

Max Flow, Min Cut



Contents.

- Max flow.
- Min cut.
- Ford-Fulkerson augmenting path algorithm.
- Shortest augmenting path.
- Fattest augmenting path.
- Bipartite matching.

Maximum Flow and Minimum Cut

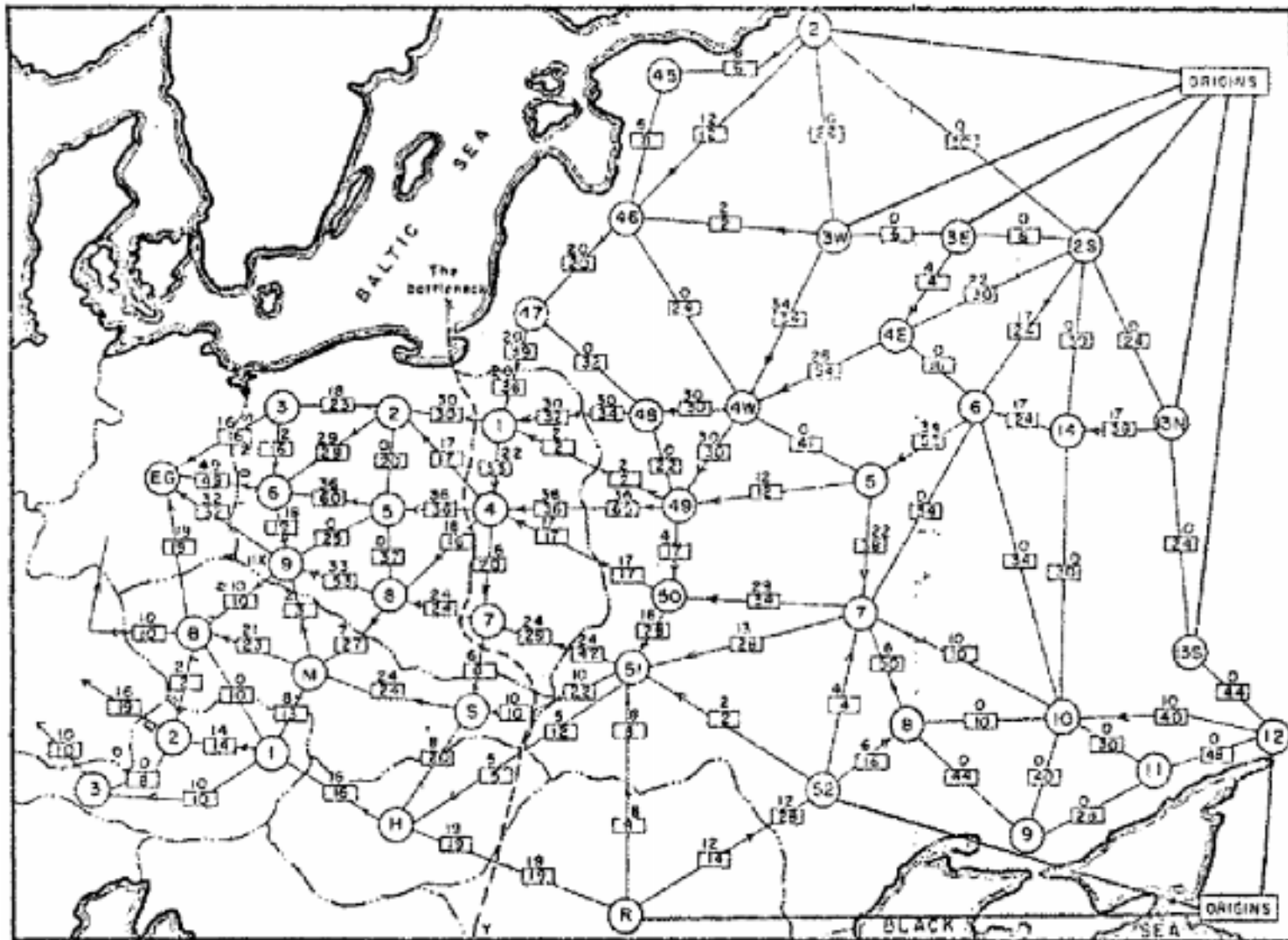
Max flow and min cut.

- Two very rich algorithmic problems.
- Cornerstone problems in combinatorial optimization.
- Beautiful mathematical duality.

Nontrivial applications / reductions.

- Network connectivity.
- Bipartite matching.
- Data mining.
- Open-pit mining.
- Airline scheduling.
- Image processing.
- Project selection.
- Baseball elimination.
- Network reliability.
- Security of statistical data.
- Distributed computing.
- Egalitarian stable matching.
- Distributed computing.
- Many many more . . .

Soviet Rail Network, 1955



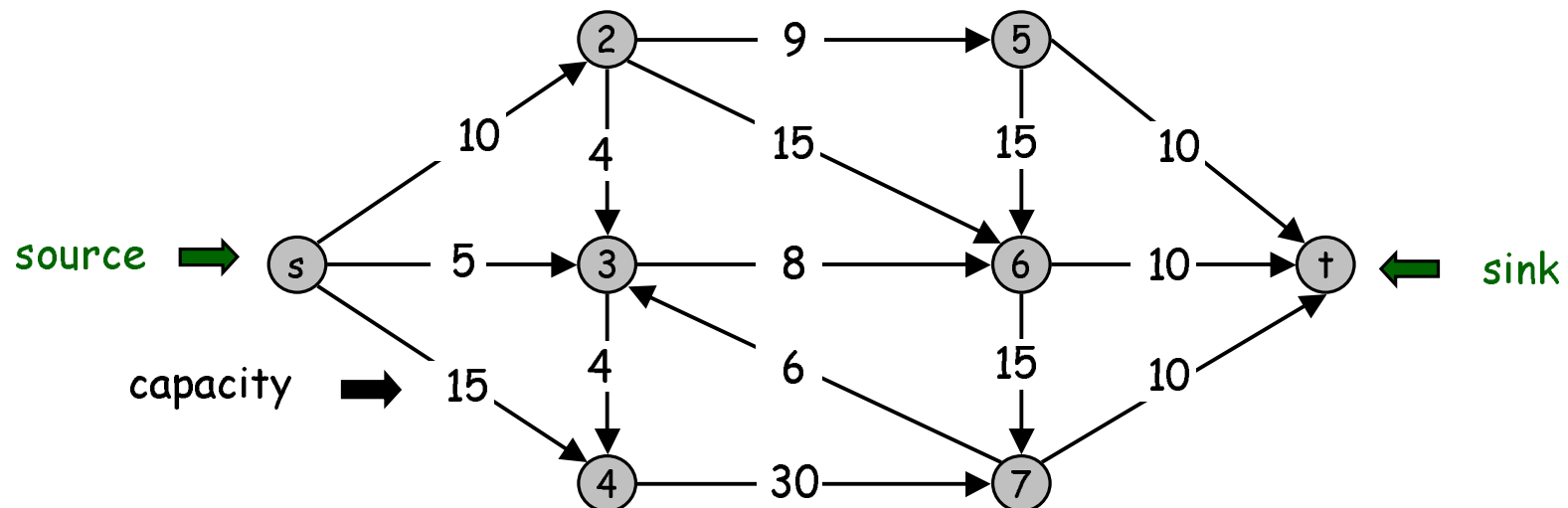
Source: *On the history of the transportation and maximum flow problems*. Alexander Schrijver in *Math Programming*, 91: 3, 2002.

Minimum Cut Problem

Network: abstraction for material FLOWING through the edges.

- Directed graph.
- Capacities on edges.
- Source node s , sink node t .

Min cut problem. Delete edges to disconnect s from t .

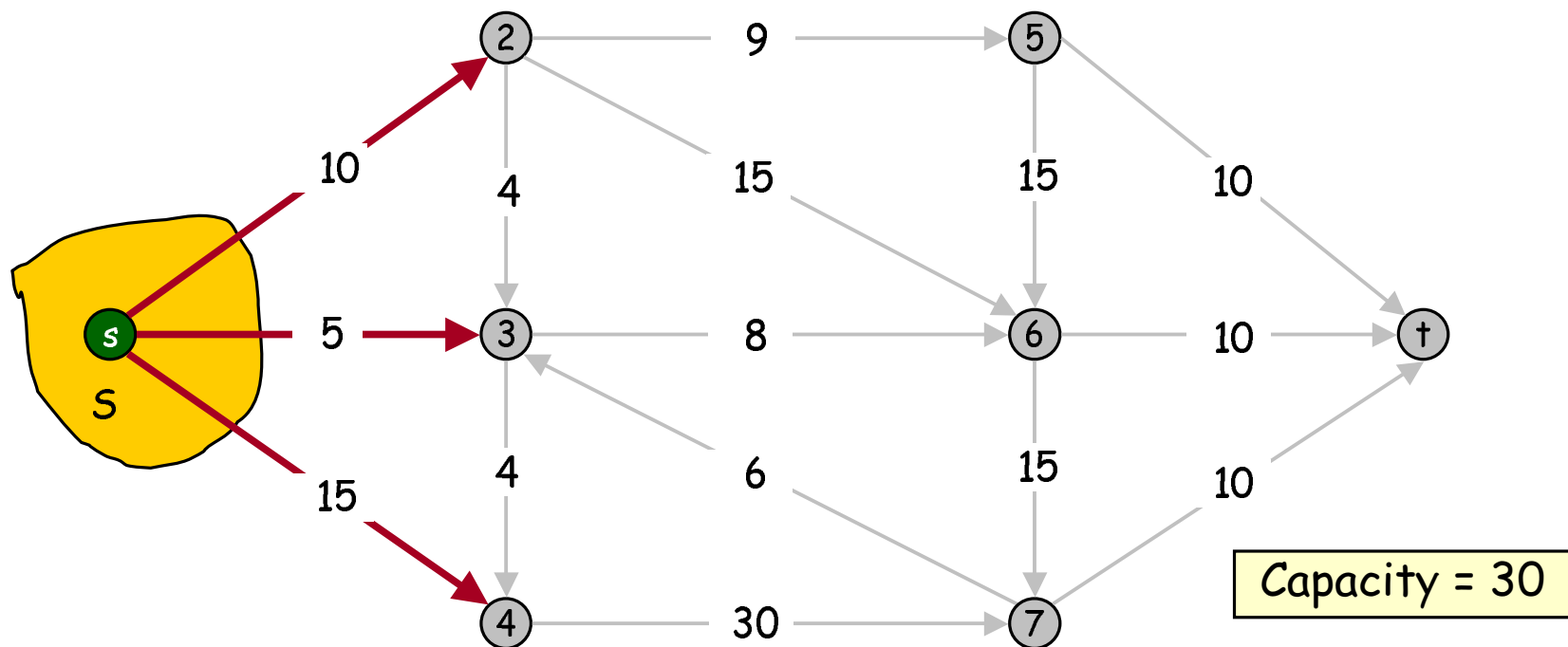


Cuts

A cut is a node partition (S, T) such that s is in S and t is in T .

- $\text{capacity}(S, T) = \text{sum of weights of edges leaving } S$.

Minimum s - t cut problem. Find an s - t cut of minimum capacity.

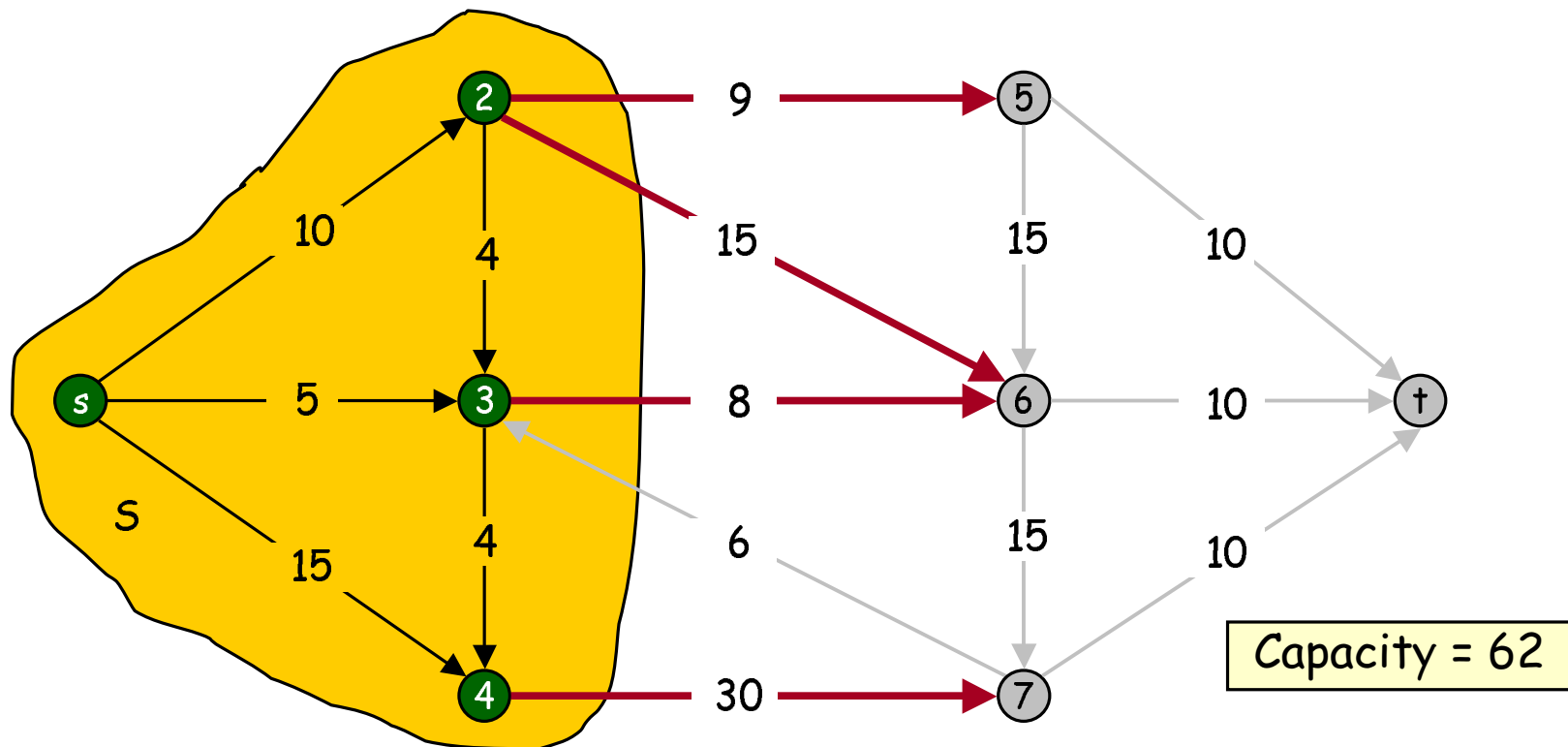


Cuts

A cut is a node partition (S, T) such that s is in S and t is in T .

- $\text{capacity}(S, T) = \text{sum of weights of edges leaving } S$.

Minimum s - t cut problem. Find an s - t cut of minimum capacity.

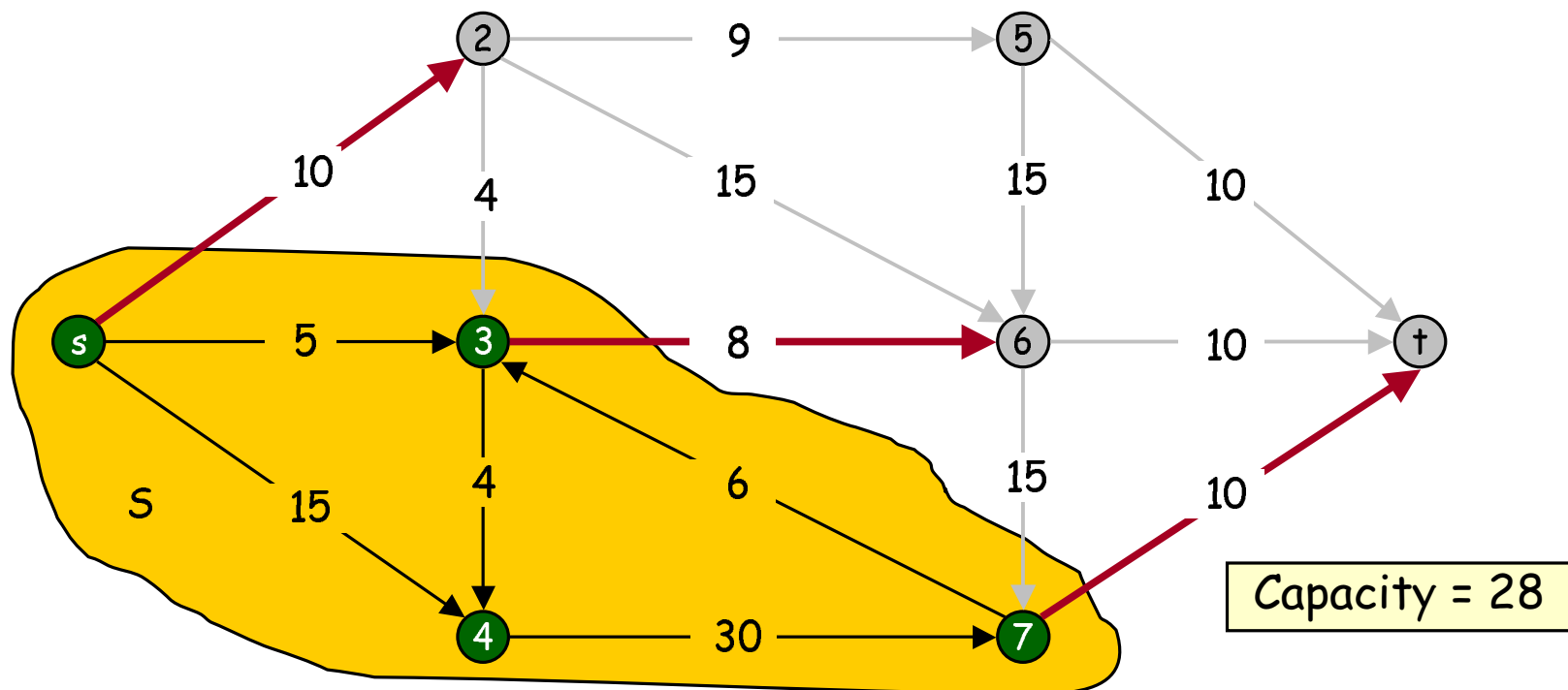


Cuts

A cut is a node partition (S, T) such that s is in S and t is in T .

- $\text{capacity}(S, T) = \text{sum of weights of edges leaving } S$.

Minimum s - t cut problem. Find an s - t cut of minimum capacity.



Maximum Flow Problem

Network: abstraction for material FLOWING through the edges.

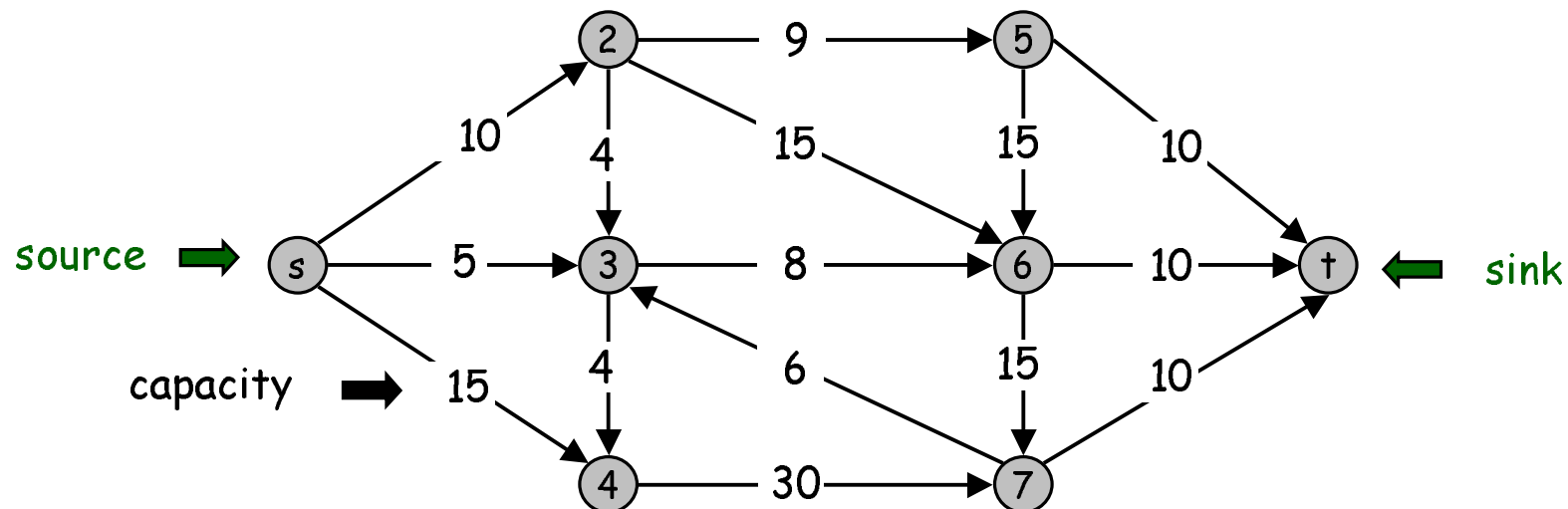
- Directed graph.
- Capacities on edges.
- Source node s , sink node t .

← exactly as for min cut problem

Max flow problem. Assign flow to edges so that:

- Equalizes inflow and outflow at every intermediate vertex.
- Maximizes flow sent from s to t .

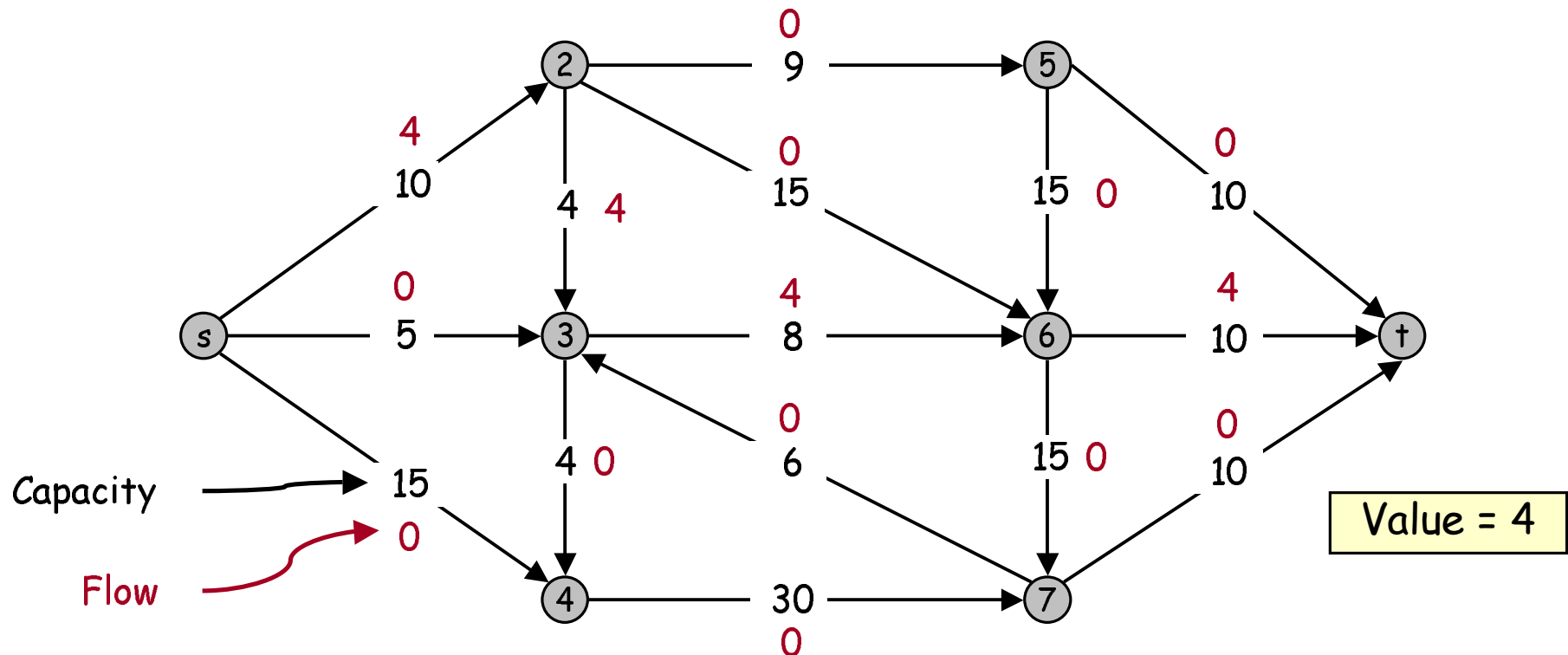
← not s or t



Flows

A flow f is an assignment of weights to edges so that:

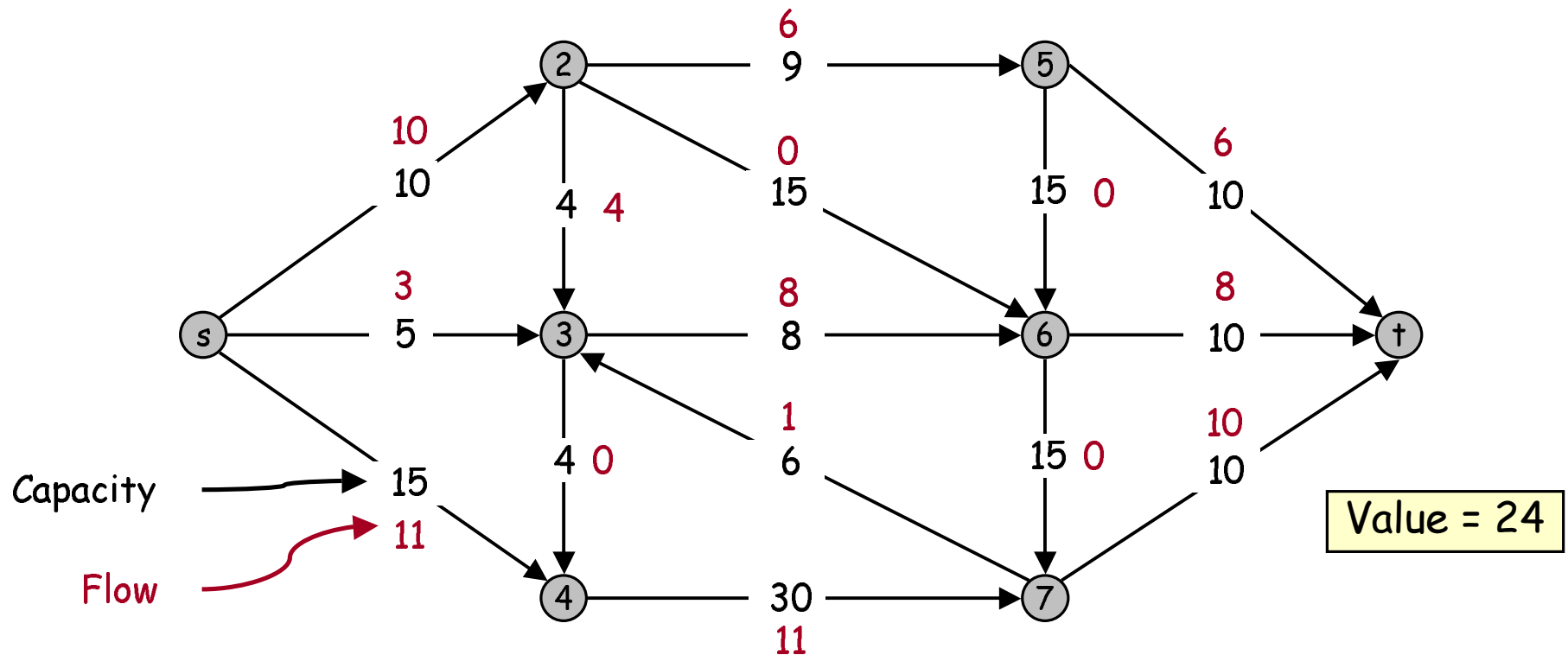
- Capacity: $0 \leq f(e) \leq u(e)$.
- Flow conservation: flow leaving v = flow entering v .



Flows

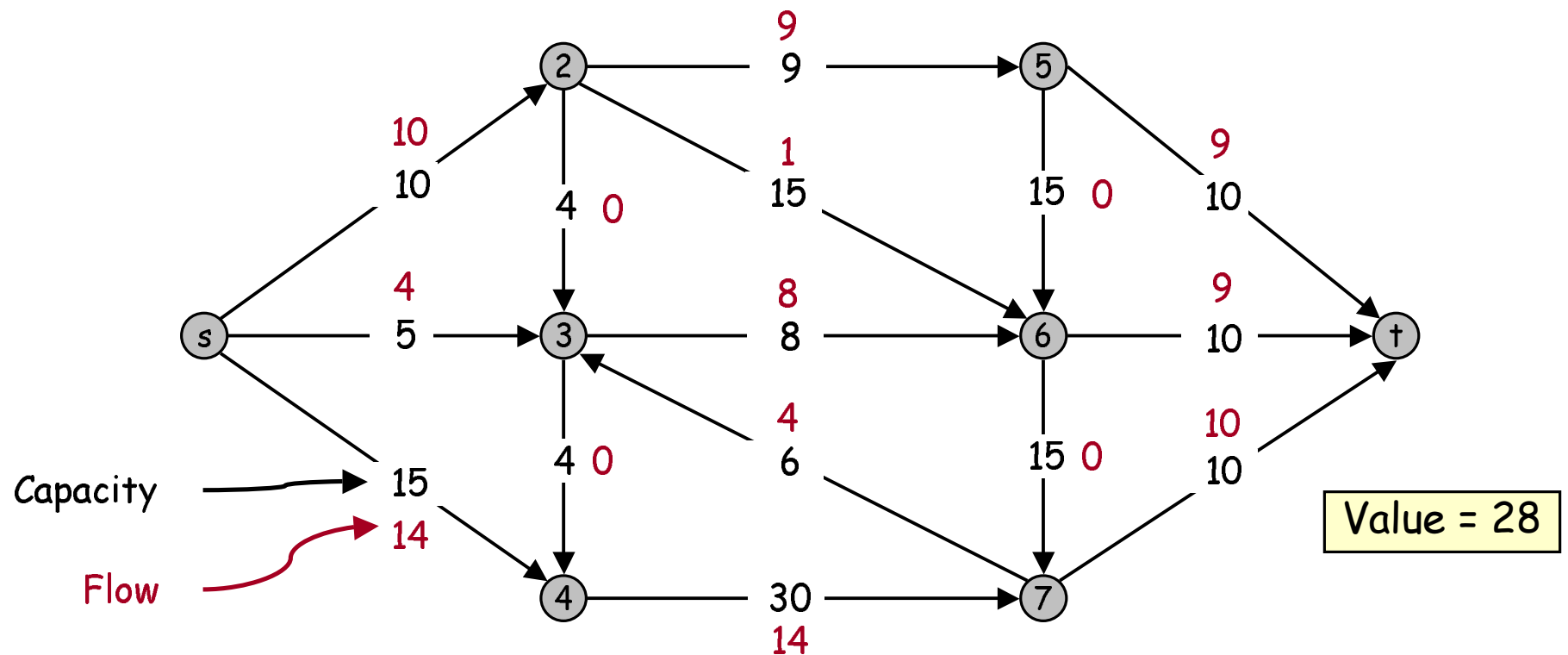
A flow f is an assignment of weights to edges so that:

- Capacity: $0 \leq f(e) \leq u(e)$.
- Flow conservation: flow leaving v = flow entering v .



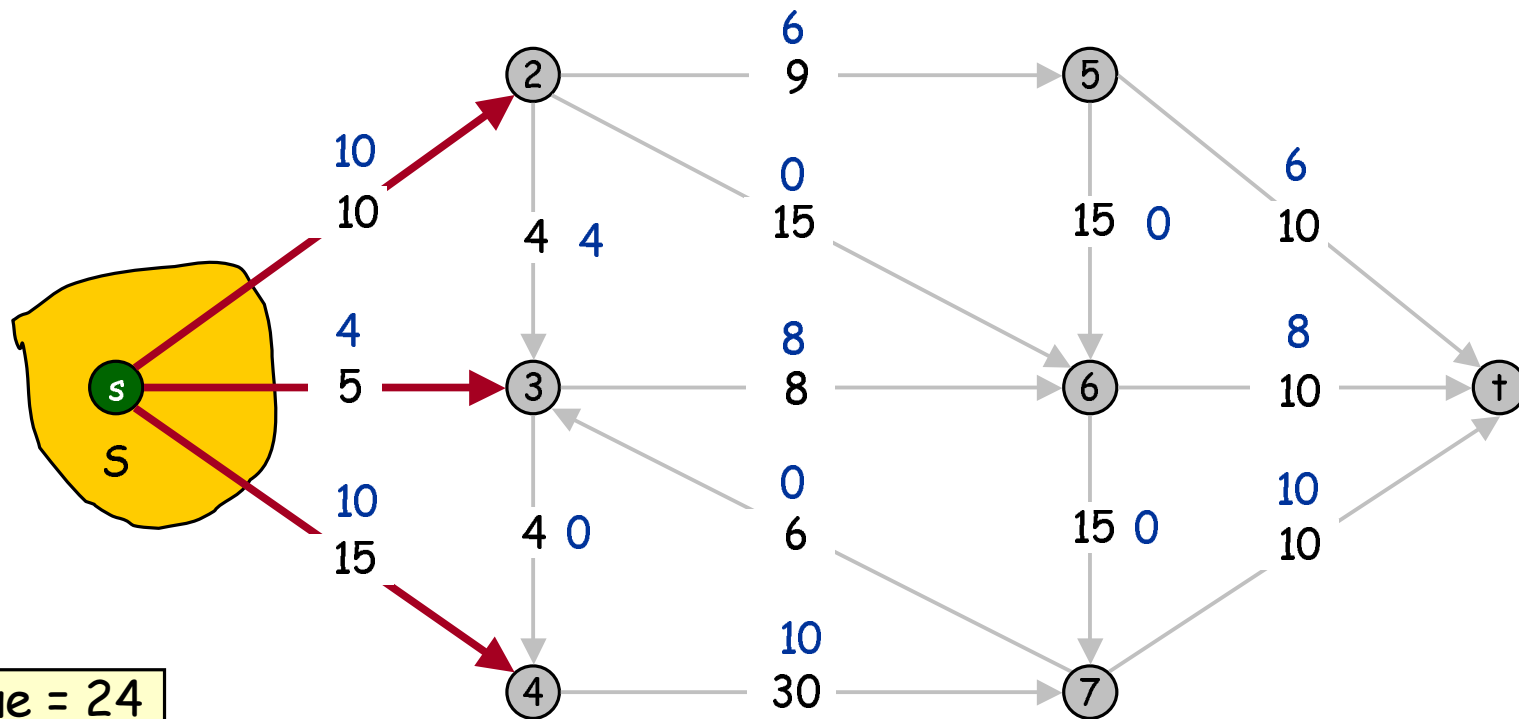
Flows

Max flow problem: find flow that maximizes net flow into sink.



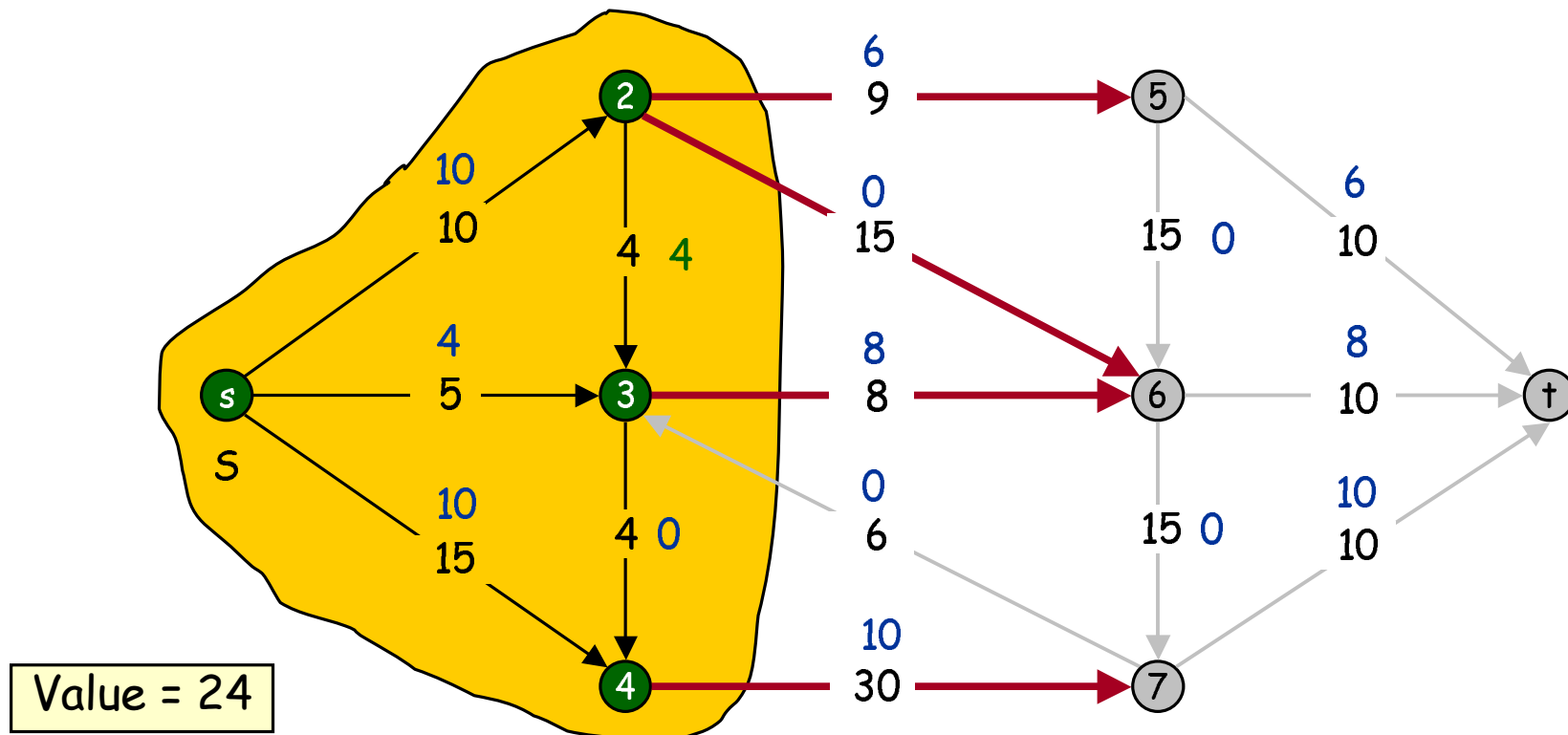
Flows and Cuts

Observation 1. Let f be a flow, and let (S, T) be any cut. Then, the net flow sent across the cut is equal to the amount reaching t .



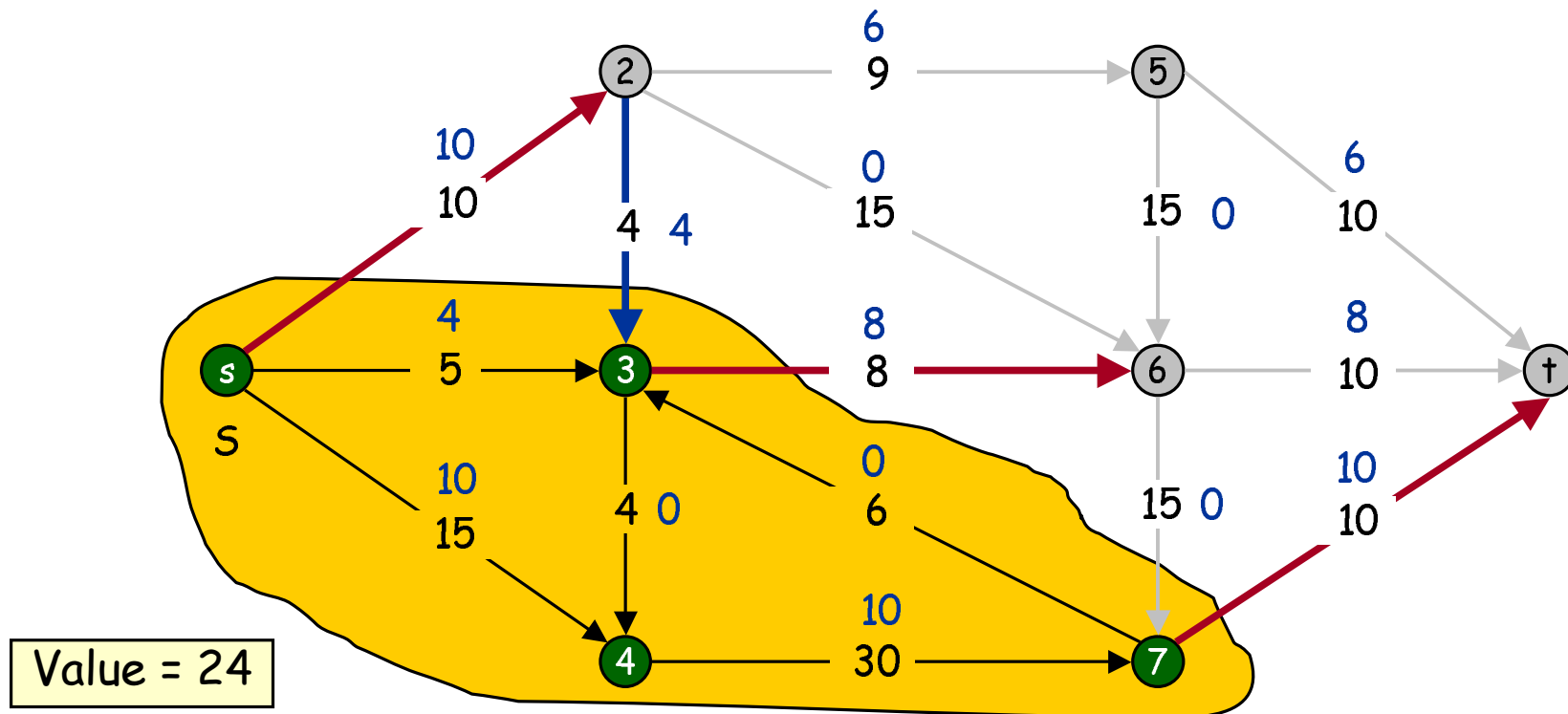
Flows and Cuts

Observation 1. Let f be a flow, and let (S, T) be any cut. Then, the net flow sent across the cut is equal to the amount reaching t .



Flows and Cuts

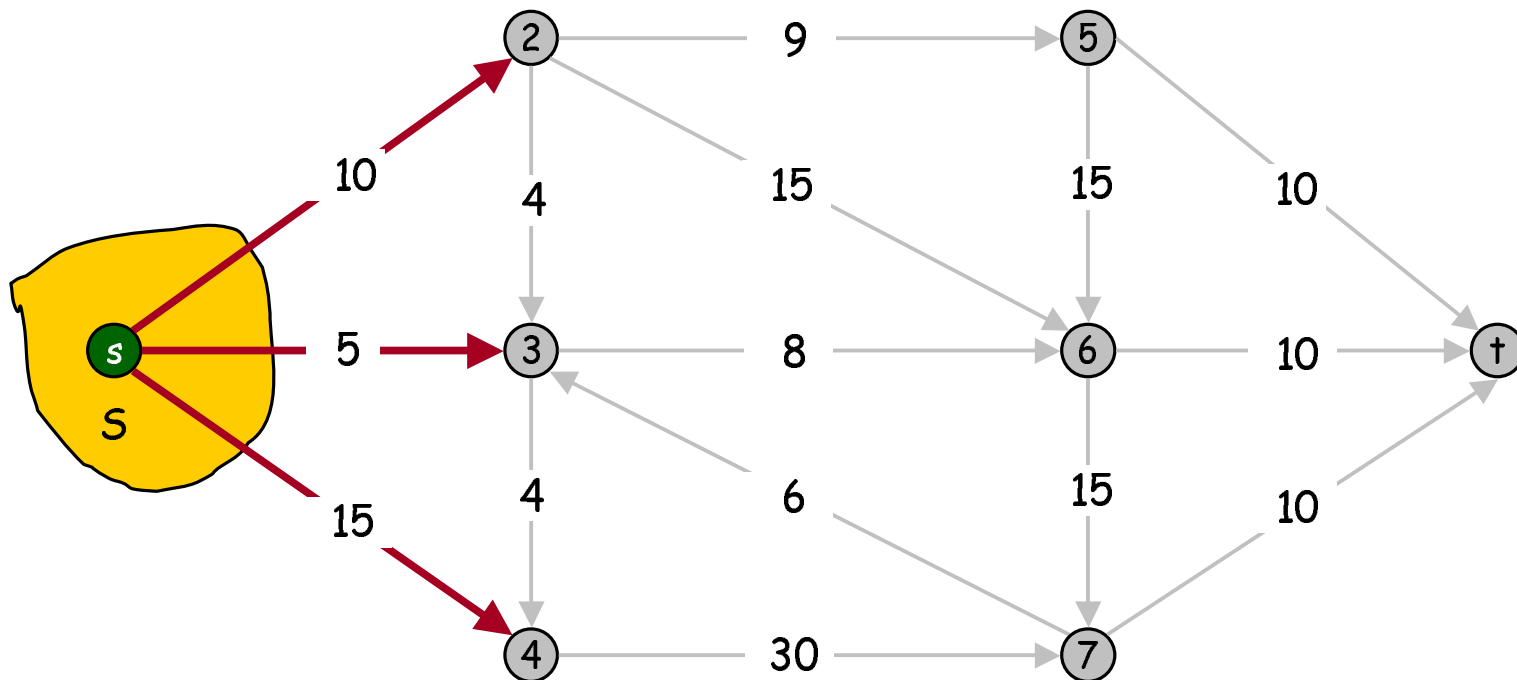
Observation 1. Let f be a flow, and let (S, T) be any cut. Then, the net flow sent across the cut is equal to the amount reaching t .



Flows and Cuts

Observation 2. Let f be a flow, and let (S, T) be any cut. Then the value of the flow is at most the capacity of the cut.

Cut capacity = 30 \Rightarrow Flow value ≤ 30

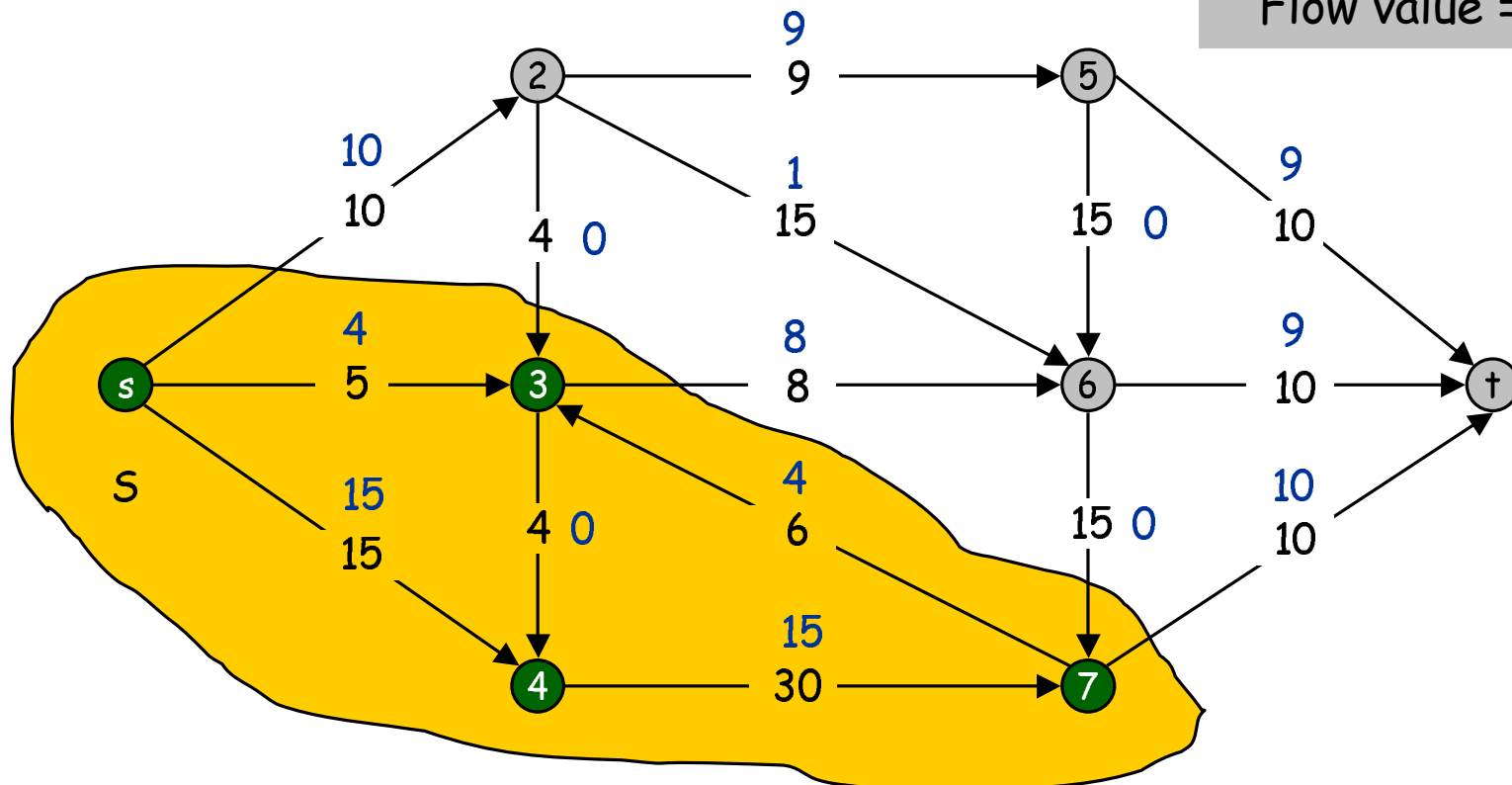


Max Flow and Min Cut

Observation 3. Let f be a flow, and let (S, T) be a cut whose capacity equals the value of f . Then f is a max flow and (S, T) is a min cut.

Cut capacity = 28 \Rightarrow Flow value ≤ 28

Flow value = 28

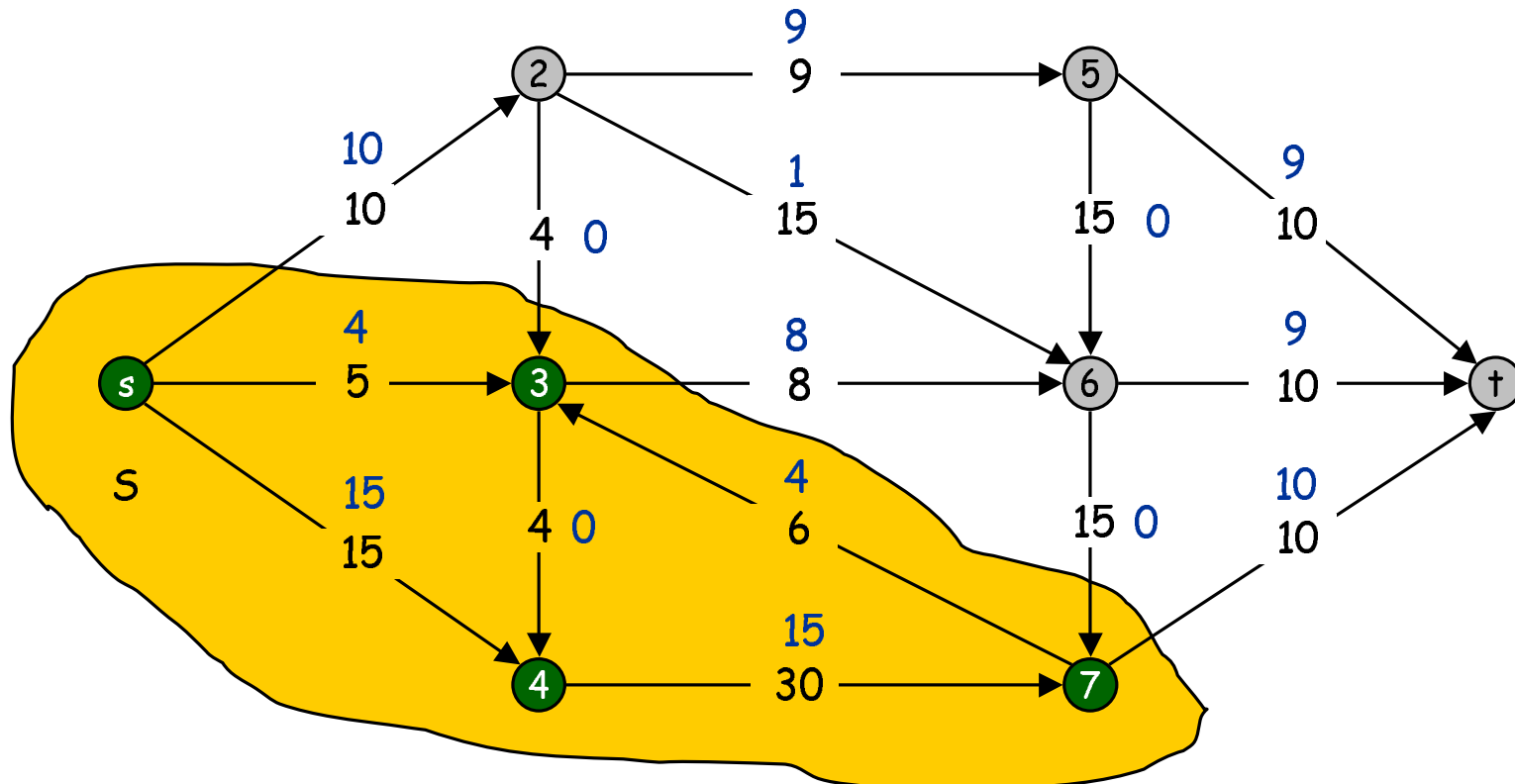


Max-Flow Min-Cut Theorem

MAX-FLOW MIN-CUT THEOREM (Ford-Fulkerson, 1956): In any network, the value of max flow is equal to the capacity of min cut.

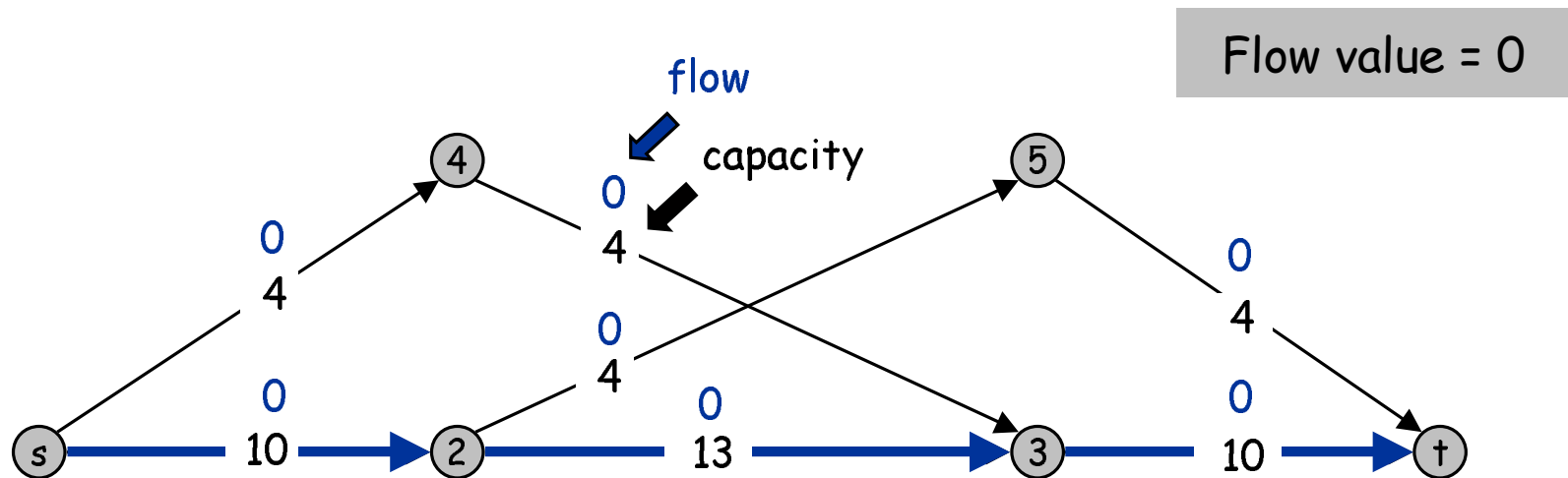
- Proof IOU: we find flow and cut such that Observation 3 applies.

Min cut capacity = 28 \Leftrightarrow Max flow value = 28



Towards an Algorithm

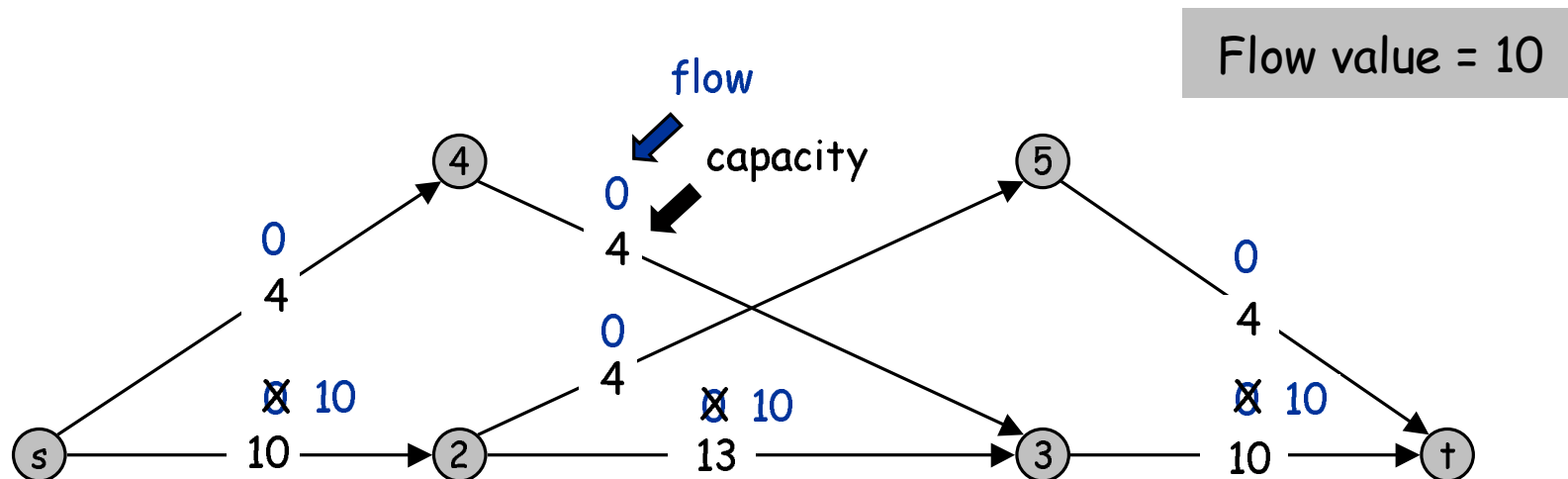
Find s - t path where each arc has $f(e) < u(e)$ and "augment" flow along it.



Towards an Algorithm

Find s - t path where each arc has $f(e) < u(e)$ and "augment" flow along it.

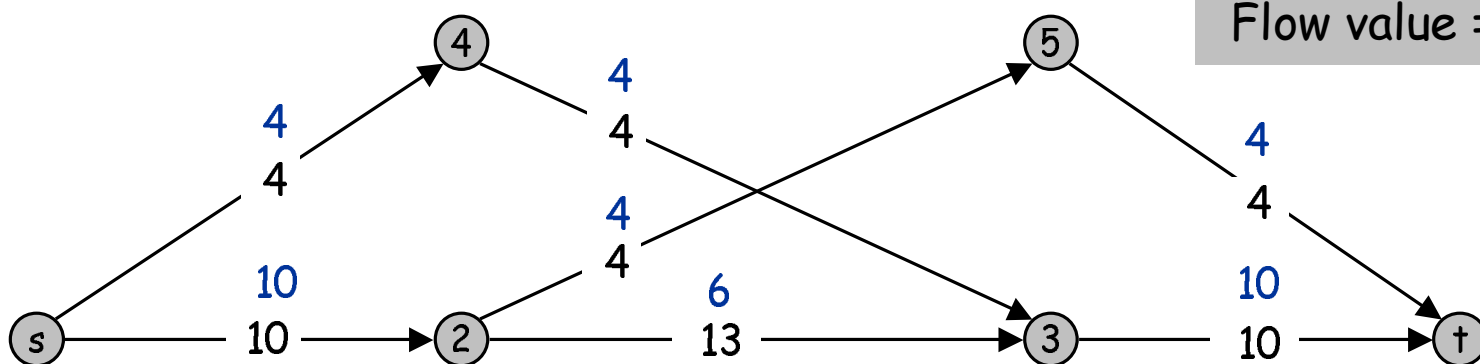
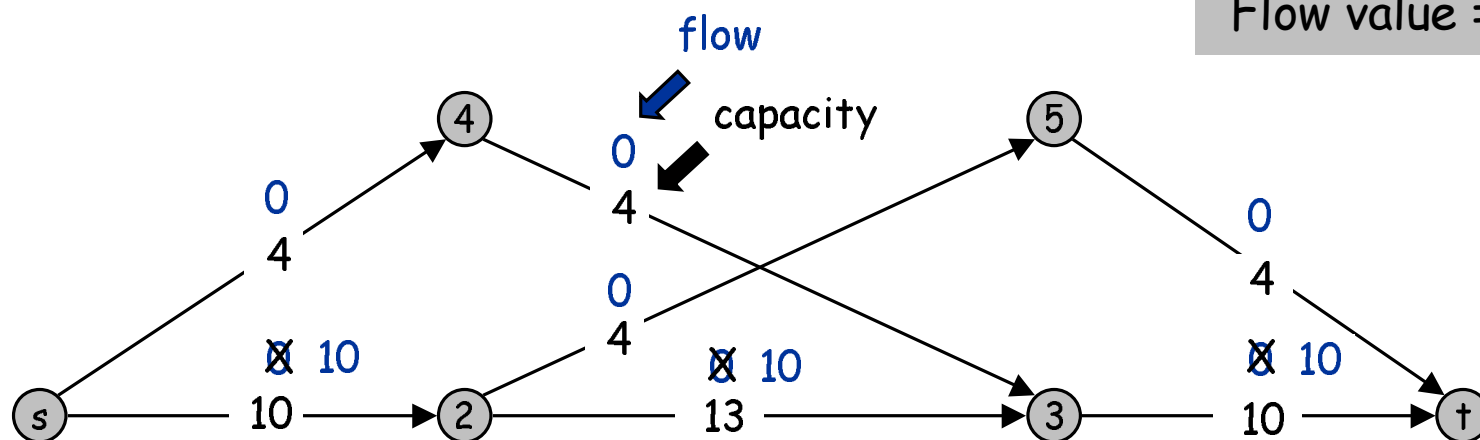
- Greedy algorithm: repeat until you get stuck.



Towards an Algorithm

Find s - t path where each arc has $f(e) < u(e)$ and "augment" flow along it.

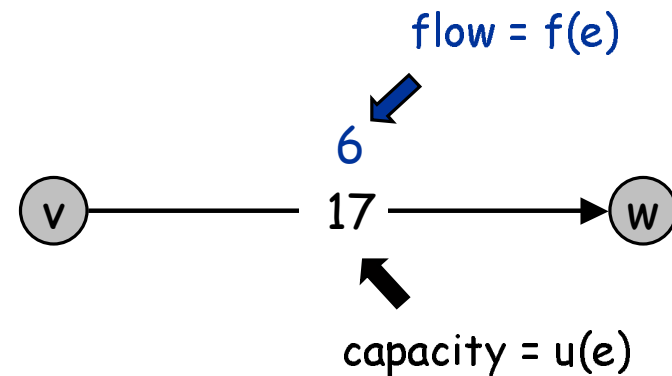
- Greedy algorithm: repeat until you get stuck.
- Fails: need to be able to "backtrack".



Residual Graph

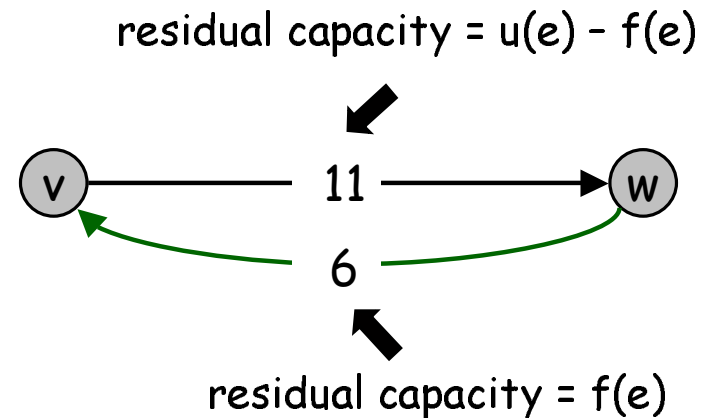
Original graph.

- Flow $f(e)$.
- Edge $e = v-w$



Residual arc.

- Edge $e = v-w$ or $w-v$.
- "Undo" flow sent.



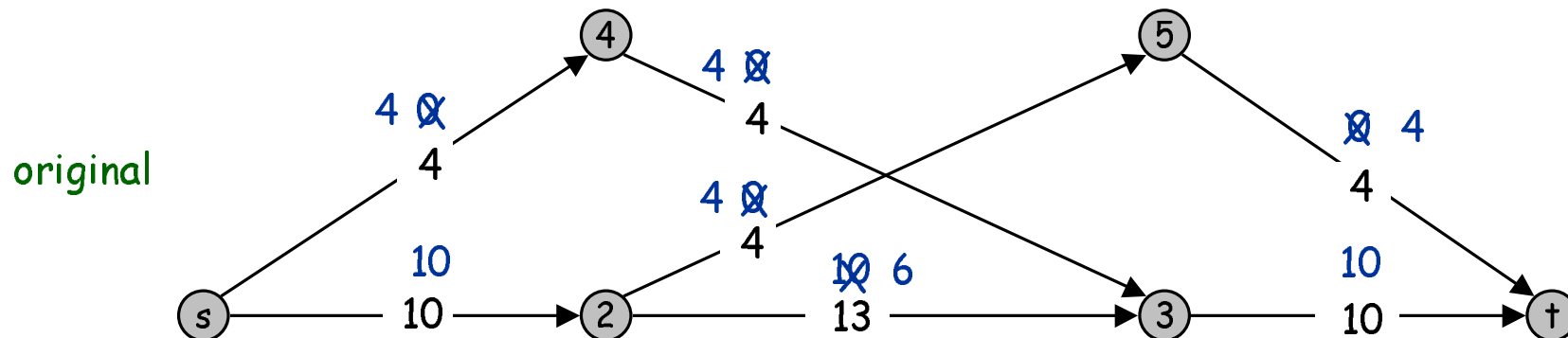
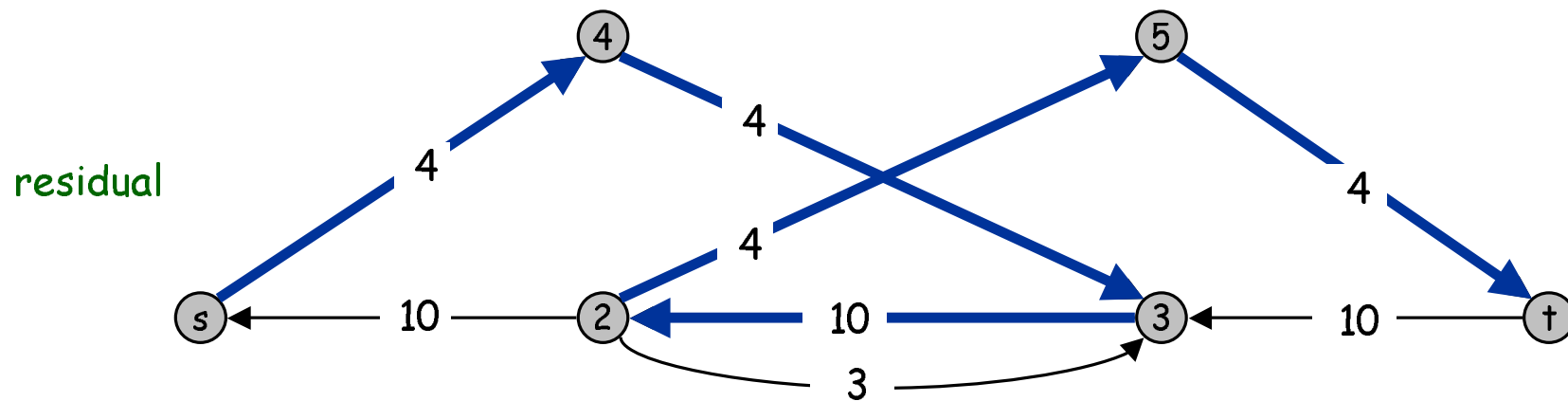
Residual graph.

- All the edges that have strictly positive residual capacity.

Augmenting Paths

Augmenting path = path in residual graph.

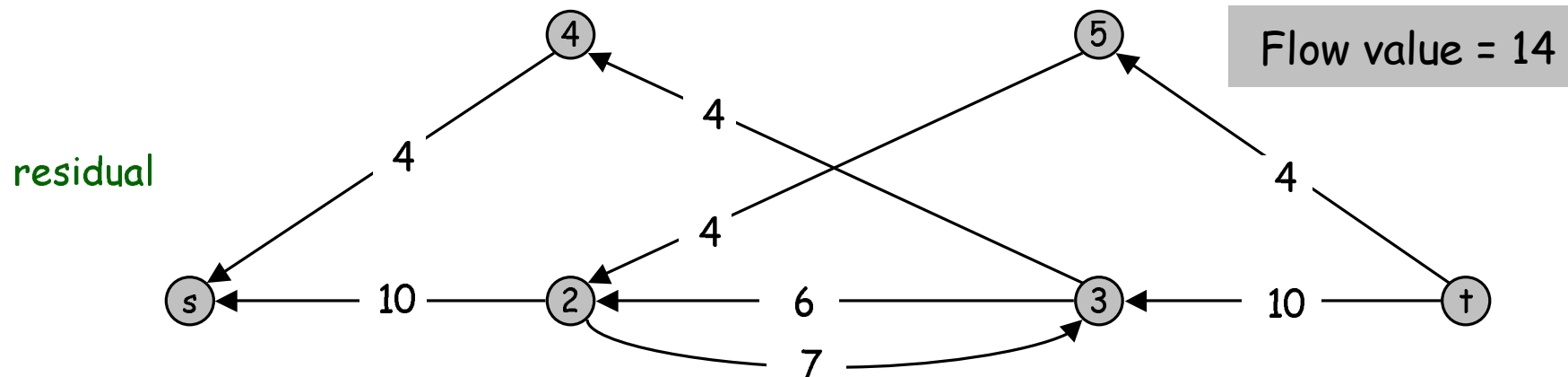
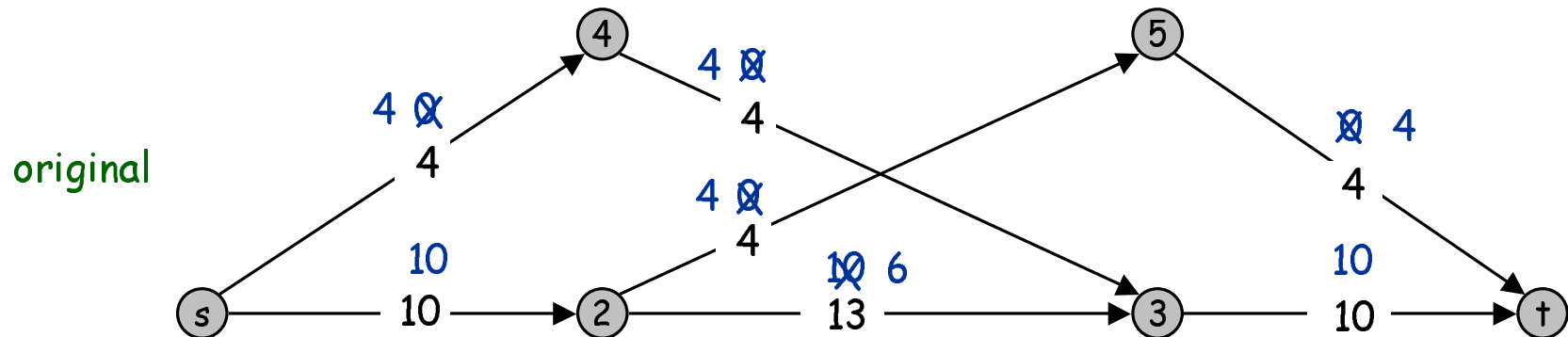
- Increase flow along forward edges.
- Decrease flow along backward edges.



Augmenting Paths

Augmenting path = path in residual graph.

- If augmenting path, then not yet a max flow.
- If no augmenting path, is it a max flow???



Ford-Fulkerson Augmenting Path Algorithm

Ford-Fulkerson algorithm.

- Generic method for solving max flow problem.

Ford-Fulkerson Augmenting Path Algorithm

Start with $f(e) = 0$ everywhere.

REPEAT (until no augmenting paths are left)
 Increase the flow along any augmenting path.

Questions.

- Does this lead to a maximum flow?
- How do we find an augmenting path? *s-t path in residual graph*
- How many augmenting paths does it take?

Max-Flow Min-Cut Theorem

Augmenting path theorem. A flow f is a max flow if and only if there are no augmenting paths.

Max-flow min-cut theorem. The value of the max flow is equal to the capacity of the min cut.

We prove both simultaneously by showing the following are equivalent:

- (i) f is a max flow.
- (ii) There is no augmenting path relative to f .
- (iii) There exists a cut whose capacity equals the value of f .

(i) \Rightarrow (ii) equivalent to not (ii) \Rightarrow not (i)

- Let f be a flow. If there exists an augmenting path, then we can improve f by sending flow along path.

(ii) \Rightarrow (iii) Next slide.

(iii) \Rightarrow (i) This was Observation 3.

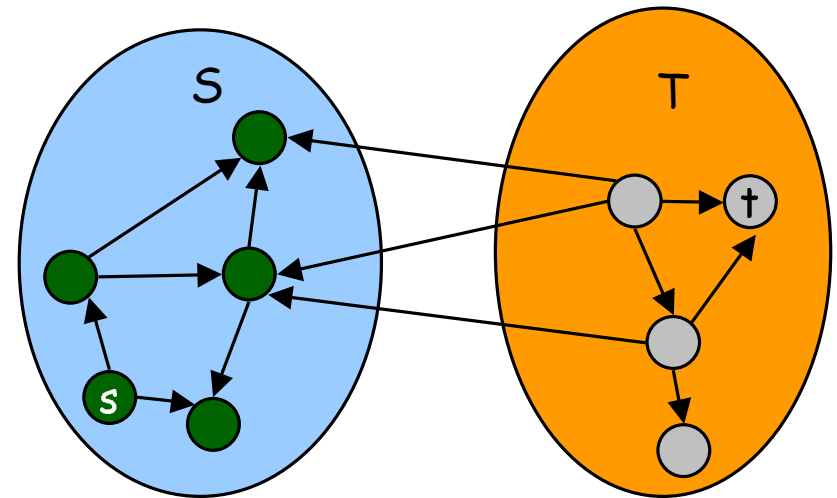
Proof of Max-Flow Min-Cut Theorem

(ii) \Rightarrow (iii). If there is no augmenting path relative to f , then there exists a cut whose capacity equals the value of f .

Proof.

- Let f be a flow with no augmenting paths.
- Let S be set of vertices reachable from s in residual graph.
 - S contains s ; since no augmenting paths, S does not contain t
 - all edges e leaving S in original network have $f(e) = u(e)$
 - all edges e entering S in original network have $f(e) = 0$

$$\begin{aligned} |f| &= \sum_{e \text{ out of } S} f(e) - \sum_{e \text{ into } S} f(e) \\ &= \sum_{e \text{ out of } S} u(e) \\ &= \text{capacity}(S, T) \end{aligned}$$

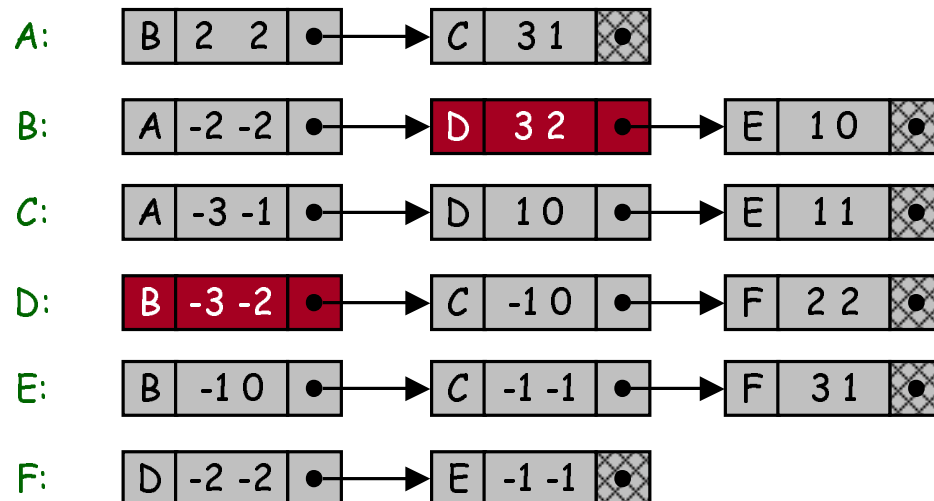
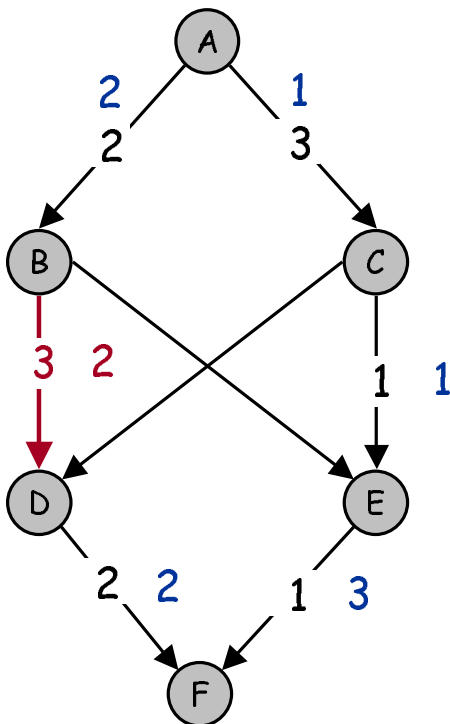


Residual Network

Ford-Fulkerson Algorithm: Implementation

Two representations of each edge in residual graph.

- May need to traverse edge in forward or reverse direction.
- Let e be edge in original network with flow $f(e)$ and capacity $u(e)$.
- In residual graph, include reverse edge and maintain anti-symmetry:
 - forward edge: flow = $f(e)$, residual capacity = $u(e) - f(e)$
 - reverse edge: flow = $-f(e)$, residual capacity = $-u(e)$



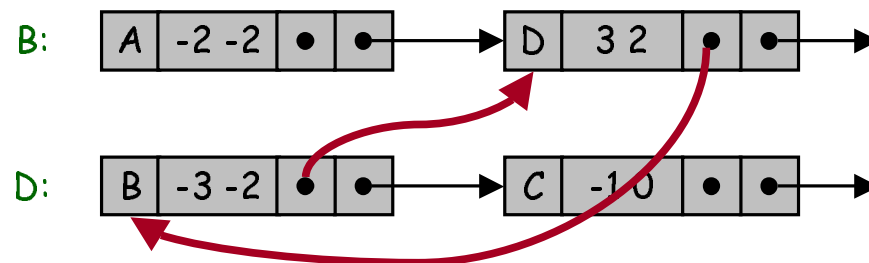
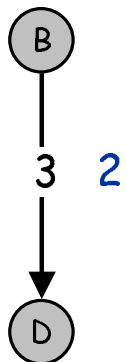
Ford-Fulkerson Algorithm: Implementation

Two representations of each edge.

- Need to update BOTH representations.
- Maintain link connecting forward and backward representations.






Edge in Adjacency List

```
typedef struct node *link;
struct node {
    int w;          // target vertex w in v-w
    int cap;        // capacity u(e)
    int flow;       // flow f(e)
    link dup;       // reversal of edge
    link next;      // next edge in adjacency list
};
```



Ford-Fulkerson Algorithm: Implementation

Compute Max Flow

```
int residualCapacity(link e) {  
    if (e->cap < 0) return -e->flow;  reverse edge  
    else return e->cap - e->flow;  forward edge  
}  
  
void GRAPHmaxflow(Graph G, int s, int t) {  
    int v, bottle;  
    link e;  
     find augmenting path  
    while (augpath(G, s, t) == TRUE) {  bottleneck capacity  
        bottle = INFINITY;  
        for (v = t; v != s; v = e->dup->v) {  
            e = G->path[v];  
            bottle = min(bottle, residualCapacity(e));  
        }  
  
        for (v = t; v != s; v = e->dup->v) {  
            e = G->path[v];  
            e->flow += bottle;  
            e->dup->flow -= bottle;  update flow on forward  
                                and reverse edges  
        }  
    }  
}
```

Ford-Fulkerson Algorithm: Analysis

Assumption: all capacities are integers between 1 and U .

Invariant: every flow value and every residual capacities remain an integer throughout the algorithm.

Theorem: the algorithm terminates in at most $|f^*| \leq VU$ iterations.

Corollary: if $U = 1$, then algorithm runs in $O(V)$ iterations.

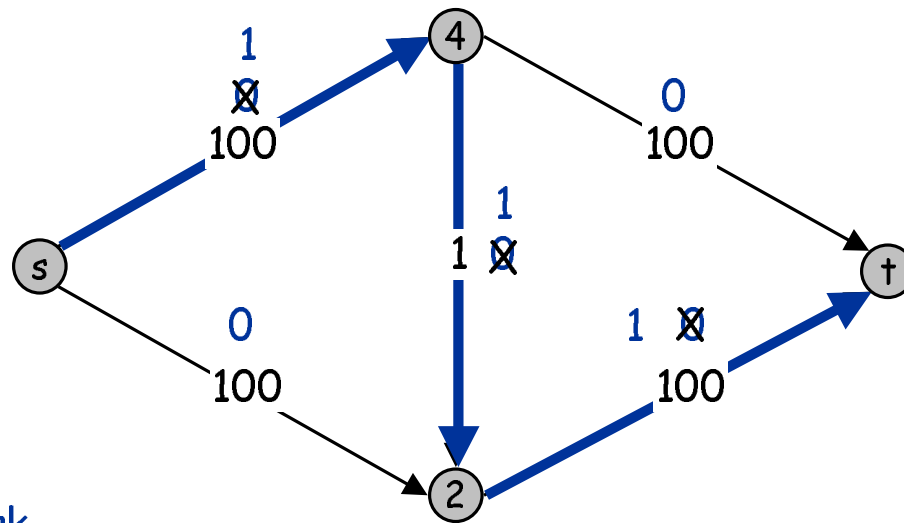
not polynomial
in input size!



Integrality theorem: if all arc capacities are integers, then there exists a max flow f for which every flow value is an integer.

Choosing Good Augmenting Paths

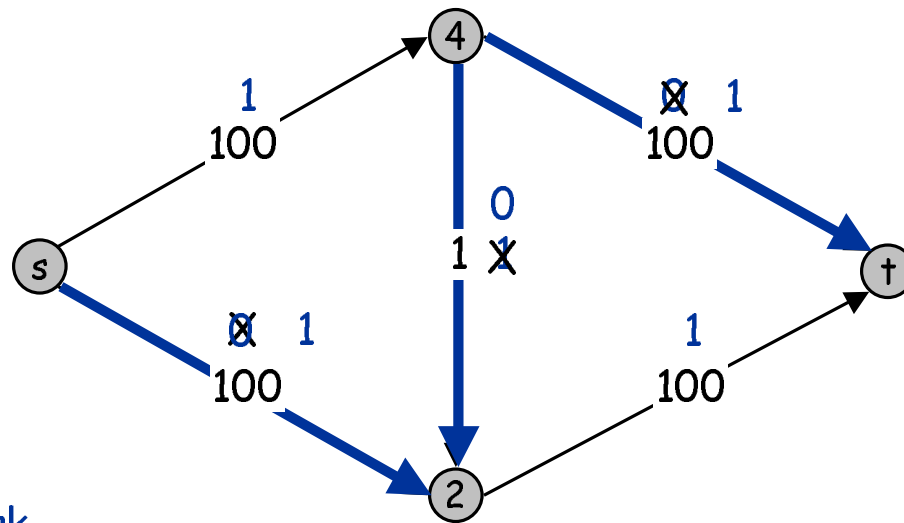
Use care when selecting augmenting paths.



Original Network

Choosing Good Augmenting Paths

Use care when selecting augmenting paths.



Original Network

Choosing Good Augmenting Paths

Use care when selecting augmenting paths.

- Some choices lead to exponential algorithms.
- Clever choices lead to polynomial algorithms.
- Optimal choices for real world problems ???

Design goal is to choose augmenting paths so that:

- Can find augmenting paths efficiently.
- Few iterations.

Choose augmenting path with:

- Fewest number of arcs.
- Max bottleneck capacity.

Edmonds-Karp (1972)

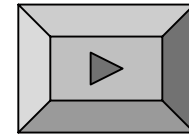
(shortest path)

(fattest path)

Shortest Augmenting Path

Shortest augmenting path.

- Easy to implement with BFS.
- Finds augmenting path with fewest number of arcs.



```
while (!QUEUEisempty()) {  
    v = QUEUEget();  
    for (e = G->adj[v]; e != NULL; e = e->next) {  
        if (residualCapacity(e) > 0) {  
            w = e->w;  
            if (G->dist[w] > G->dist[v] + 1) {  
                G->dist[w] = G->dist[v] + 1;  
                G->path[w] = e;  
                QUEUEput(w);  
            }  
        }  
    }  
}  
  
return (dist[t] < INFINITY);
```

← ignore unless it's a residual arc

← keep track of path

← return 1 if there's an augmenting path

Shortest Augmenting Path Analysis

Length of shortest augmenting path increases monotonically.

- Strictly increases after at most E augmentations.
- At most $E \cdot V$ total augmenting paths.
- $O(E^2 \cdot V)$ running time.



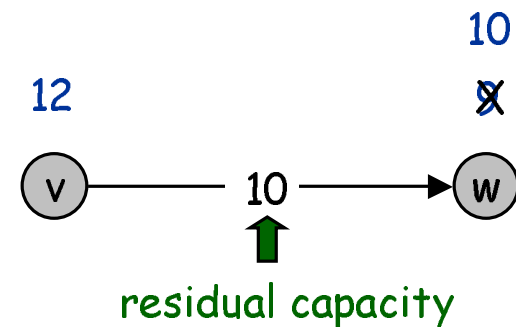
Fattest Augmenting Path

Fattest augmenting path.

- Finds augmenting path whose bottleneck capacity is maximum.
- Delivers most amount of flow to sink.
- Solve using Dijkstra-style (PFS) algorithm.

```
if (wt[w] < min(wt[v], cap[e])  
    wt[w] = min(wt[v], cap[e])
```

Relax an edge $e = v-w$



Analysis.

- $O(E \log V)$ per augmentation with binary heap.
- $O(E + V \log V)$ per augmentation with Fibonacci heap.
- Fact: $O(E \log U)$ augmentations if capacities are between 1 and U .

Choosing an Augmenting Path

Choosing an augmenting path.

- Any path will do \Rightarrow wide latitude in implementing Ford-Fulkerson.
- Generic priority first search.
- Some choices lead to good worst-case performance.
 - shortest augmenting path
 - fattest augmenting path
 - variation on a theme: PFS
- Average case not well understood.

Research challenges.

- Practice: solve max flow problems on real networks in linear time.
- Theory: prove it for worst-case networks.

History of Worst-Case Running Times

Year	Discoverer	Method	Big-Oh
1951	Dantzig	Simplex	$E V^2 U^\dagger$
1955	Ford, Fulkerson	Augmenting path	$E V U^\dagger$
1970	Edmonds-Karp	Shortest path	$E^2 V$
1970	Edmonds-Karp	Max capacity	$E \log U (E + V \log V)^\dagger$
1970	Dinitz	Improved shortest path	$E V^2$
1972	Edmonds-Karp, Dinitz	Capacity scaling	$E^2 \log U^\dagger$
1973	Dinitz-Gabow	Improved capacity scaling	$E V \log U^\dagger$
1974	Karzanov	Preflow-push	V^3
1983	Sleator-Tarjan	Dynamic trees	$E V \log V$
1986	Goldberg-Tarjan	FIFO preflow-push	$E V \log (V^2 / E)$
...
1997	Goldberg-Rao	Length function	$E^{3/2} \log (V^2 / E) \log U^\dagger$ $E V^{2/3} \log (V^2 / E) \log U^\dagger$

† Arc capacities are between 1 and U.

An Application

Jon placement.

- Companies make job offers.
- Students have job choices.

Can we fill every job?

Can we employ every student?

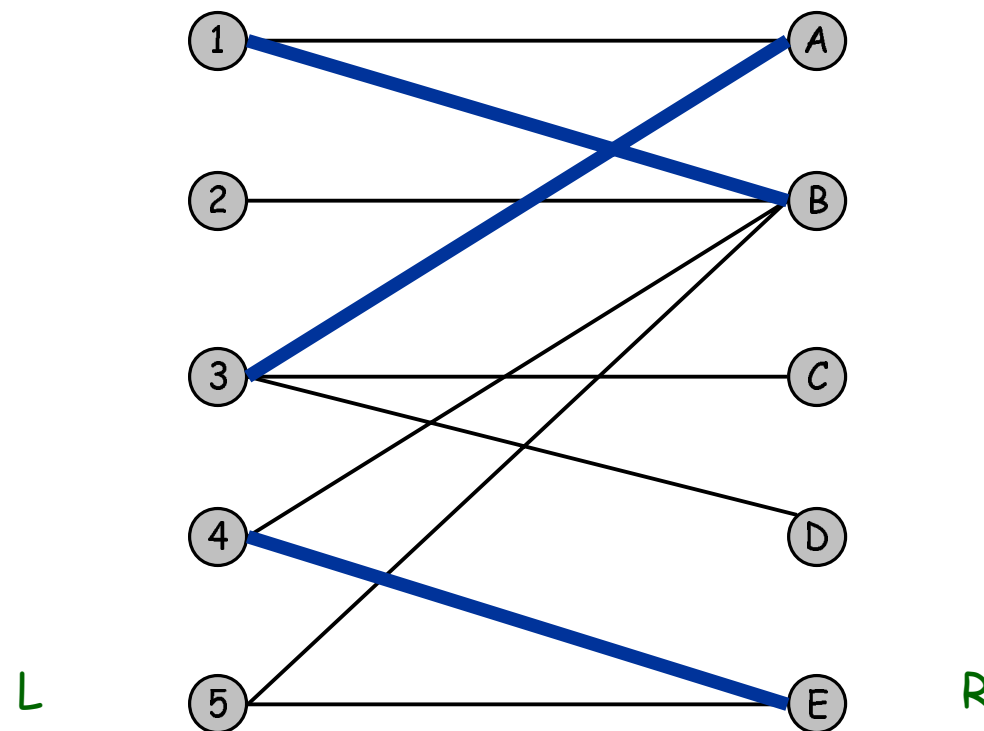
Alice-Adobe
Bob-Yahoo
Carol-HP
Dave-Apple
Eliza-IBM
Frank-Sun

Alice	Adobe
Adobe	Alice
Apple	Bob
HP	Dave
Bob	Apple
Adobe	Alice
Apple	Bob
Yahoo	Dave
Carol	HP
HP	Alice
IBM	Carol
Sun	Frank
Dave	IBM
Adobe	Carol
Apple	Eliza
Eliza	Sun
IBM	Carol
Sun	Eliza
Yahoo	Frank
Frank	Yahoo
HP	Bob
Sun	Eliza
Yahoo	Frank

Bipartite Matching

Bipartite matching.

- Input: undirected, **bipartite** graph G .
- A set of edges M is a matching if each vertex appears at most once.
- Max matching: find a max cardinality matching.



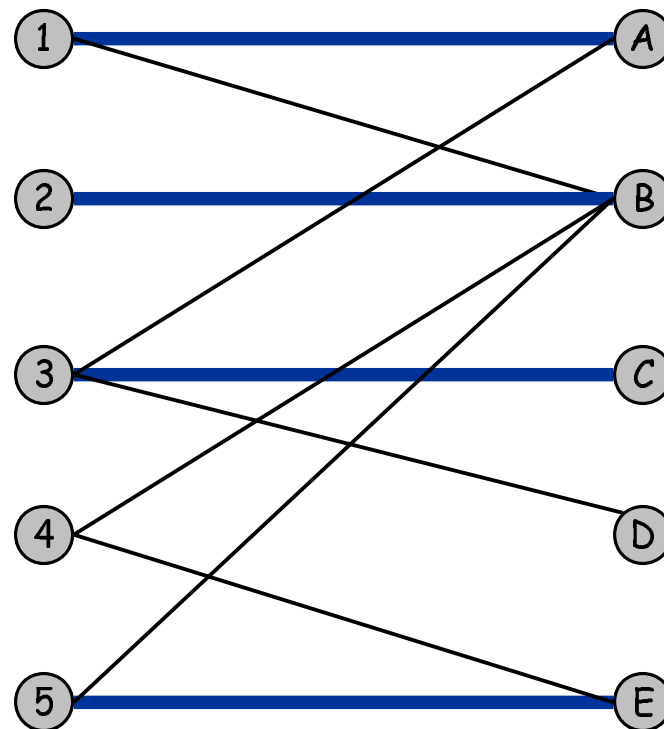
Matching

1-B, 3-A, 4-E

Bipartite Matching

Bipartite matching.

- Input: **undirected, bipartite** graph G .
- A set of edges M is a matching if each vertex appears at most once.
- Max matching: find a max cardinality matching.



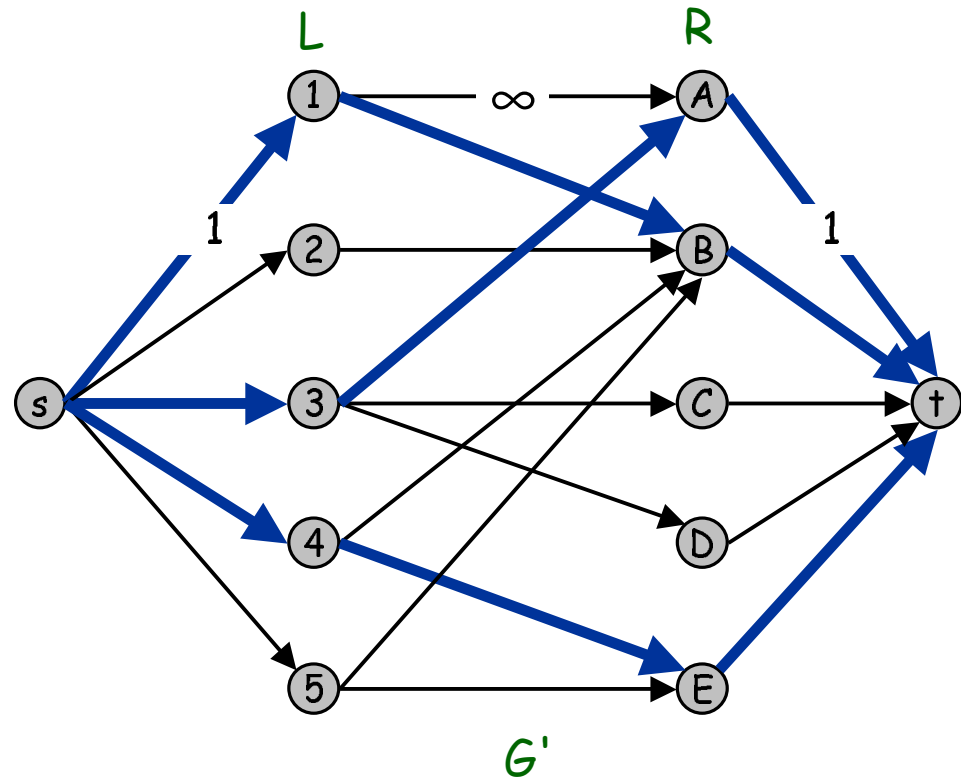
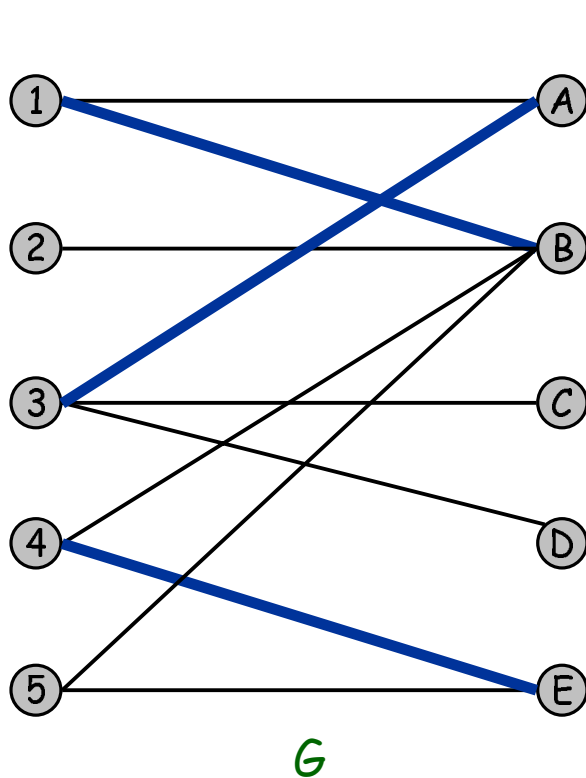
Matching

1-A, 2-B, 3-C, 4-D

Bipartite Matching

Reduces to max flow.

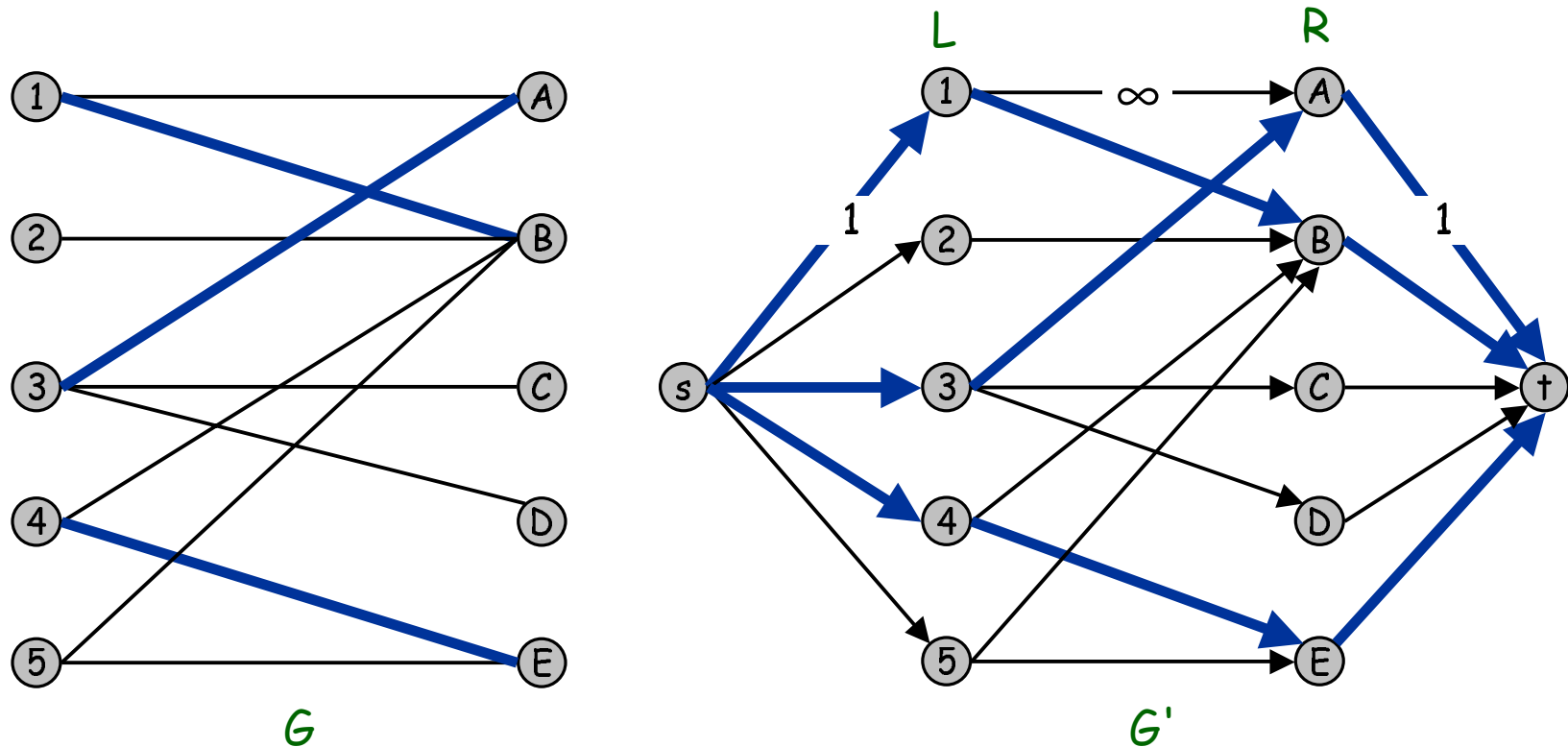
- Create **directed graph G'** .
- Direct all arcs from L to R, and give infinite (or unit) capacity.
- Add source s , and unit capacity arcs from s to each node in L.
- Add sink t , and unit capacity arcs from each node in R to t .



Bipartite Matching: Proof of Correctness

Claim. Matching in G of cardinality k induces flow in G' of value k .

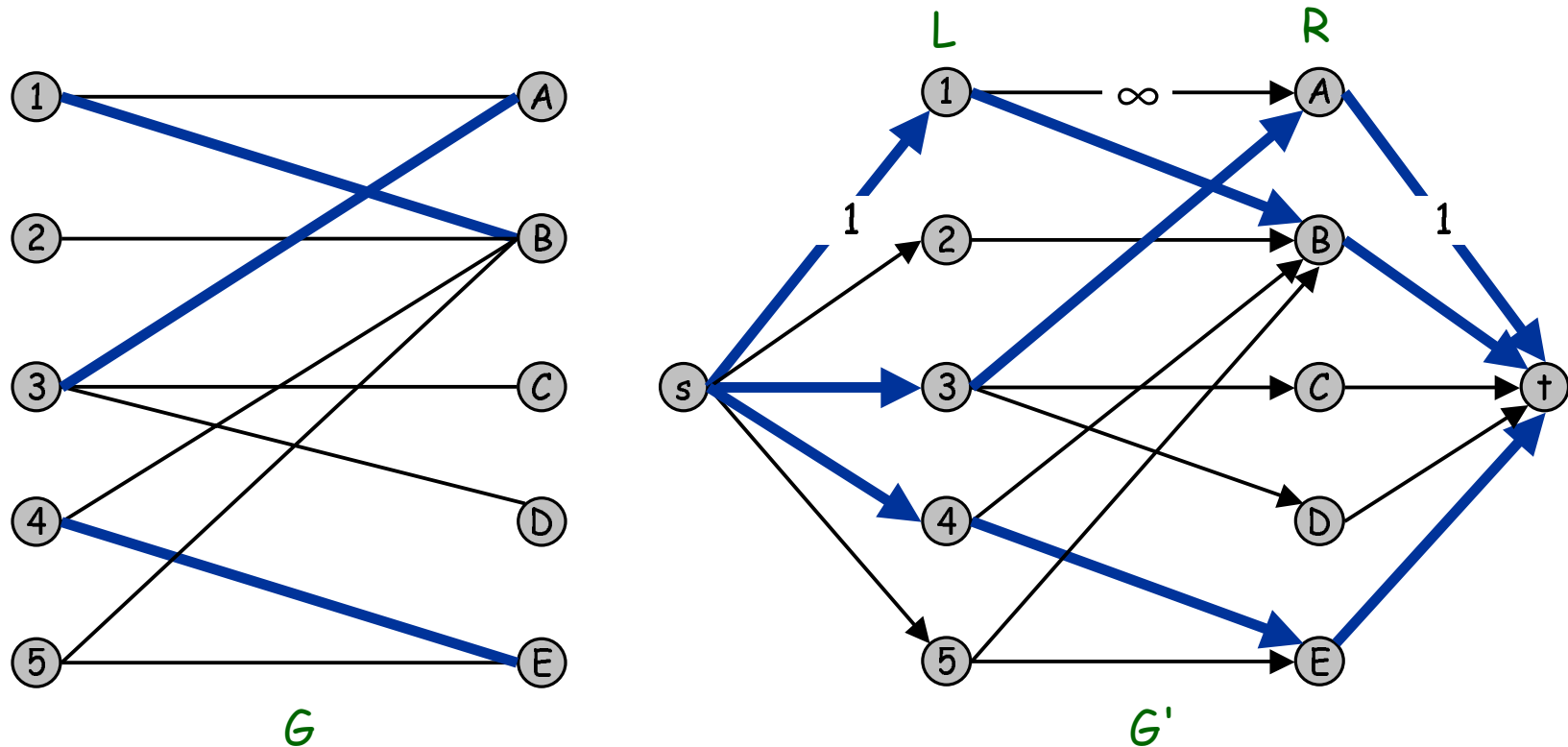
- Given matching $1-2'$ $3-1'$ $4-5'$ of cardinality 3.
- Consider flow that sends 1 unit along each of 3 paths:
 $s-1-2'-t$ $s-3-1'-t$ $s-4-5'-t$.
- f is a flow, and has cardinality 3.



Bipartite Matching: Proof of Correctness

Claim. Flow f of value k in G' induces matching of cardinality k in G .

- By integrality theorem, there exists 0/1 valued flow f of value k .
- Consider M = set of edges from L to R with $f(e) = 1$.
 - each node in L and R participates in at most one edge in M
 - $|M| = k$: consider cut $(L \cup s, R \cup t)$



Reduction

Reduction.

- Given an instance of **bipartite matching**.
- Transform it to a **max flow** problem.
- Solve max flow problem.
- Transform max flow solution to bipartite matching solution.

Issues.

- How expensive is transformation?
- Is it better to solve problem directly? $O(E V^{1/2})$ **bipartite matching**

Bottom line: max flow is an extremely rich problem-solving model.

- Many important practical problems reduce to max flow.
- We know good algorithms for solving max flow problems.