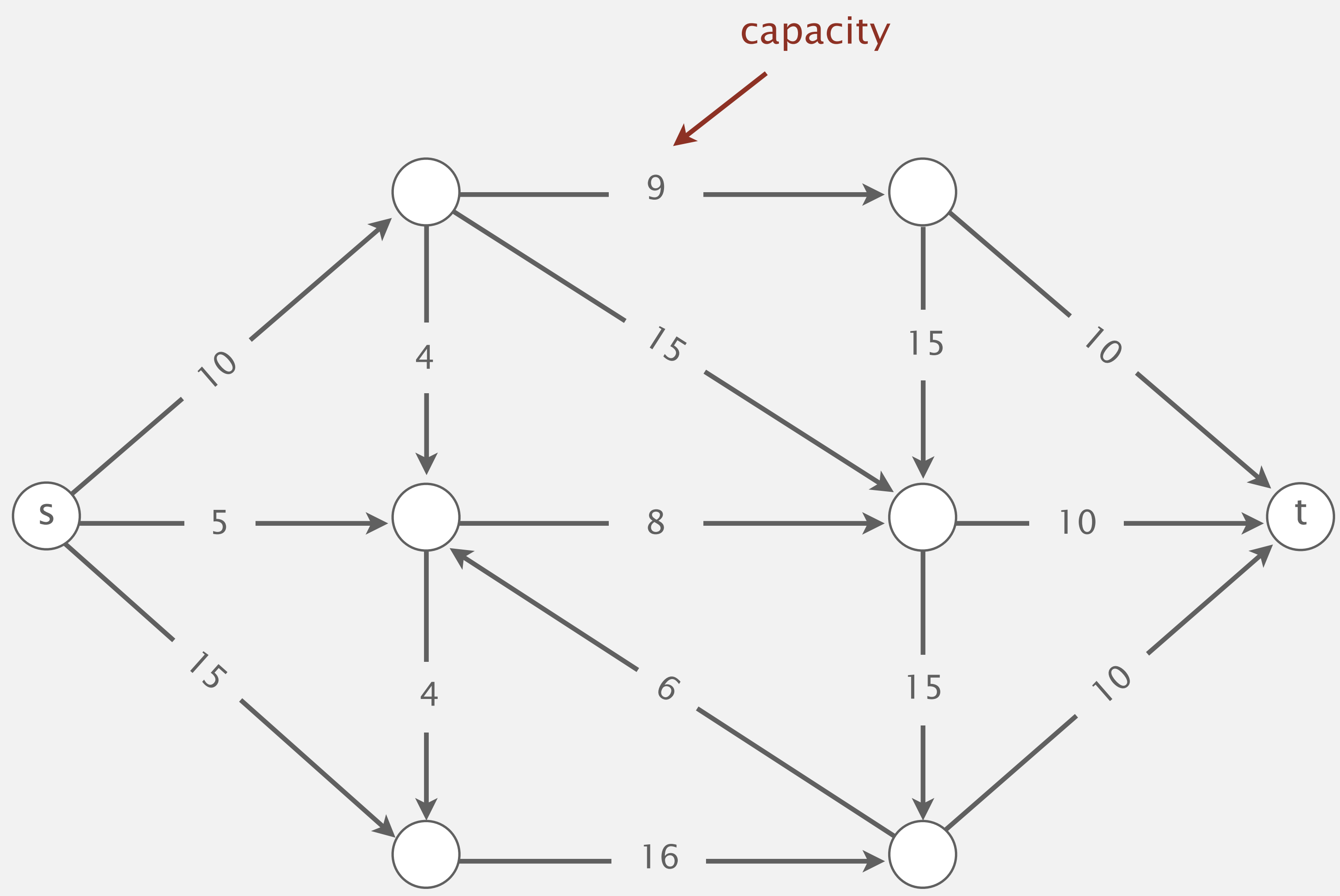
Efficient Maximum Flow Algorithms by Andrew Goldberg and Bob Tarjan

http://vimeo.com/100774435

Maxflow problem

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

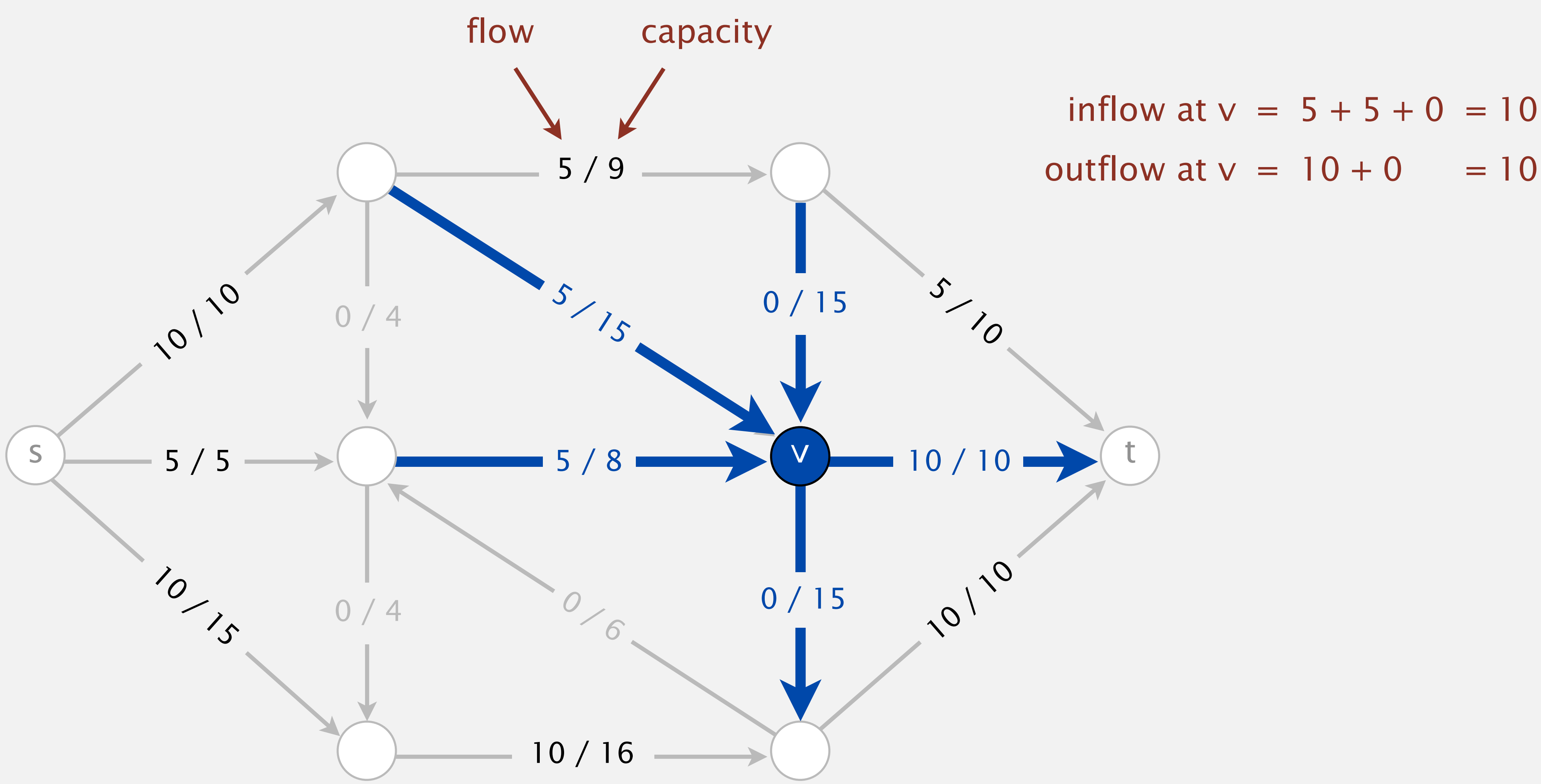
each edge has a positive capacity



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).



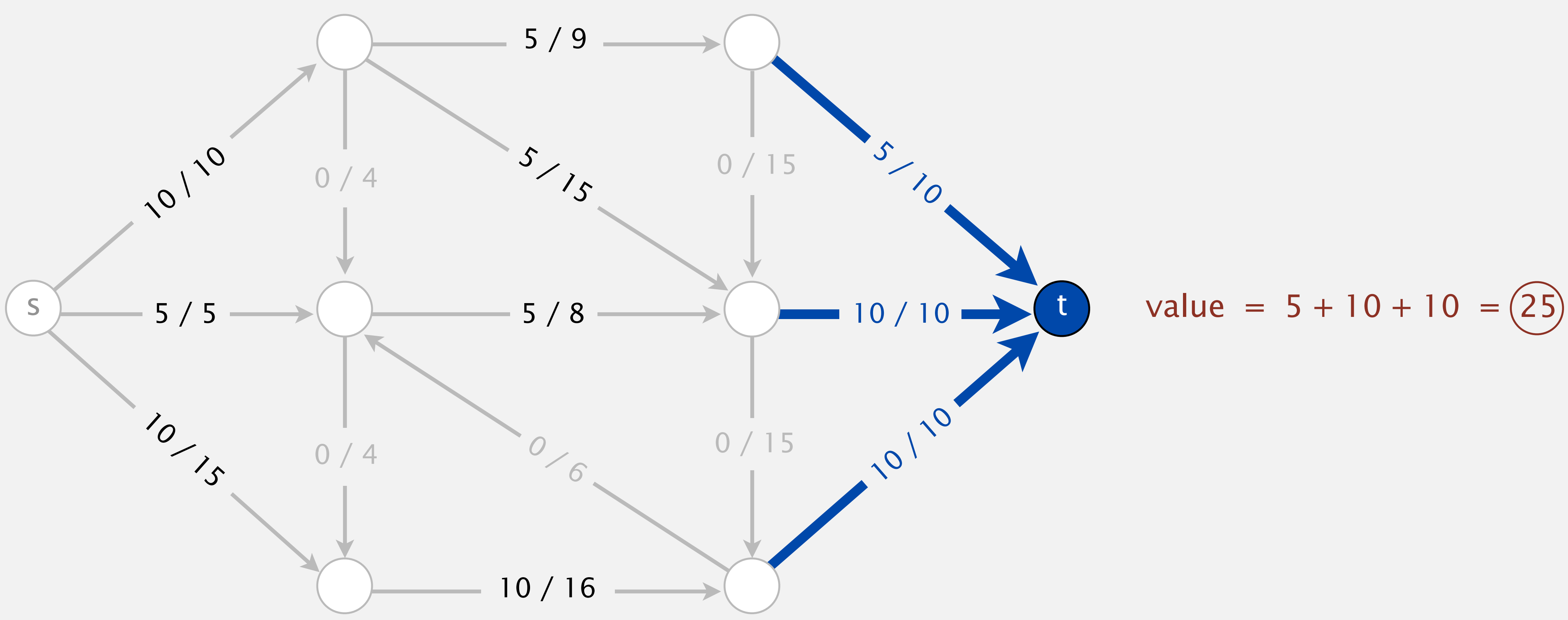
Maxflow problem

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.

we assume no edges point to s or from t



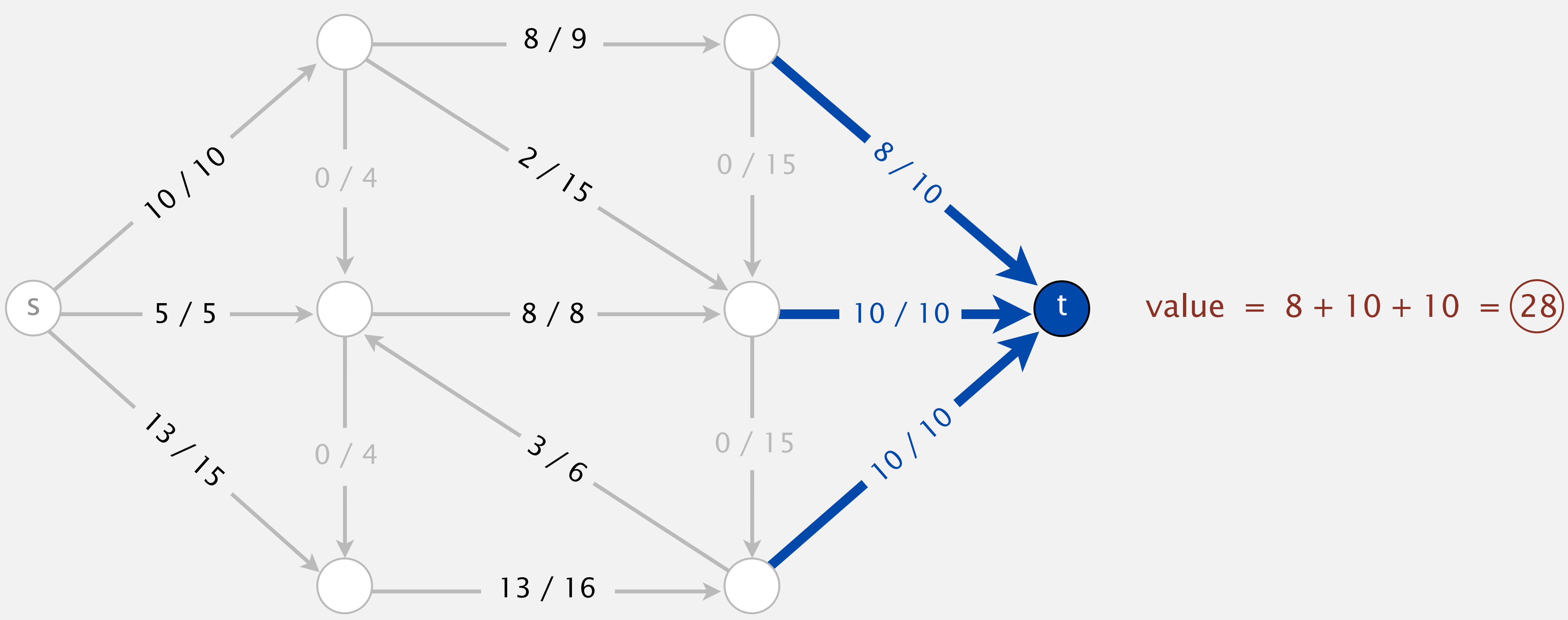
Maxflow problem

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.

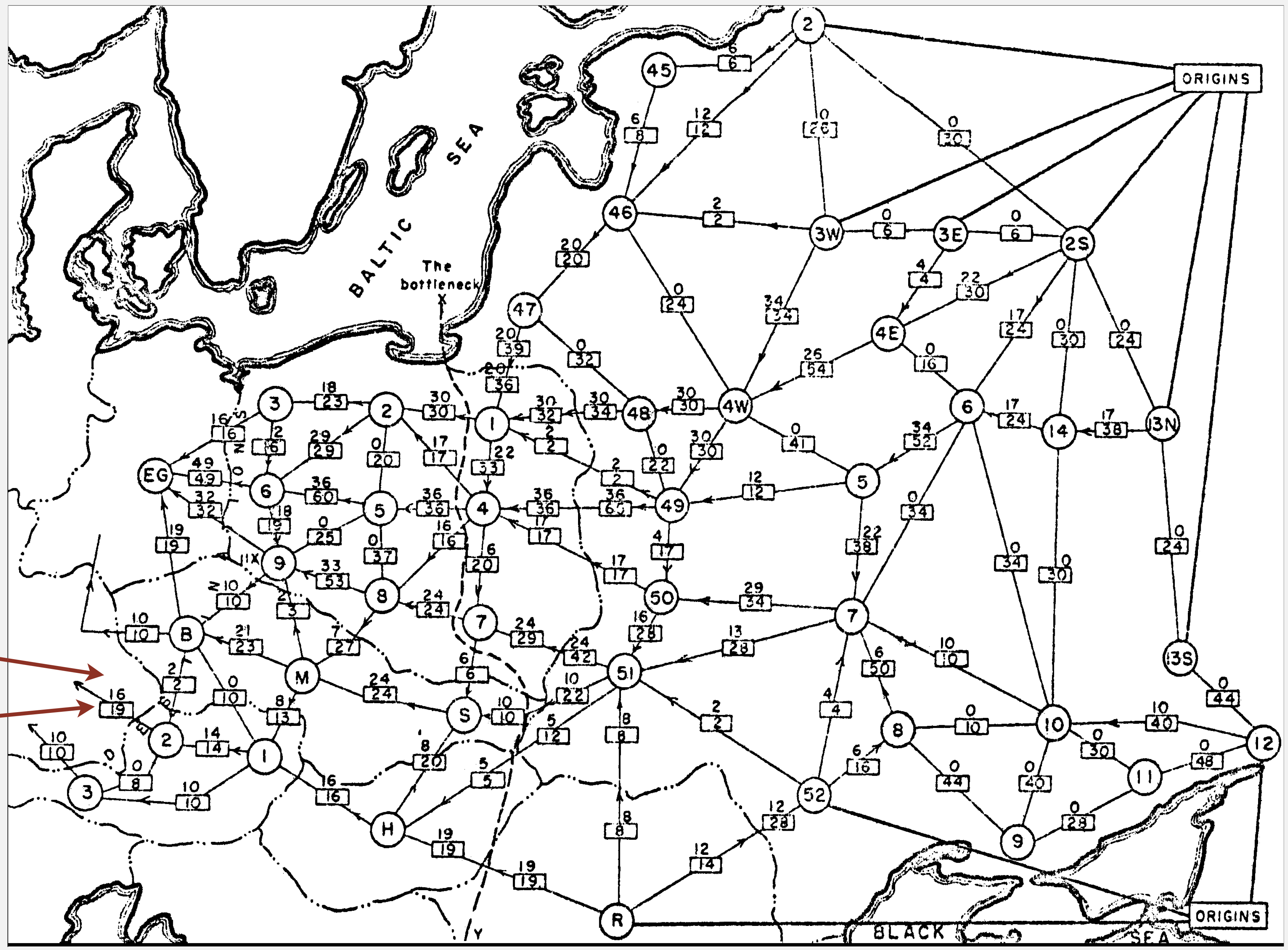


Maxflow application (Tolstoǐ 1930s)

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

flow

capacity



rail network connecting Soviet Union with Eastern European countries

(map declassified by Pentagon in 1999)

Potential maxflow application (2010s)

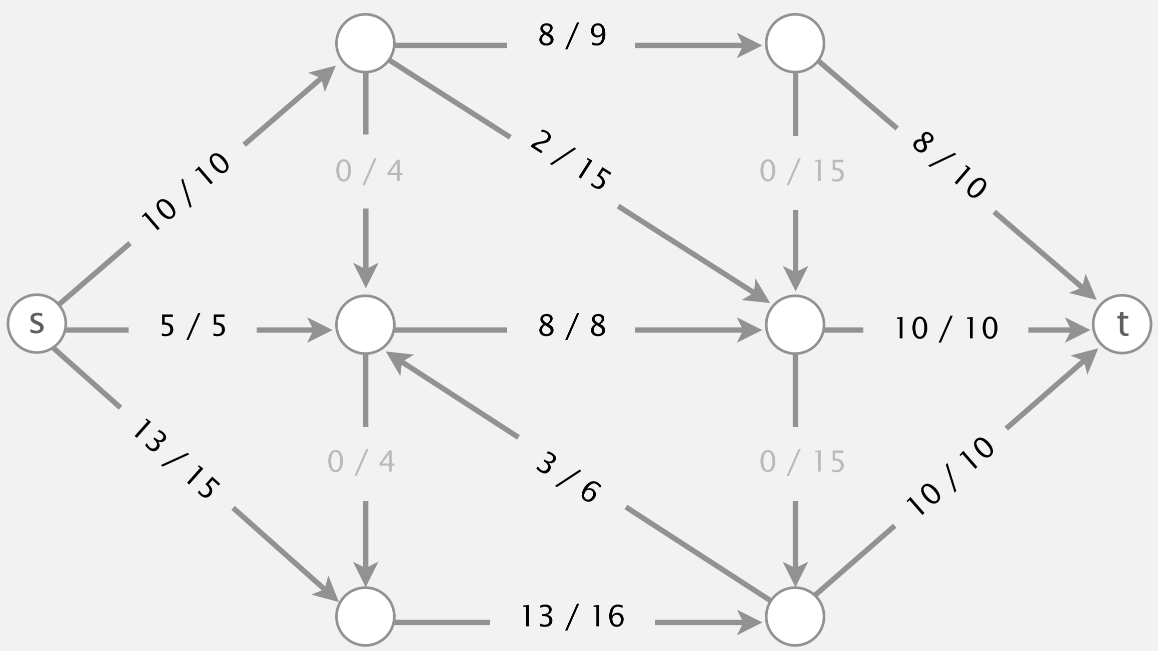
"Free world" goal. Maximize flow of information to specified set of people.

Summary

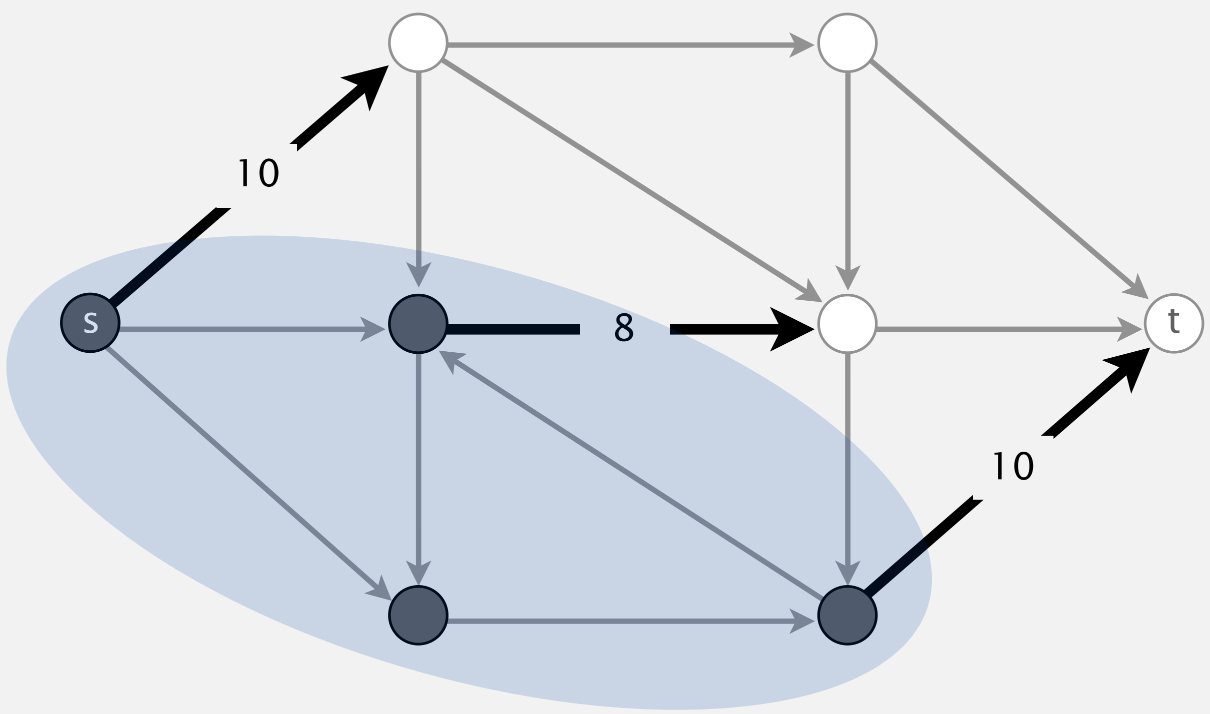
Input. An edge-weighted digraph, source vertex s, and target vertex t.

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!

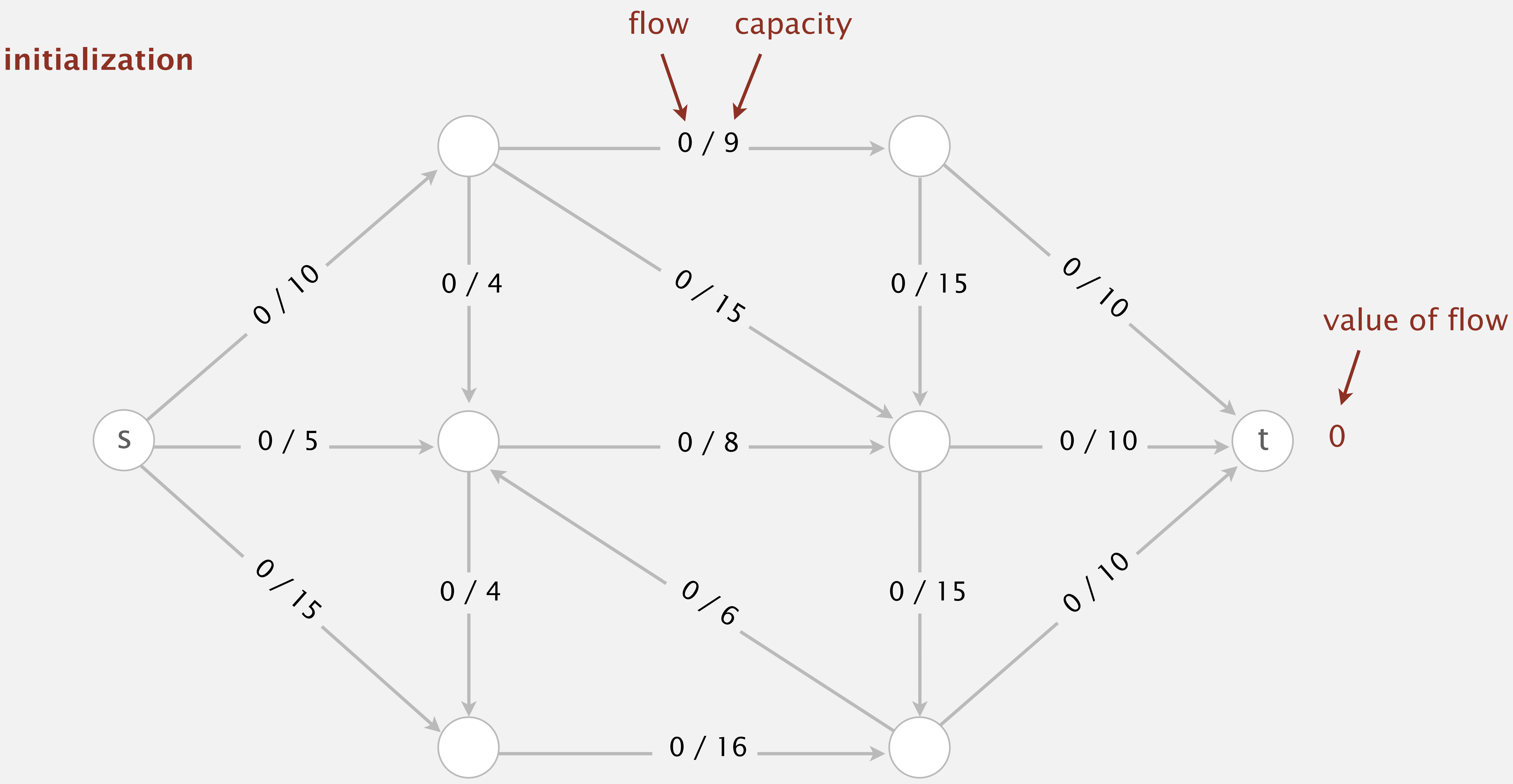
http://algs4.cs.princeton.edu

6.4 Maximum Flow

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

Ford–Fulkerson algorithm

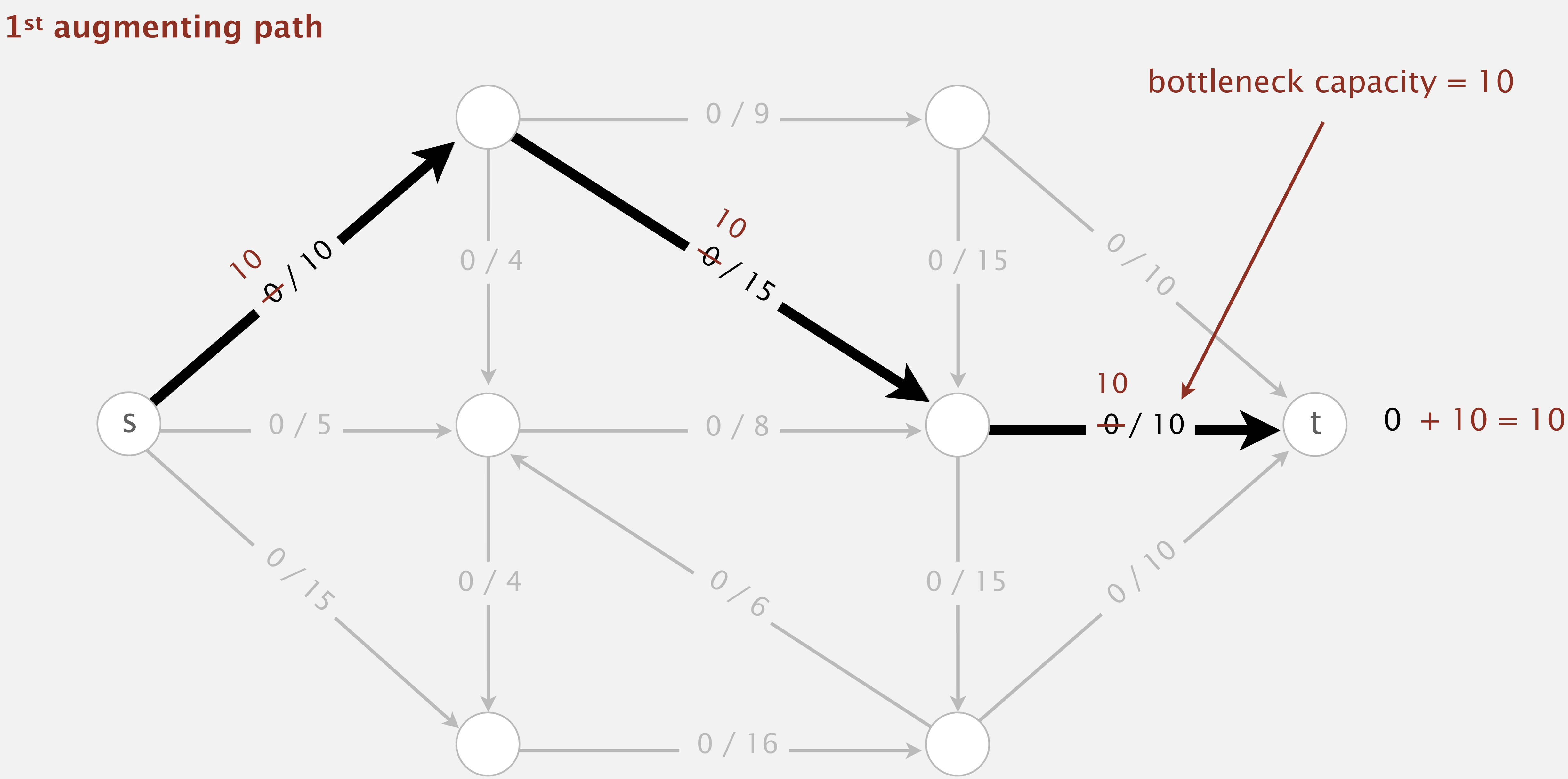
Initialization. Start with 0 flow.



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).

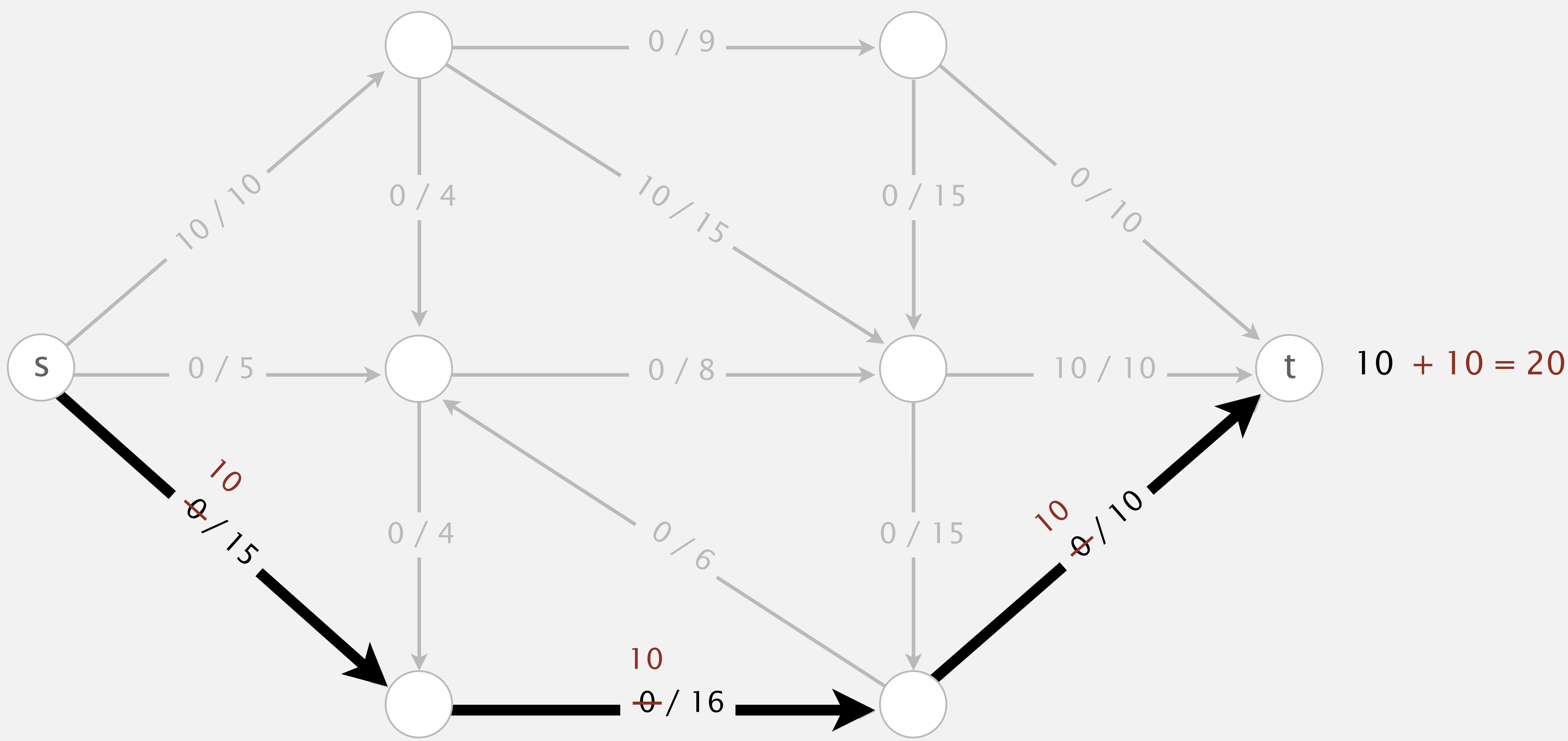


Idea: increase flow along augmenting paths

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).

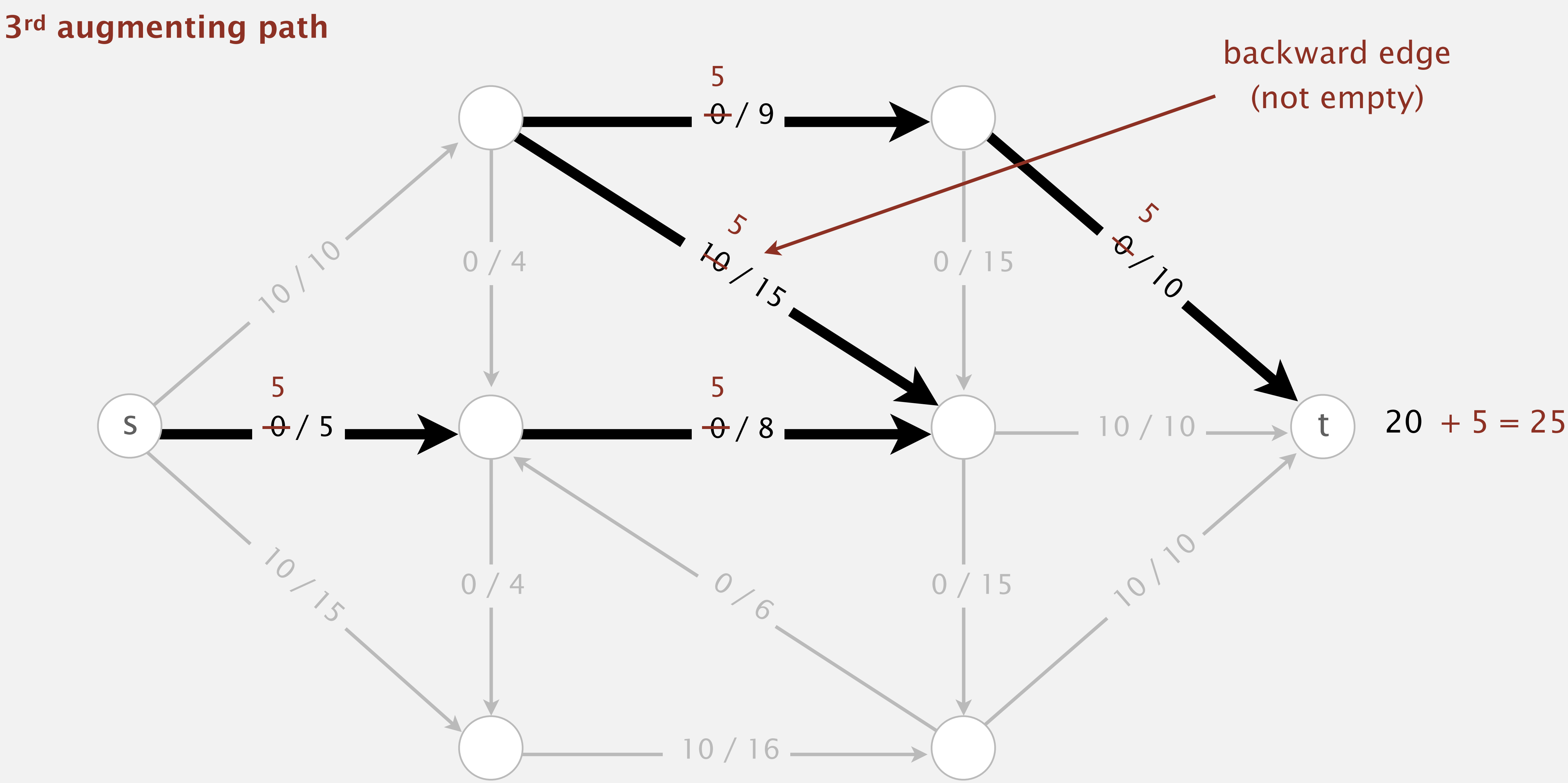
2nd augmenting path



Idea: increase flow along augmenting paths

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

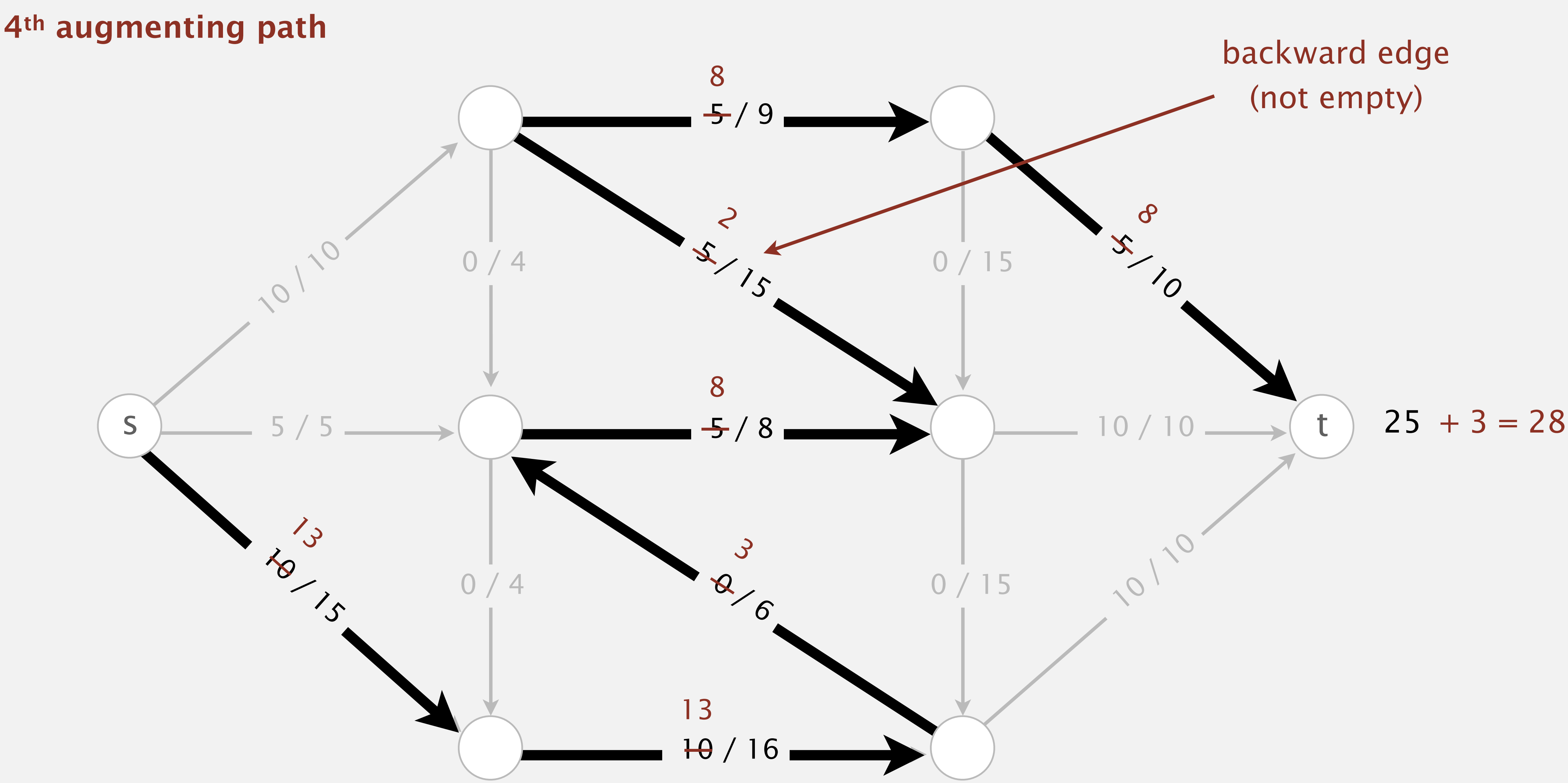
- Can increase flow on forward edges (not full).
- Can decrease flow on backward edge (not empty).



Idea: increase flow along augmenting paths

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

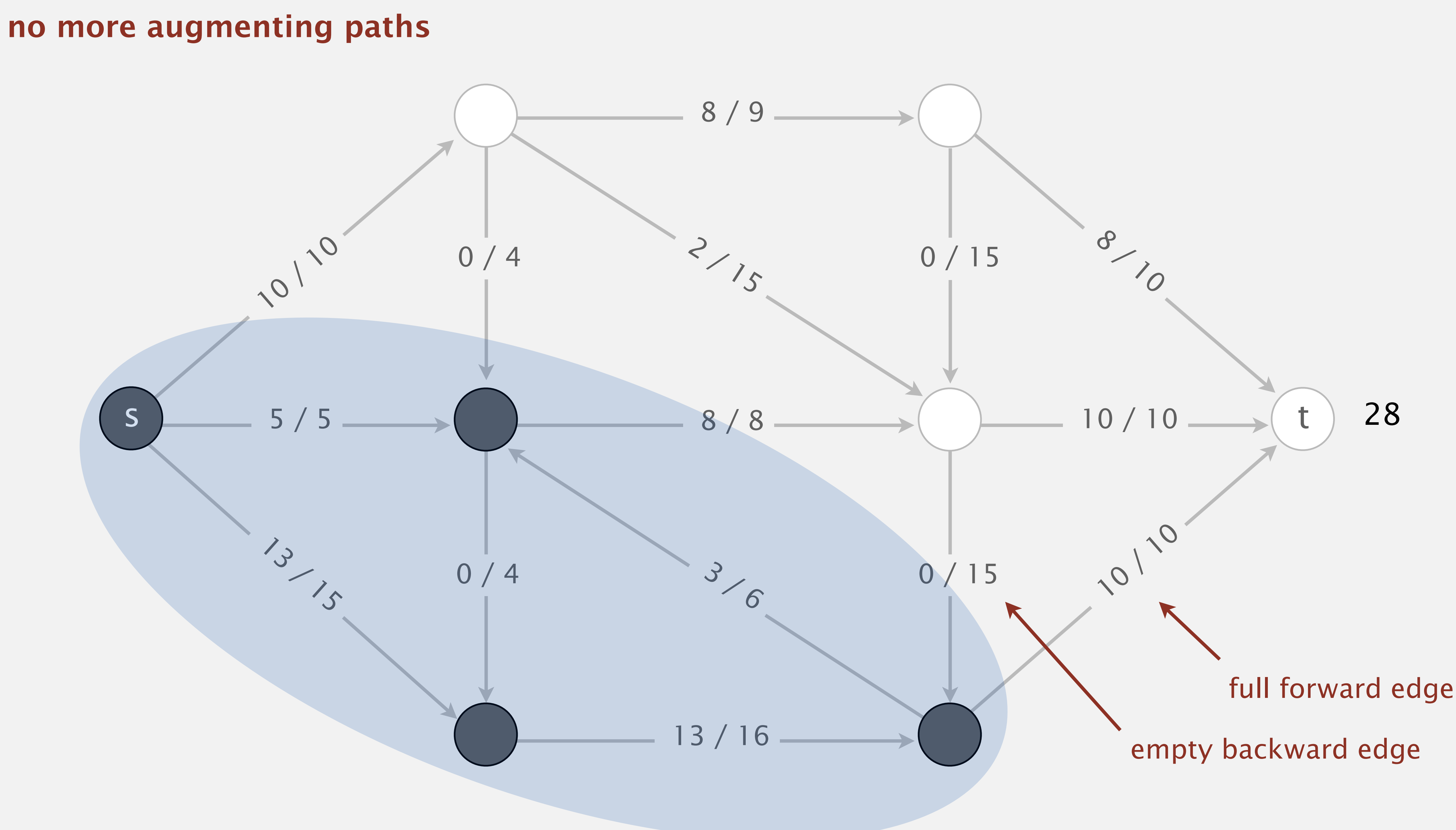
- Can increase flow on forward edges (not full).
- Can decrease flow on backward edge (not empty).



Idea: increase flow along augmenting paths

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

- Full forward edge.
- Empty backward edge.



Maxflow: quiz 2

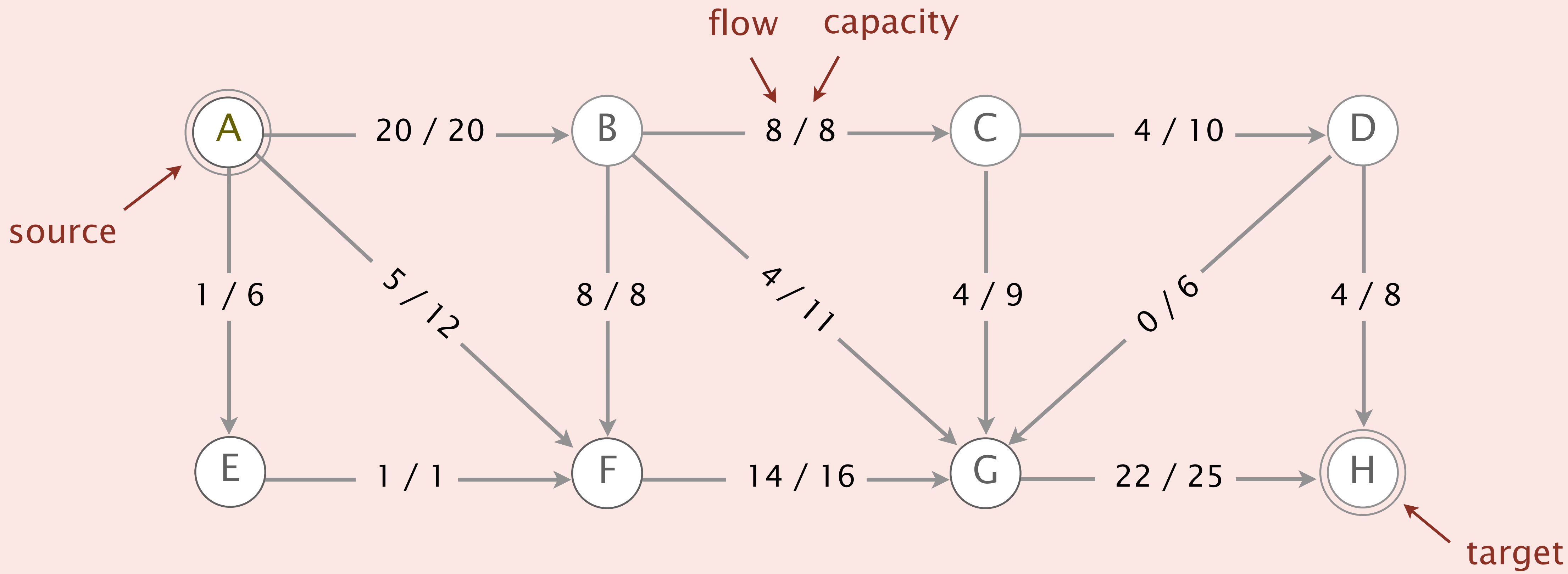
Which is the augmenting path of highest bottleneck capacity?

A. $A \to F \to G \to H$

B. $A \to B \to C \to D \to H$

C. $A \to F \to B \to G \to H$

D. $A \to F \to B \to G \to C \to D \to H$



Ford–Fulkerson algorithm

Ford–Fulkerson algorithm

Start with 0 flow.

While there exists an augmenting path:

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

Fundamental 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?

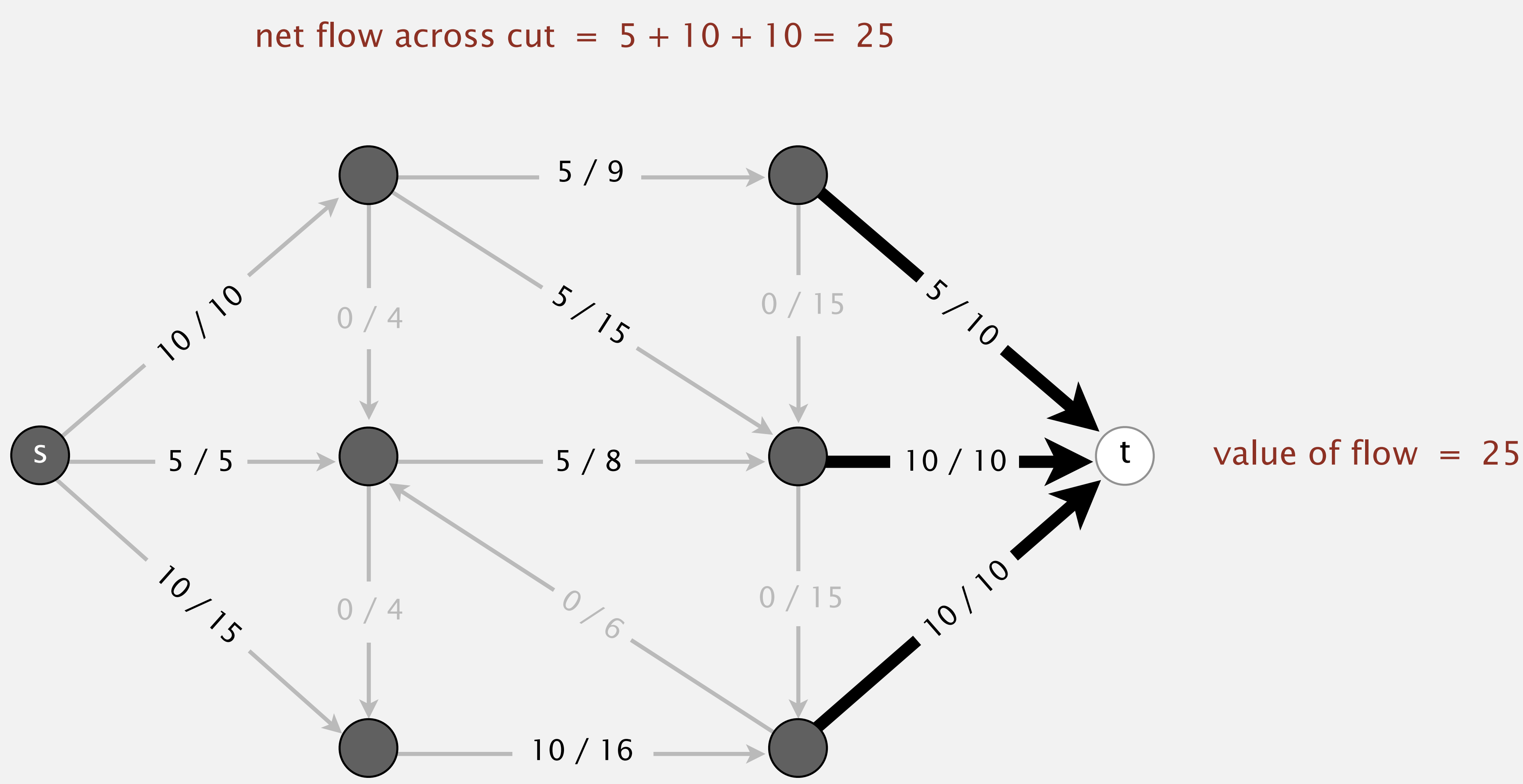
http://algs4.cs.princeton.edu

6.4 Maximum Flow

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

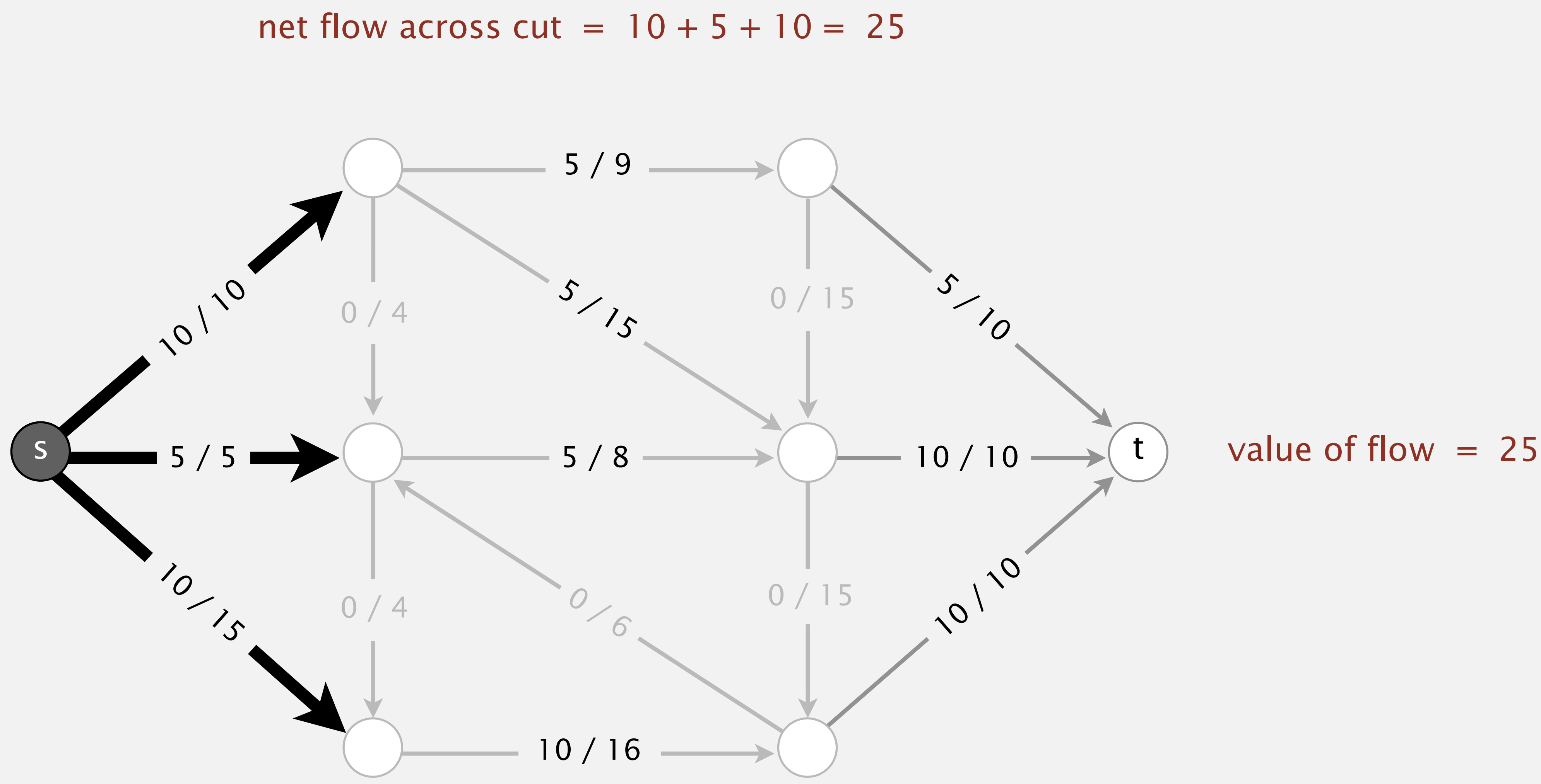
Relationship between flows and cuts

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.



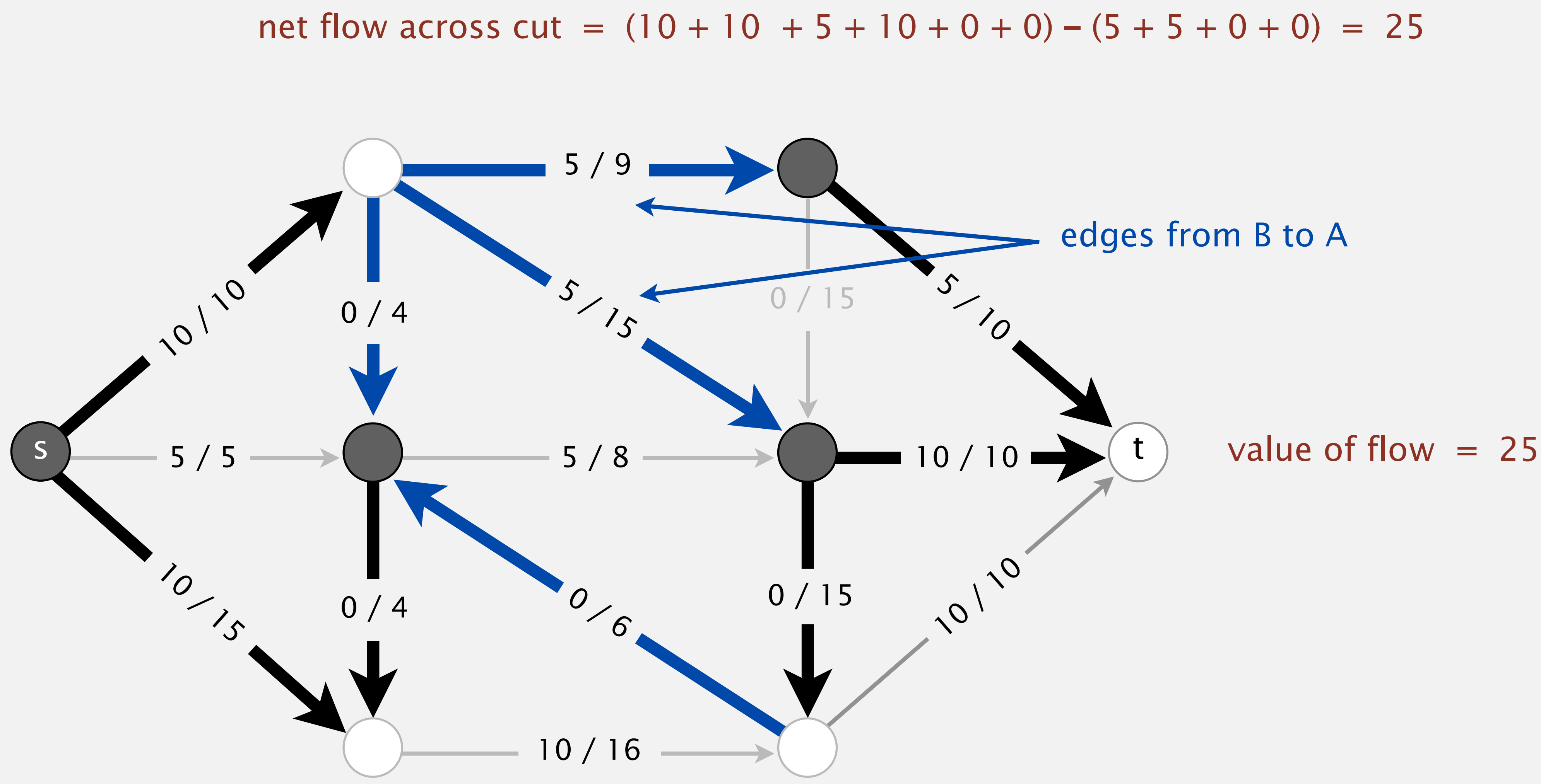
Relationship between flows and cuts

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.



Maxflow: quiz 3

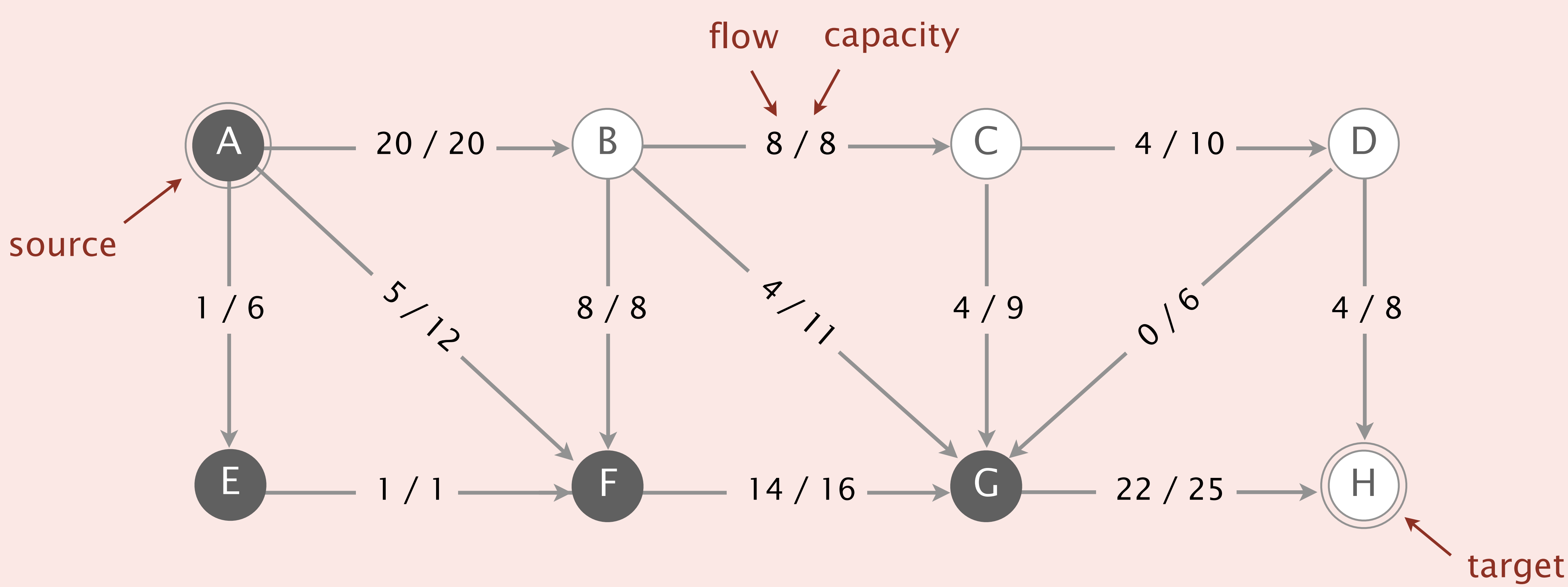
Which is the net flow across the *st*-cut $\{A, E, F, G\}$?

A. 11 (20 + 25 – 8 – 11 – 9 – 6)

B. 26 (20 + 22 – 8 – 4 – 4)

C. 42 (20 + 22)

D. 45 (20 + 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.

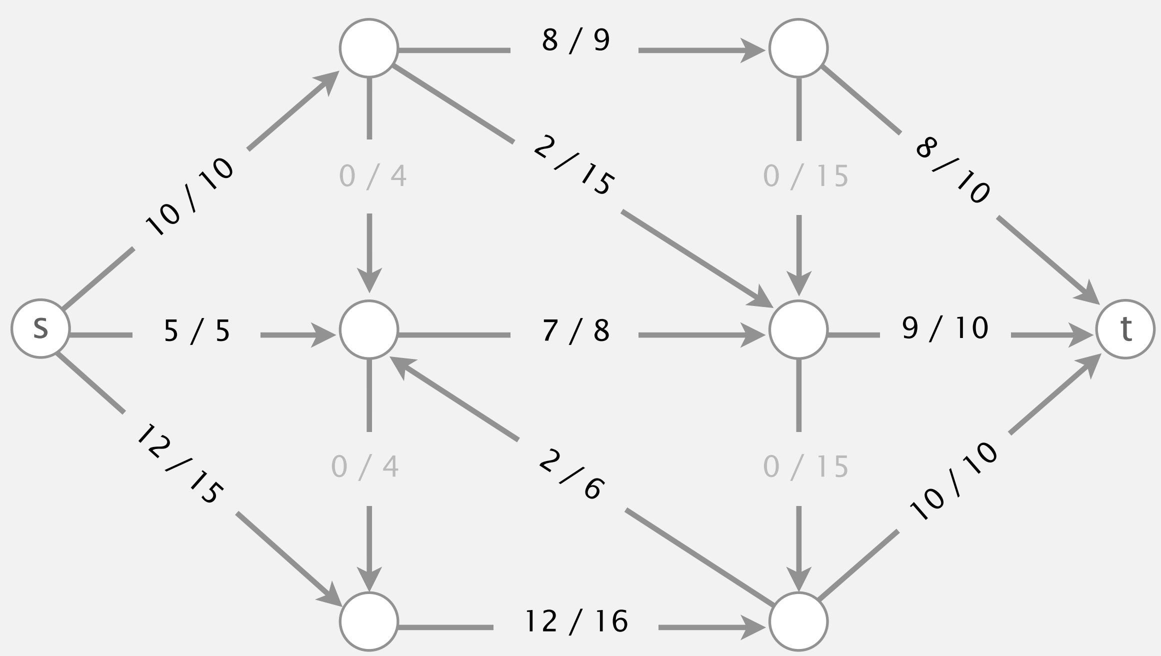
Relationship between flows and cuts

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

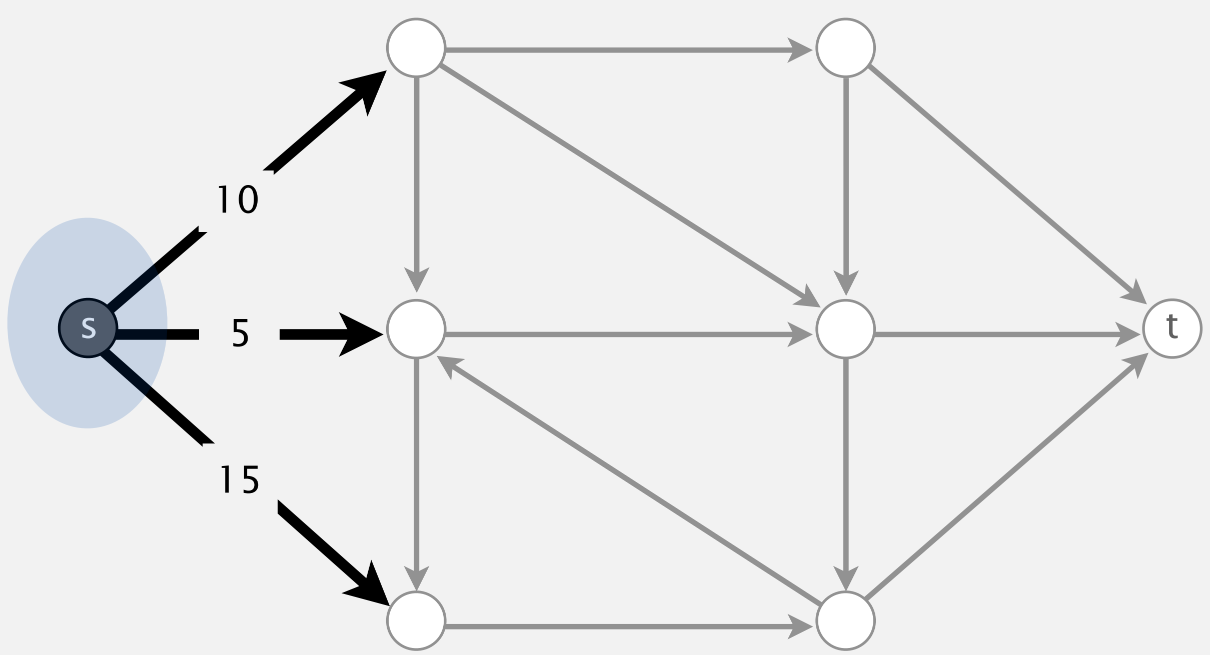
Pf. Value of flow f = net flow across cut (A, B) $\leq$ capacity of cut (A, B).

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.

strong duality

Pf. For any flow f, the following three conditions are equivalent:

i. f is a maxflow.

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

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

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

- 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. For any flow f, the following three conditions are equivalent:

i. f is a maxflow.

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

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

[iii ⇒ i]

- Suppose that (A, B) is a cut with capacity equal to the value of f.
- Then, the value of any flow $f\,' \underset{\text{weak duality}}{\leq}$ capacity of $(A, B) \underset{\text{by assumption}}{=}$ value of f.
- Thus, f is a maxflow. ▪

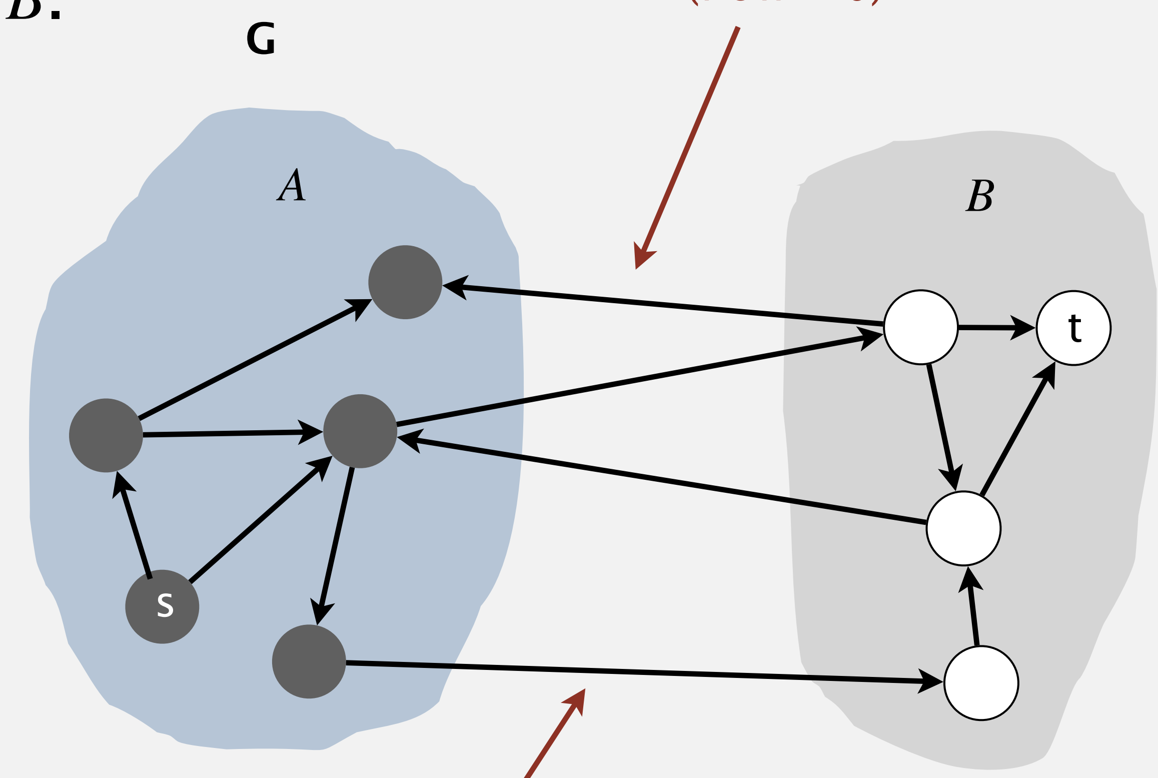
Maxflow–mincut theorem

[ii ⇒ iii]

- Let f be a flow with no augmenting paths.
- Let A be set of vertices connected to s by an undirected path with no full forward or empty backward edges.
- By definition of cut A, s is in A.
- By definition of cut A and flow f, t is in B.
- Capacity of cut = net flow across cut
  = value of flow f. ▪

flow–value
lemma

backward edge from B to A
(flow = 0)

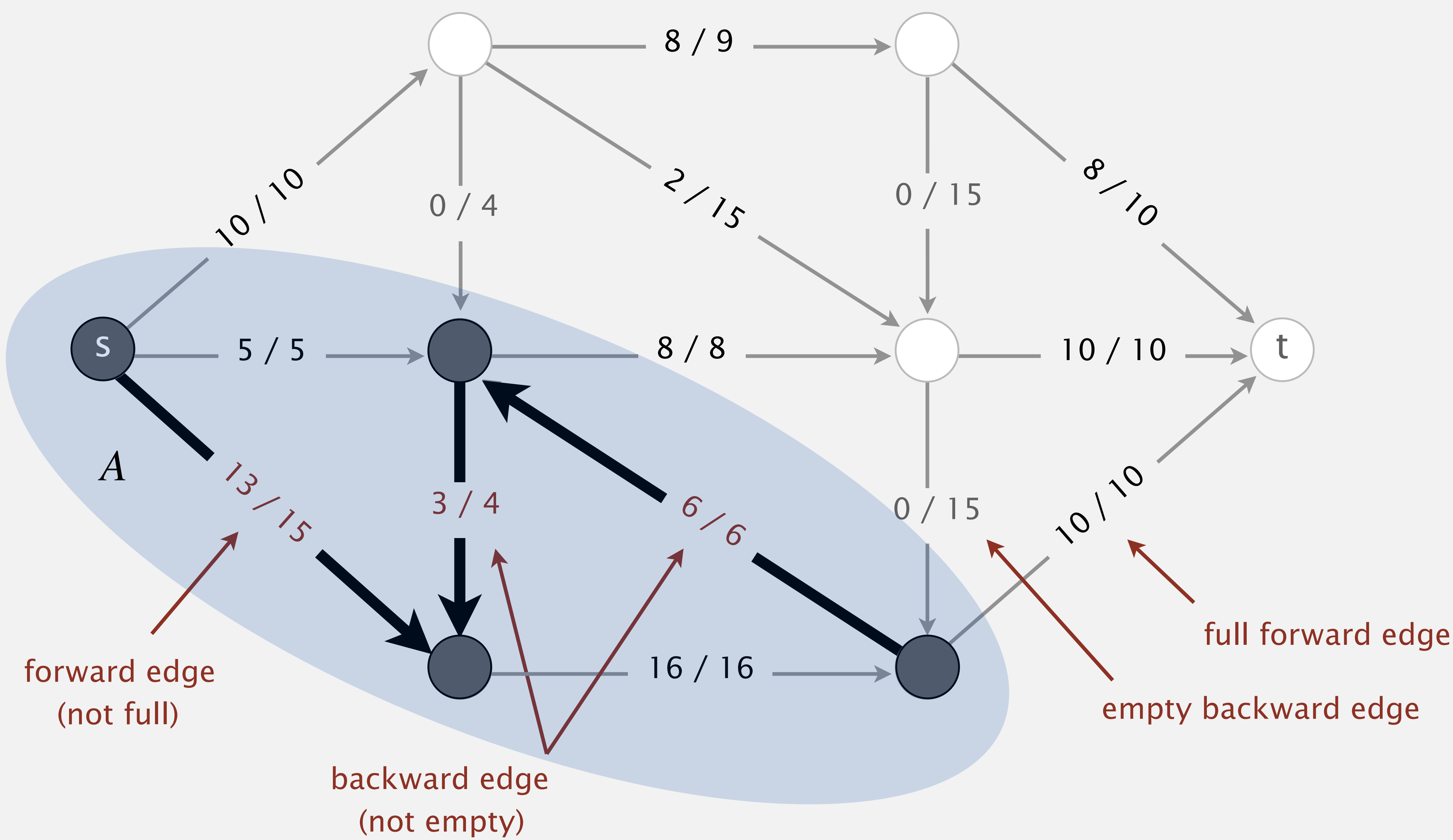


forward edge from A to B
(flow = capacity)

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.
- Capacity of cut (A, B) = value of flow f.



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**
- **update flow on that path by bottleneck capacity**

Fundamental questions.

- How to find an augmenting path? BFS works well.
- Given a maxflow, how to compute a mincut? BFS or DFS. ✔
- 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 analysis (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 integral 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.

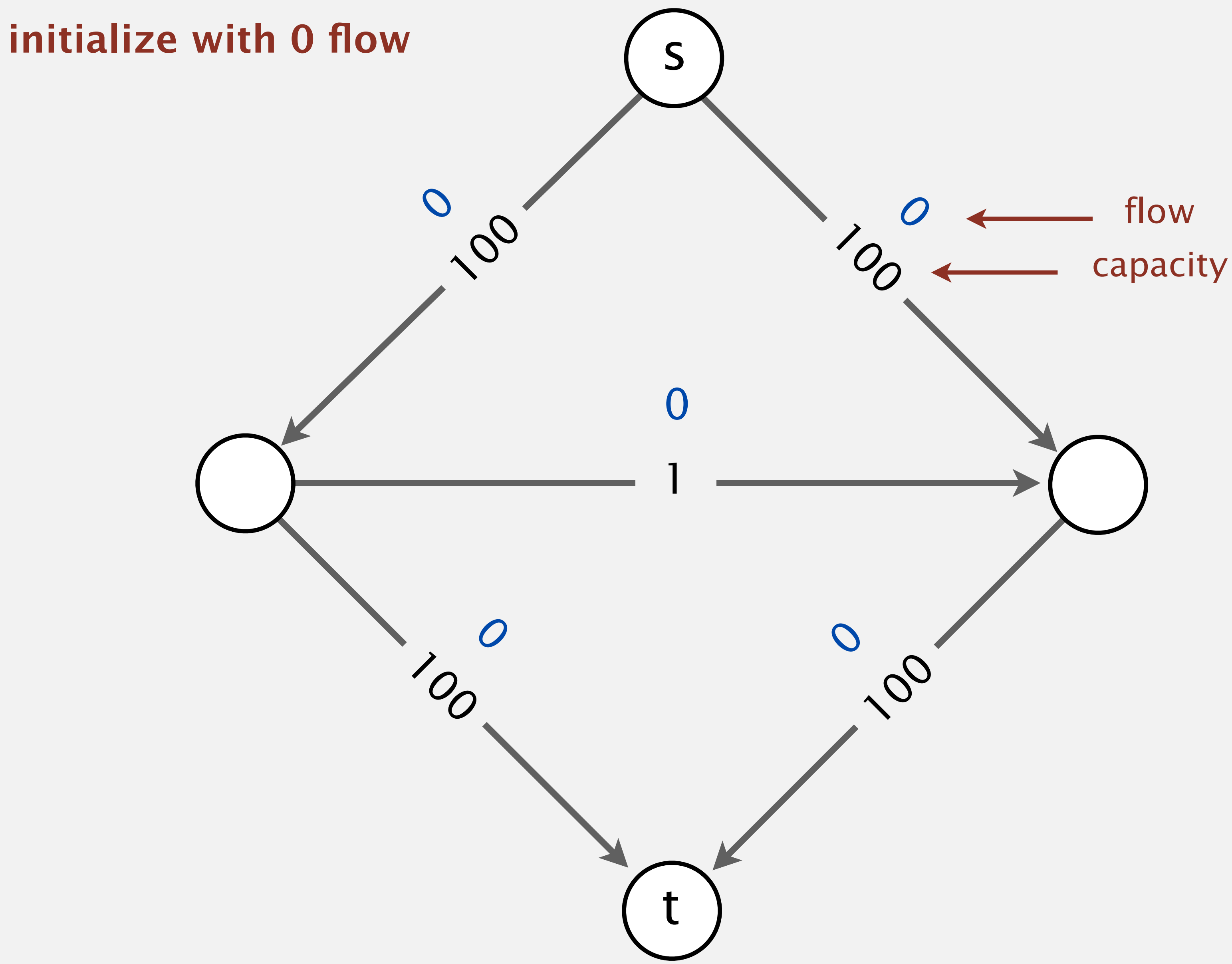
critical for some applications (stay tuned)

Integrality theorem. There exists an integral maxflow.

Pf. Ford–Fulkerson terminates and maxflow that it finds is integral.

Bad case for Ford–Fulkerson

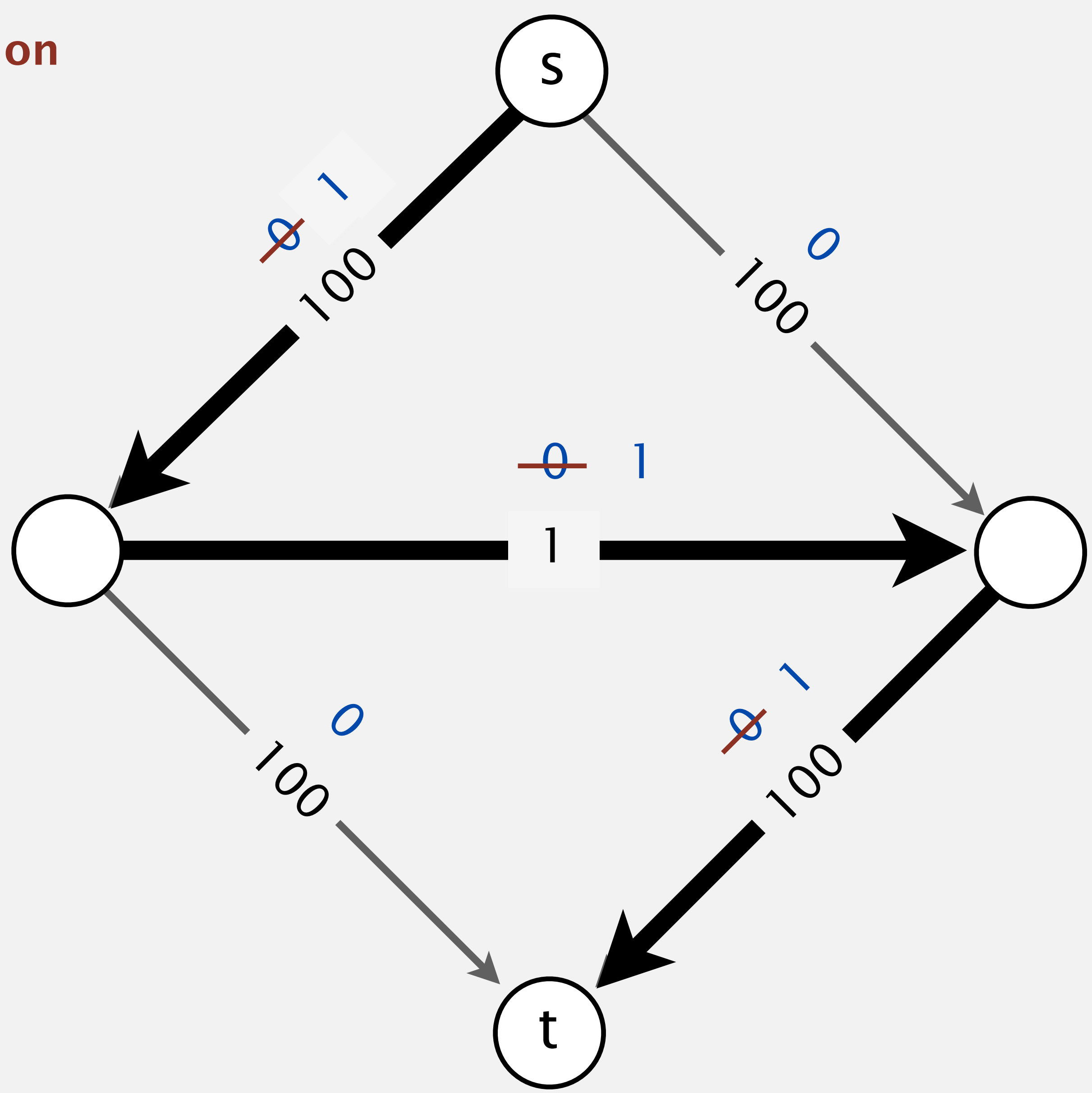
Bad news. Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maxflow.



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.

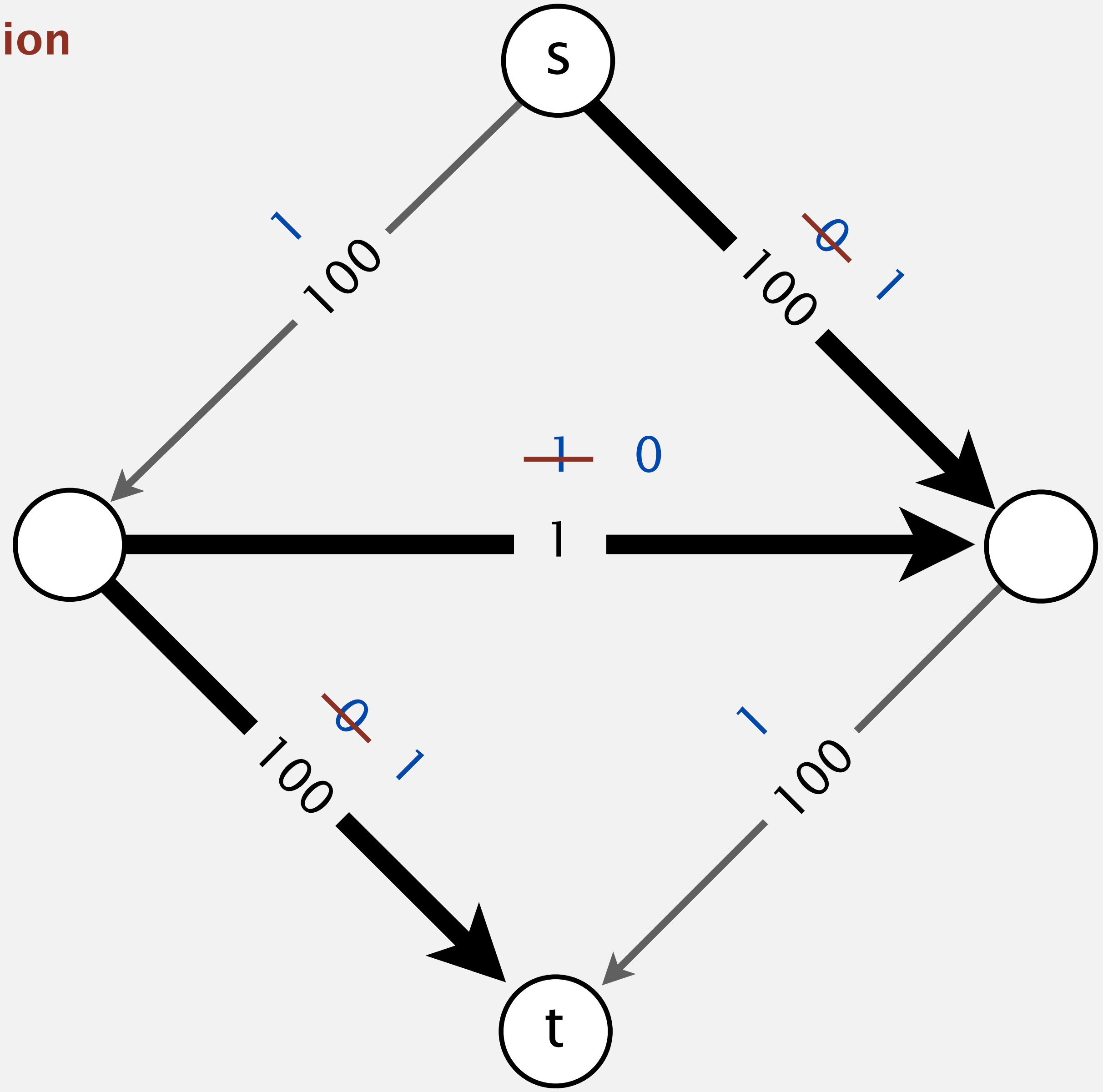
1st iteration



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.

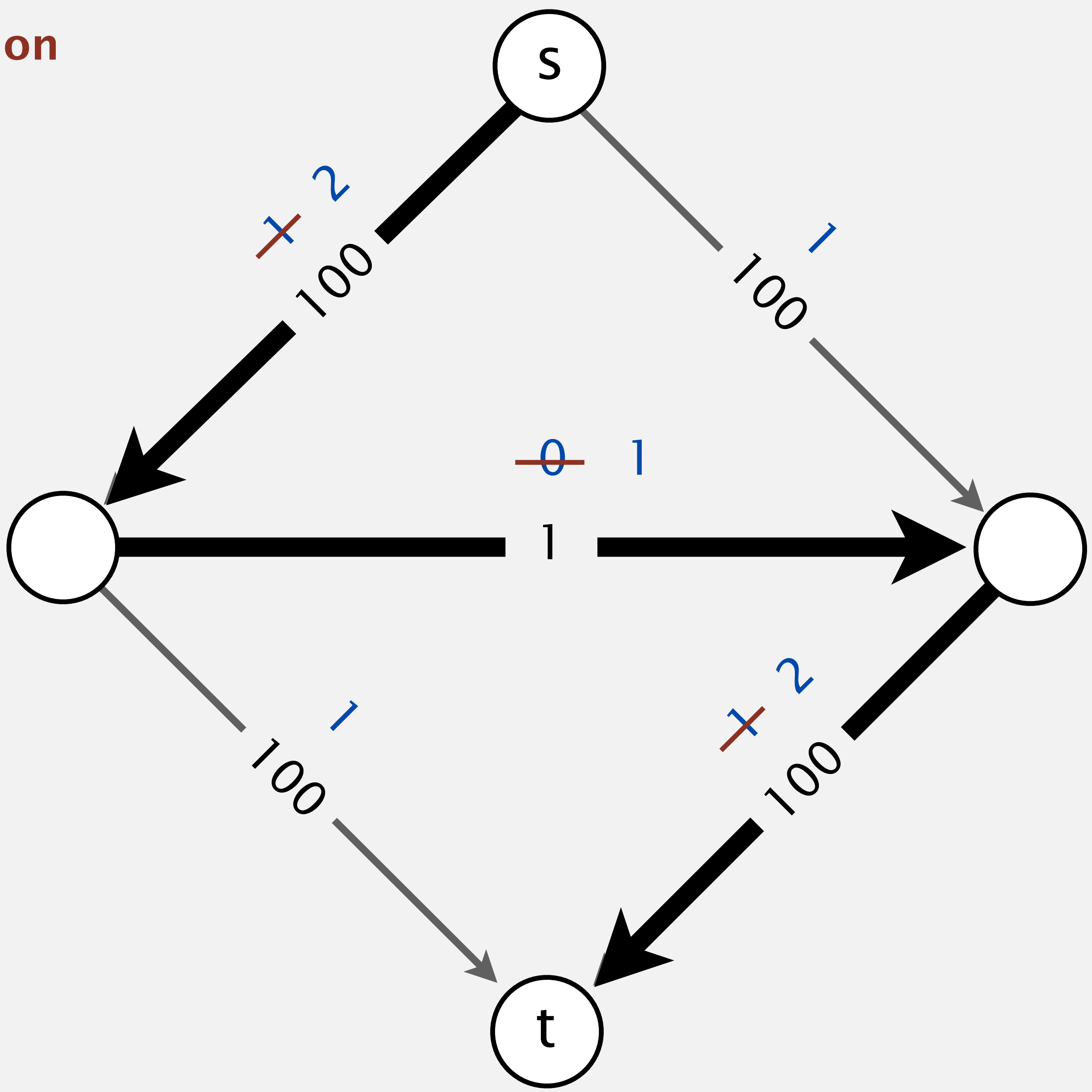
2nd iteration



Bad case for Ford–Fulkerson

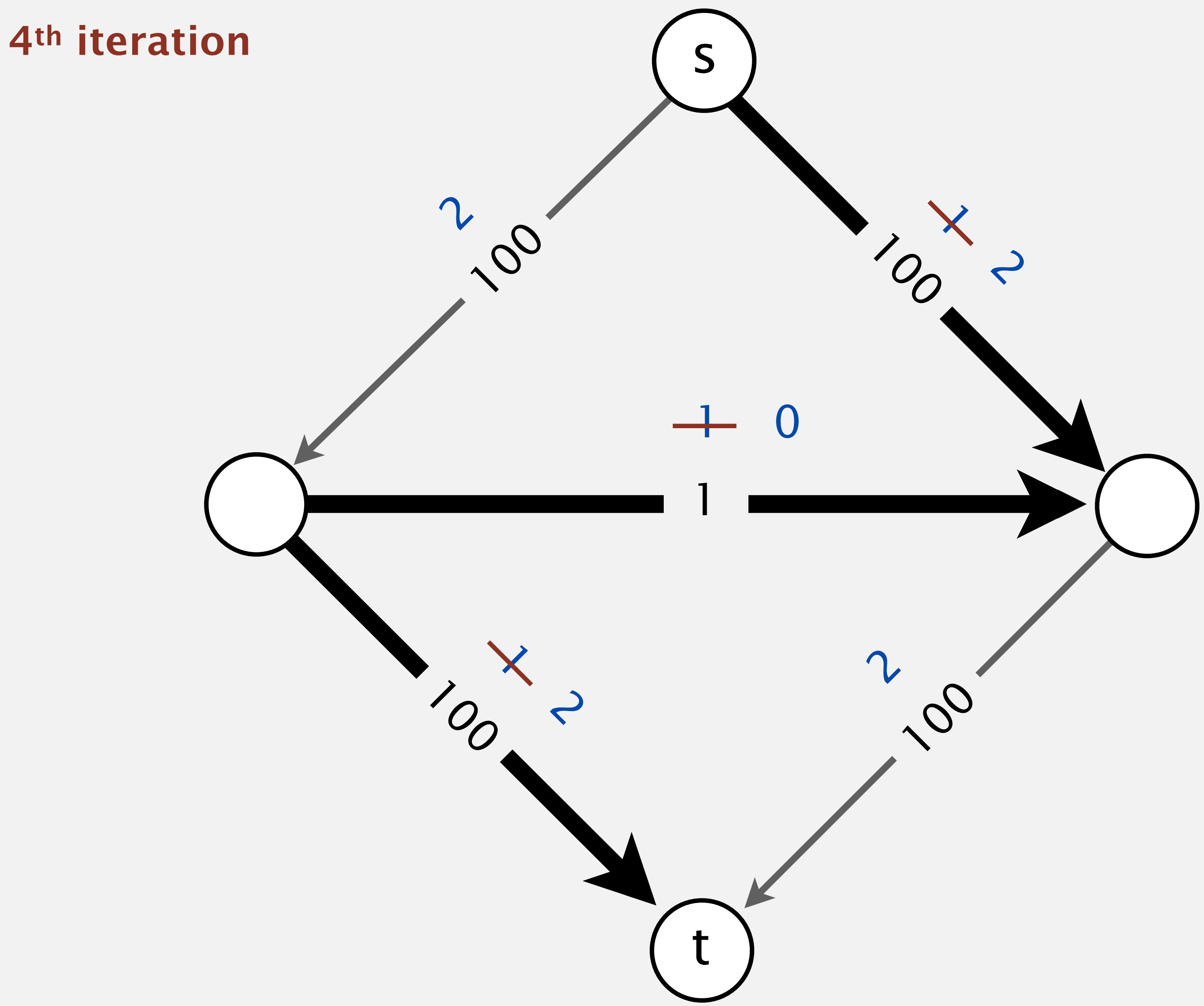
Bad news. Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maxflow.

3rd 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.



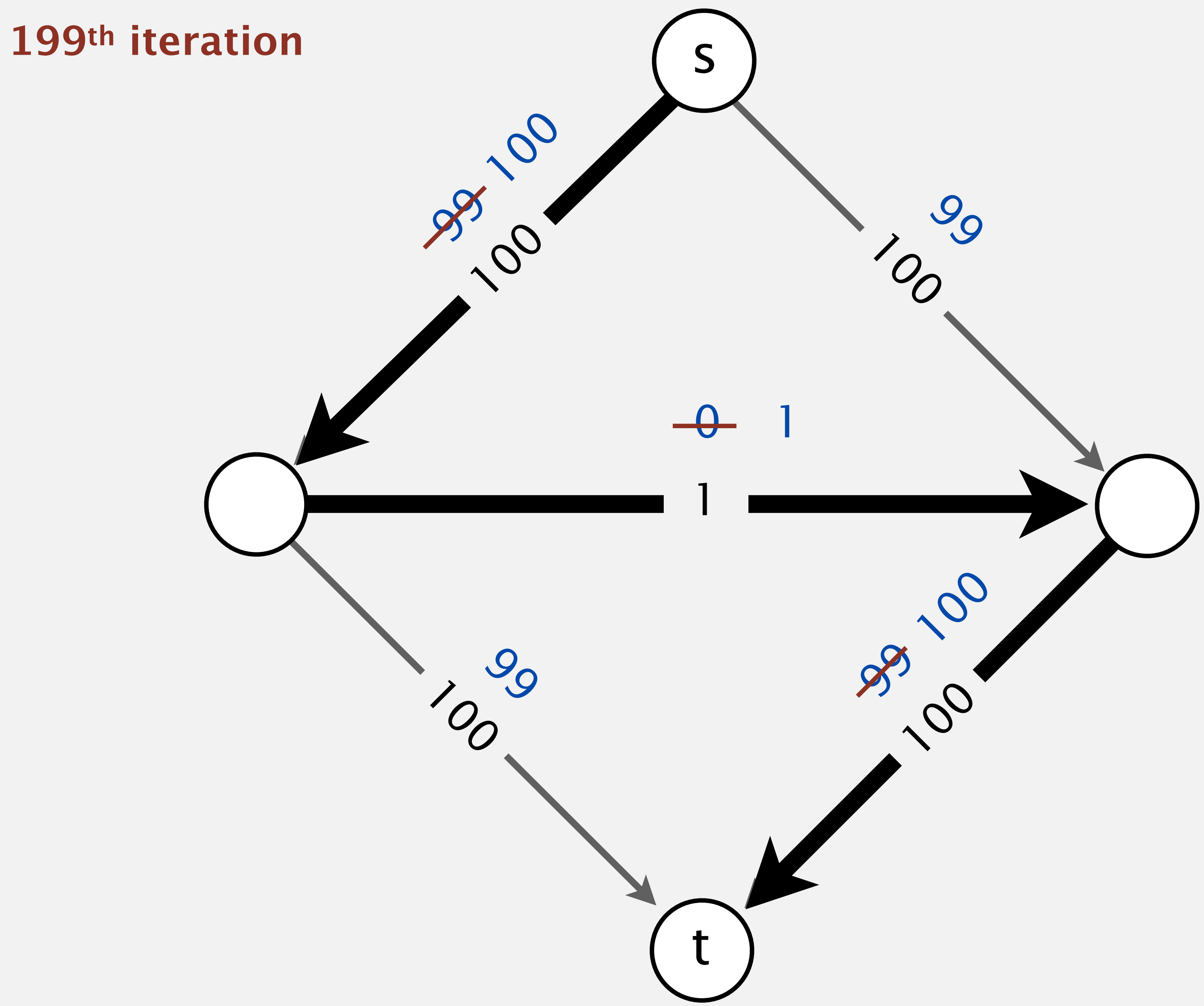
Bad case for Ford–Fulkerson

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

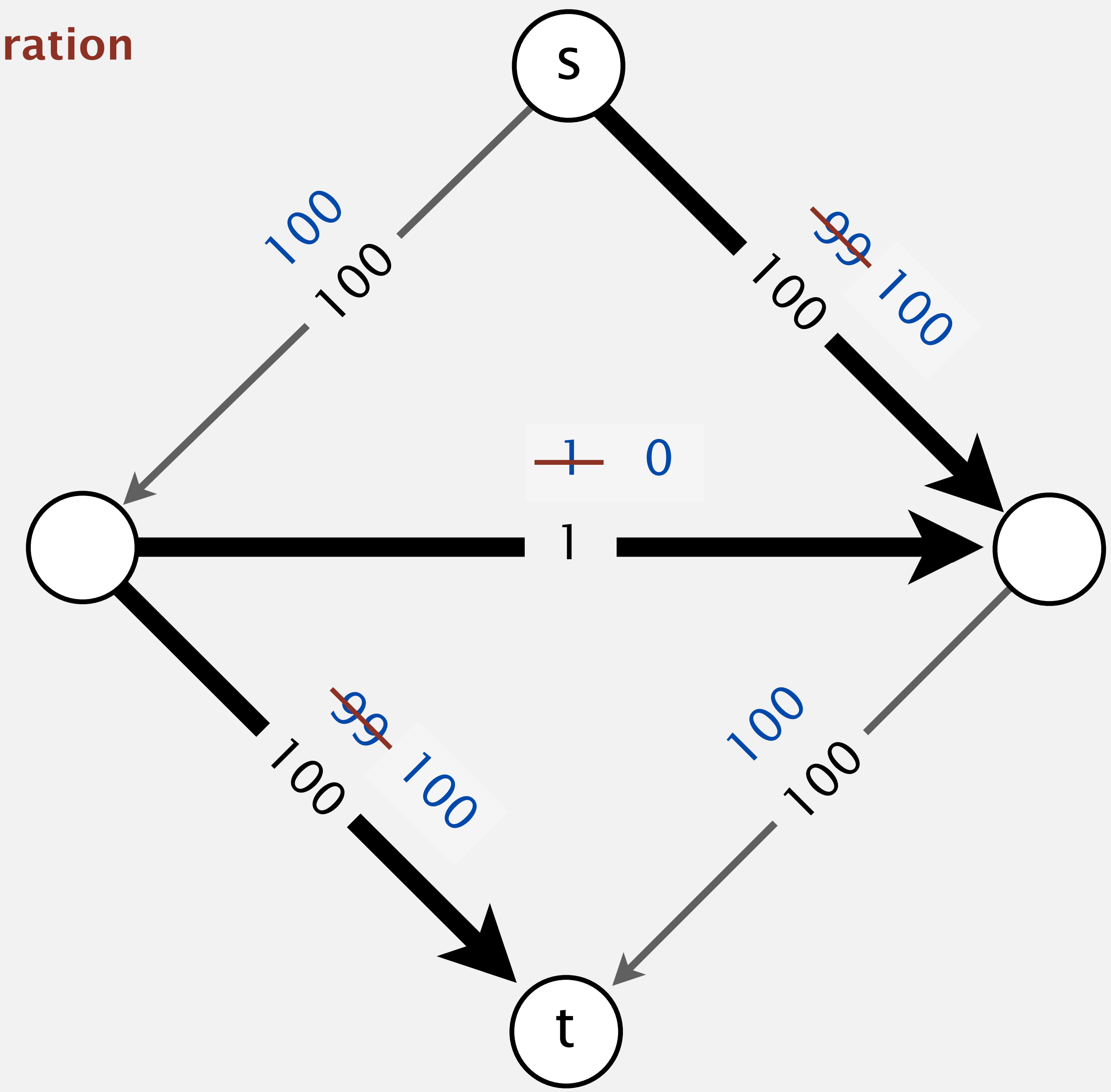
Bad news. Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maxflow.



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.

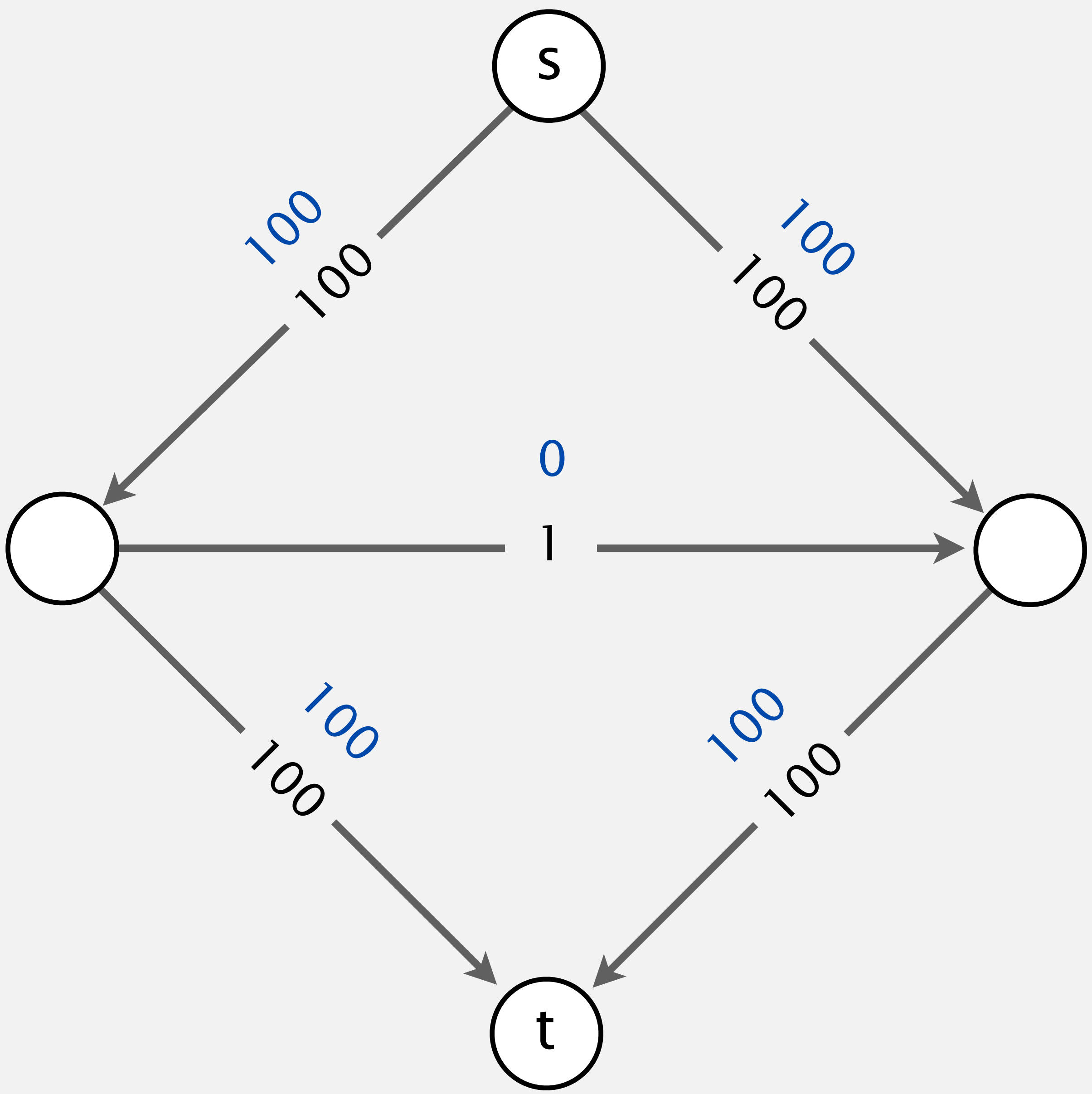
200th iteration



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



How to choose augmenting paths?

Good news. Clever choices lead to efficient algorithms.

augmenting path	number of paths	implementation
shortest path (fewest edges)	$\leq \frac{1}{2} E V$	queue (BFS)
fattest path (max bottleneck capacity)	$\leq E \ln(E U)$	priority queue

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

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)

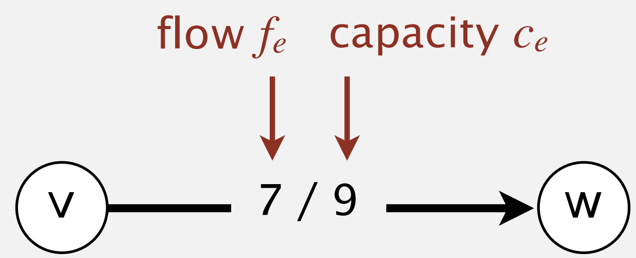
http://algs4.cs.princeton.edu

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 $e = v \rightarrow w$.



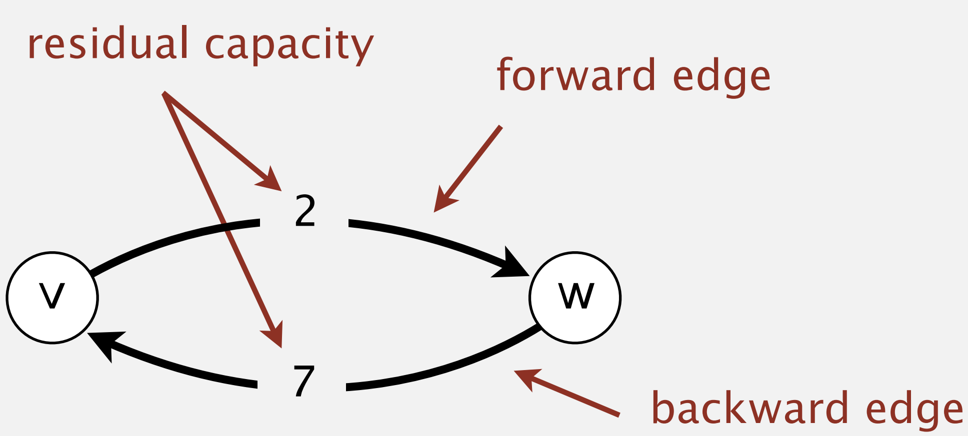
Flow network data type. Must be able to process edge $e = v \rightarrow w$ in either direction: include e in adjacency lists of both v and w.

Residual (spare) capacity.

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

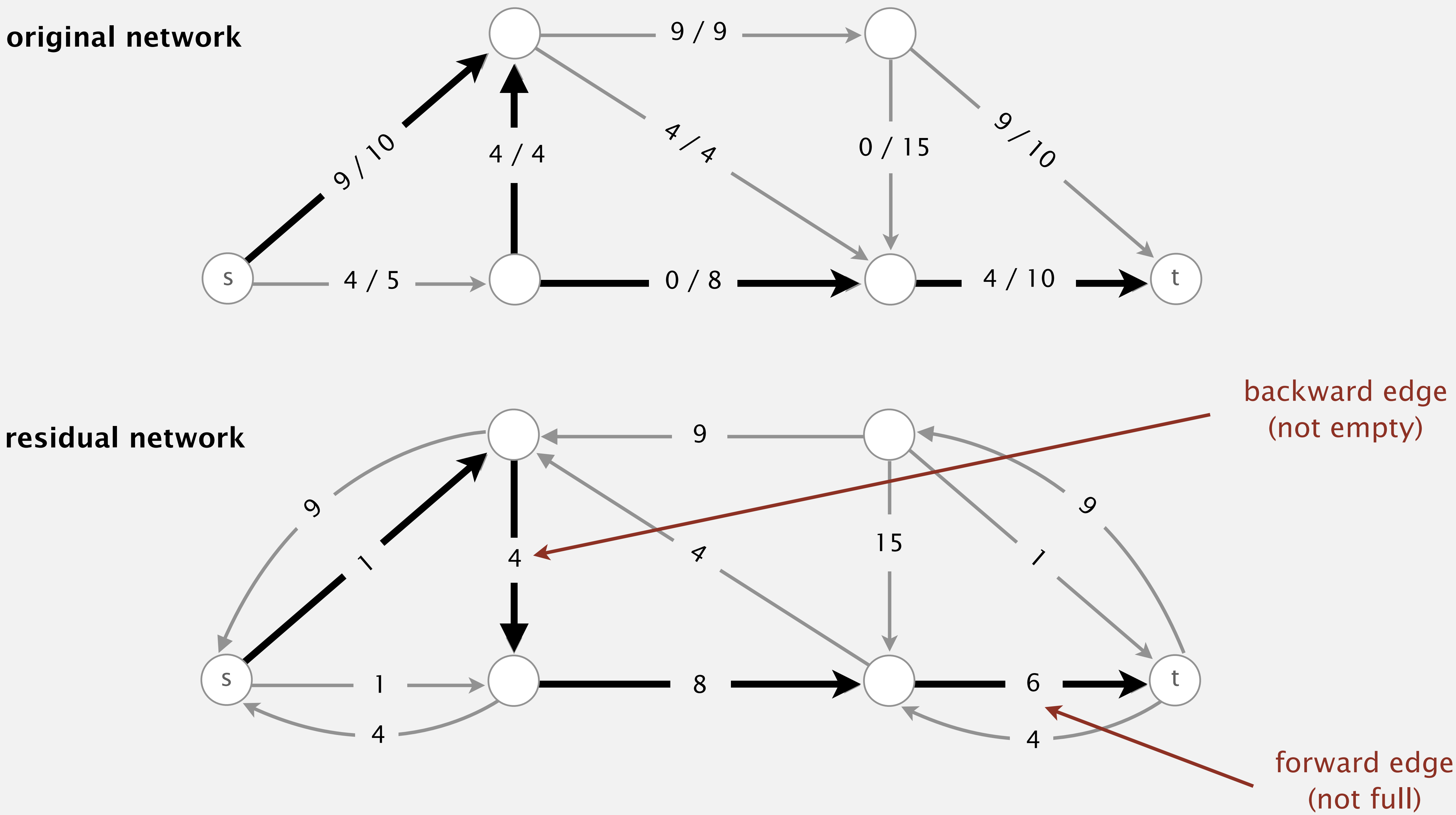
Augment flow.

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



Flow network representation

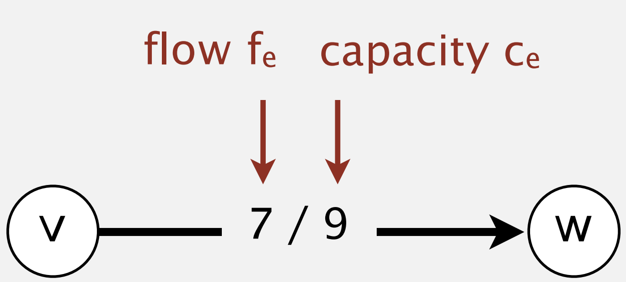
Residual network. A useful view of a flow network. ← includes all edges with positive residual capacity

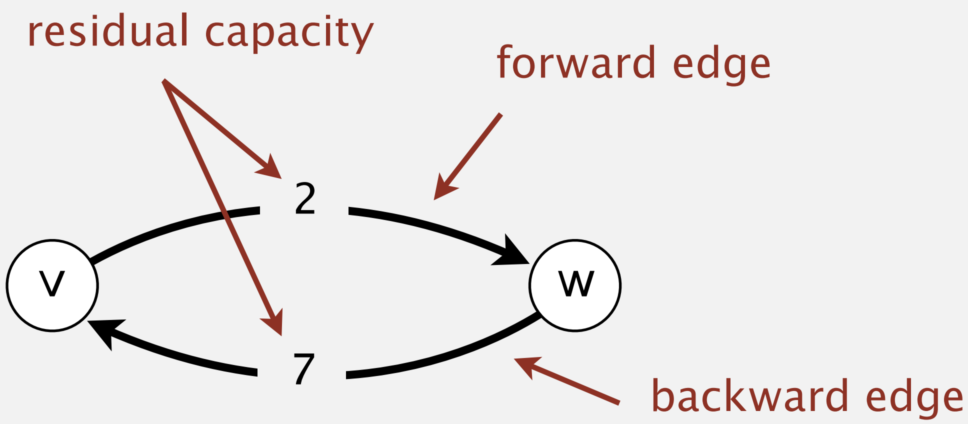


Key point. Augmenting paths in original network are in one-to-one correspondence with directed paths in residual network.

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();
}
```

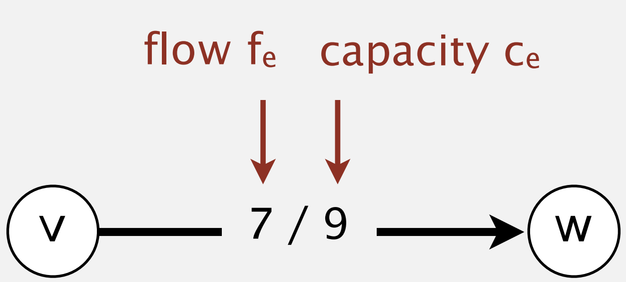
forward edge

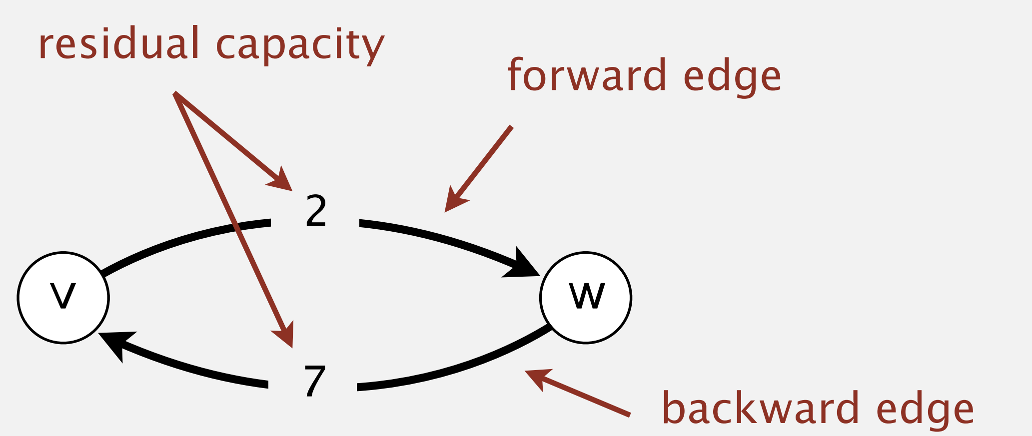
backward edge

```
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/from 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 final 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(), w = e.to();
      adj[v].add(e);
      adj[w].add(e);
   }

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

}
```

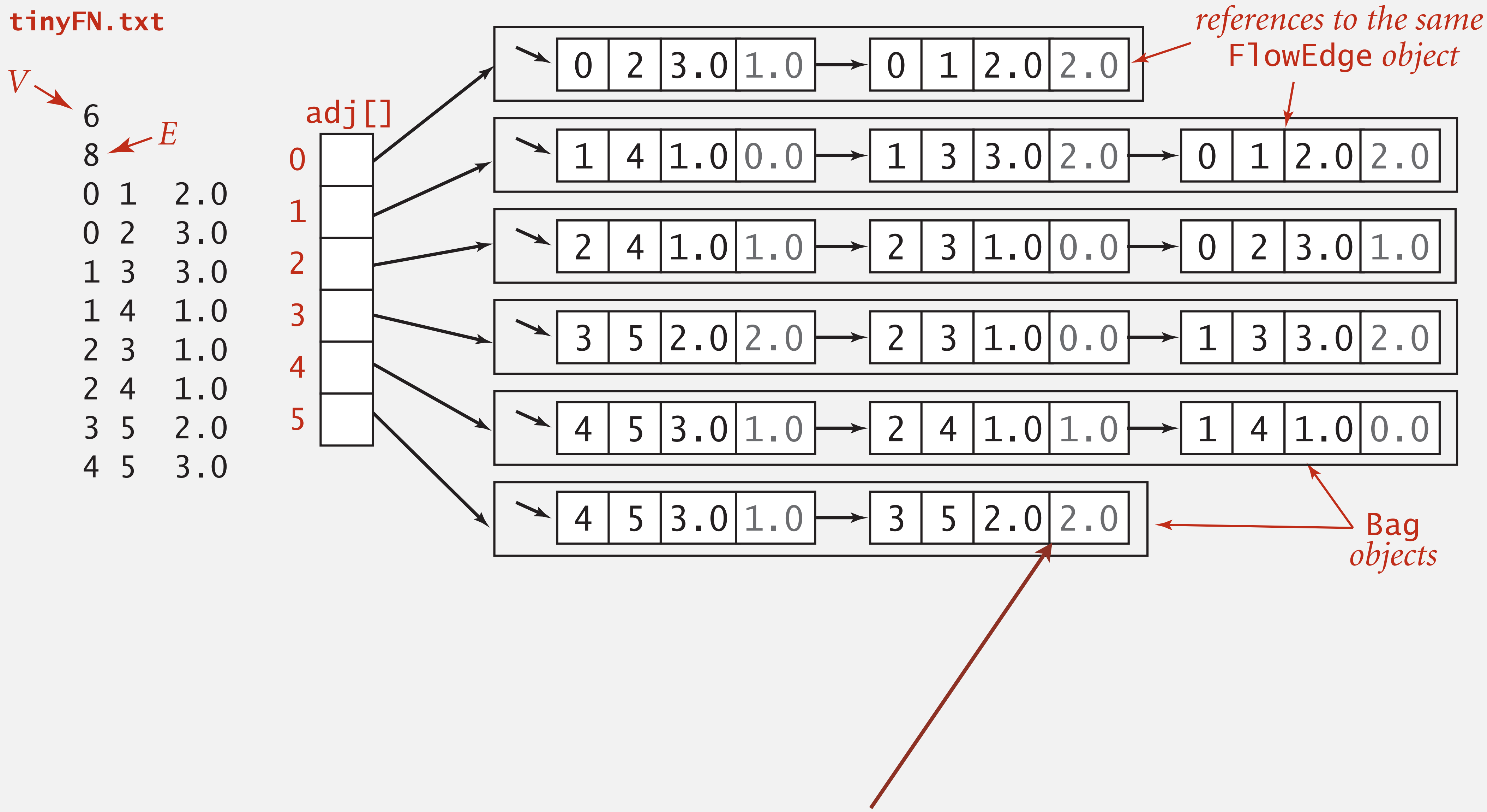
same as EdgeWeightedGraph, but adjacency lists of FlowEdges instead of Edges

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 (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() && !marked[t])
    {
        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];
}
```

can stop BFS as soon as augmenting path is discovered

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

```java
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;

   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[]` and `marked[]`

compute bottleneck capacity

augment flow

update value of flow

is v reachable from s in residual network?

http://algs4.cs.princeton.edu

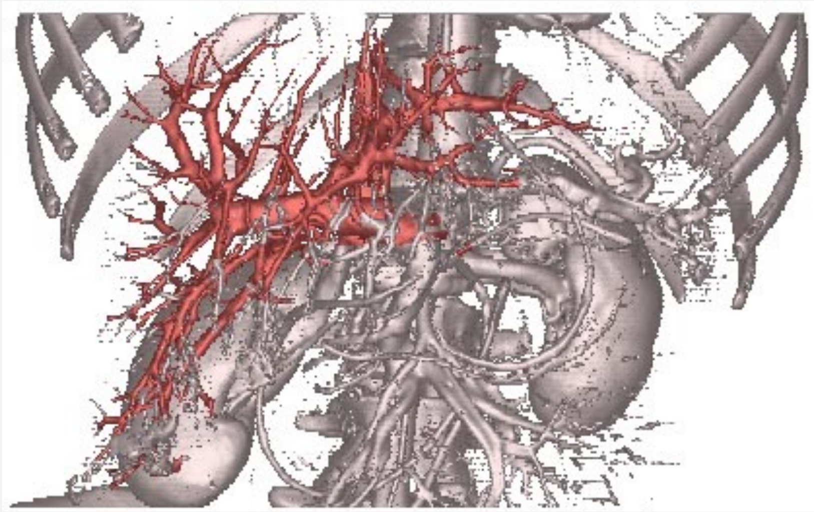
6.4 MAXIMUM FLOW

- 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

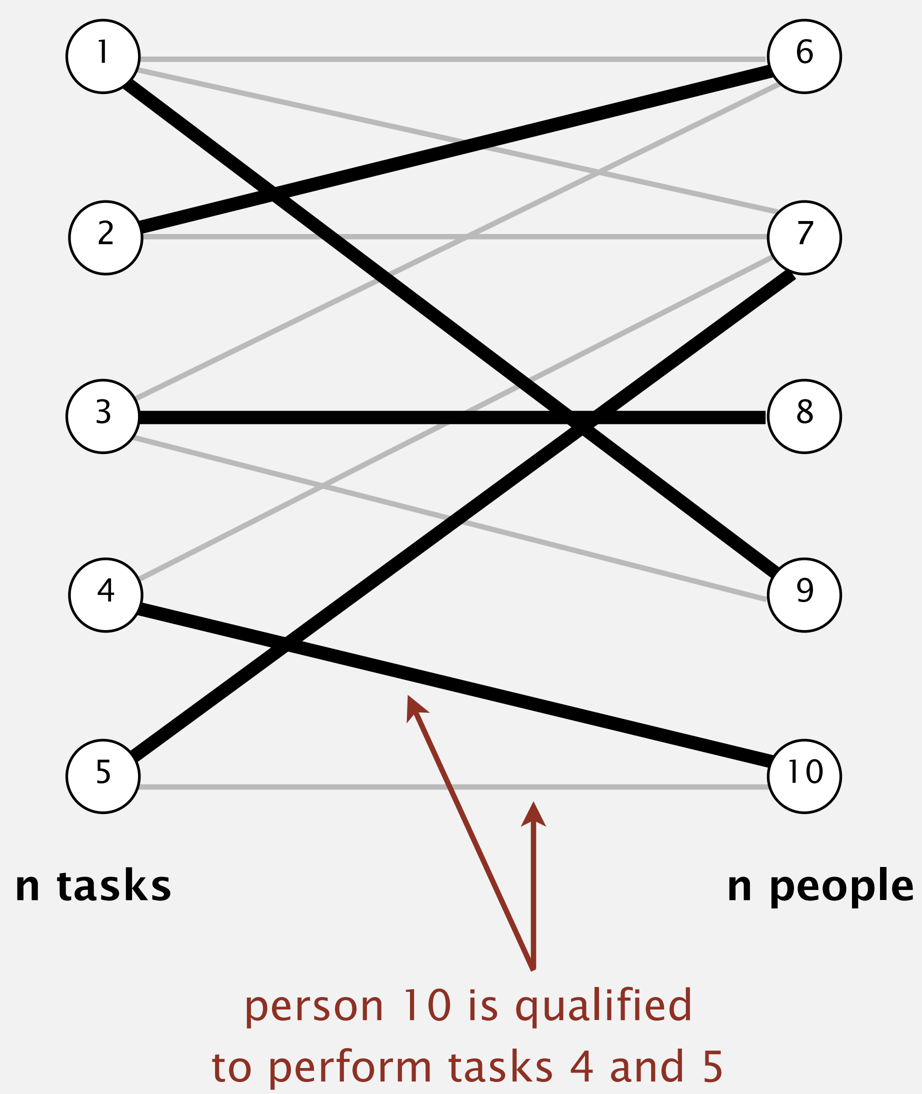
Problem. Given n people and n tasks, assign the tasks to people so that:

- Every task is assigned to a qualified person.
- Every person is assigned to exactly one task.

Bipartite matching problem

Problem. Given a bipartite graph, find a perfect matching (if one exists).

bipartite graph

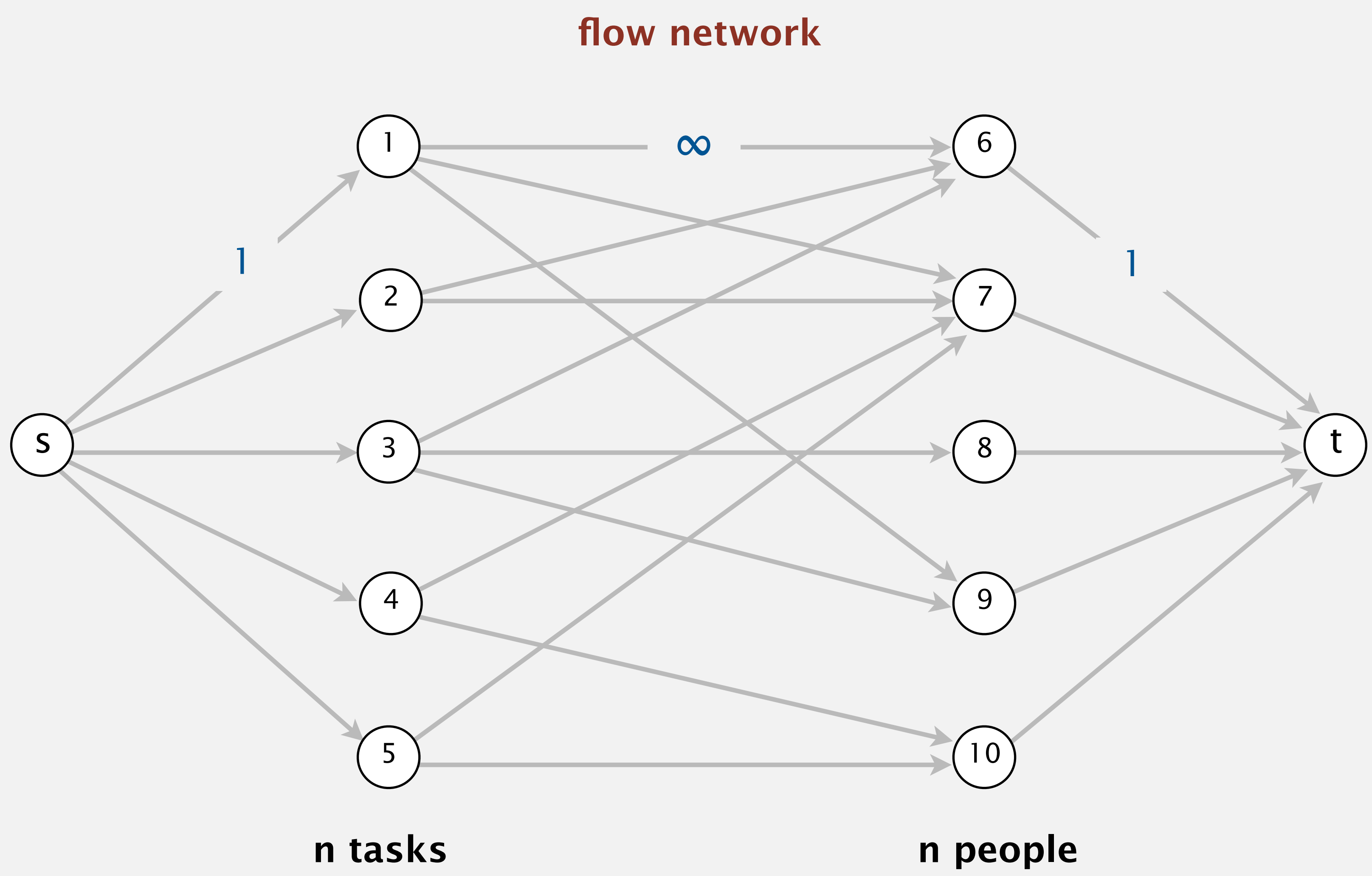


perfect matching

1–9
2–6
3–8
4–10
5–7

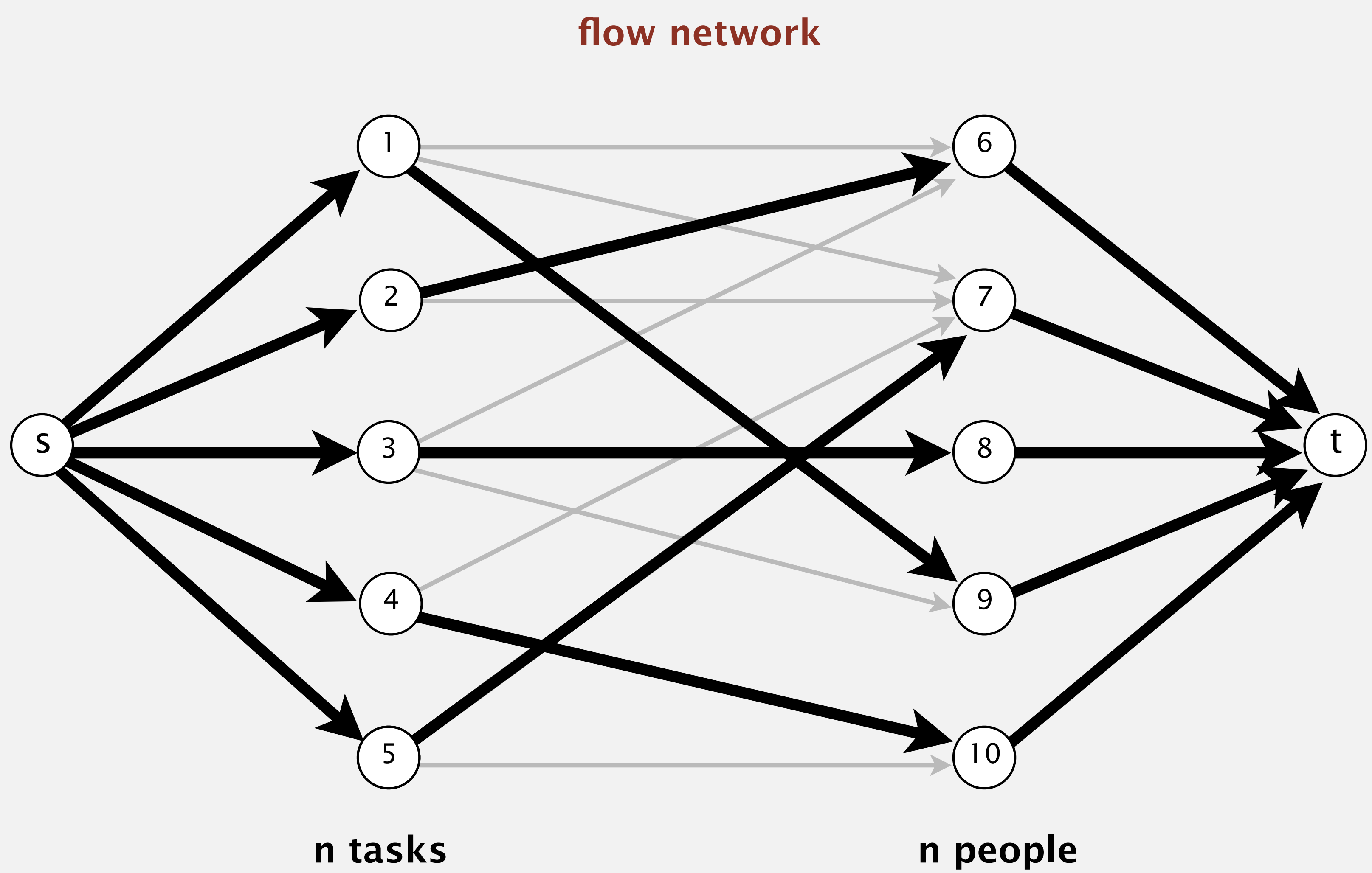
Maxflow formulation of bipartite matching

- Create s, t, one vertex for each task, and one vertex for each person.
- Add edge from s to each task (of capacity 1).
- Add edge from each person to t (of capacity 1).
- Add edge from task to qualified person (of infinite capacity).



Maxflow flow formulation of bipartite matching

1–1 correspondence between perfect matchings in bipartite graph and integer-valued flows of value n in flow network.



Maxflow: quiz 4

How many augmentations does the Ford–Fulkerson algorithms make to find a perfect matching in a bipartite graph with n vertices per side?

A. n

B. n^2

C. n^3

D. n^4

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 networks 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: $E^{3/2}$.

Computer vision. Specialized algorithms for problems with special structure.

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!