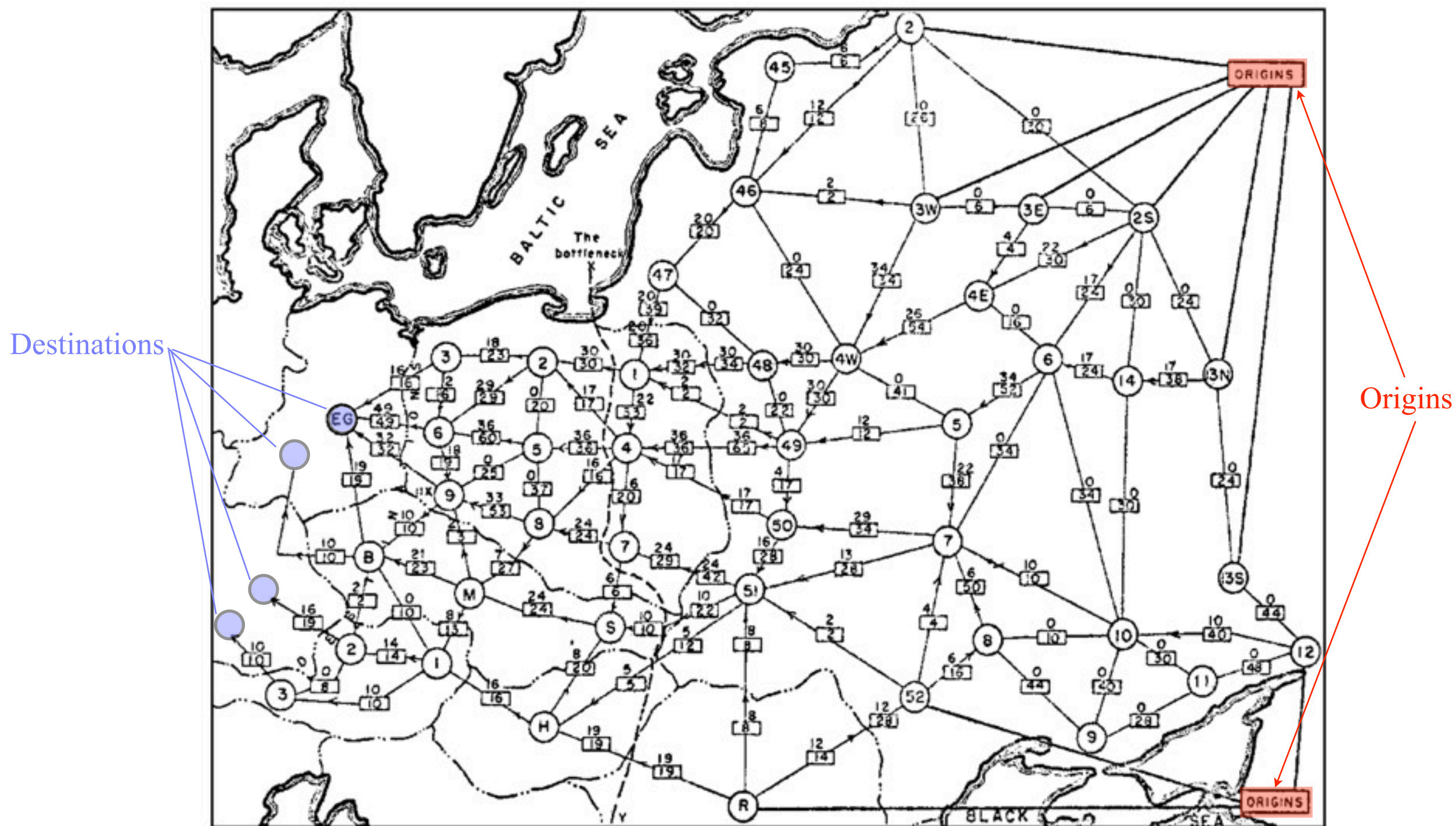
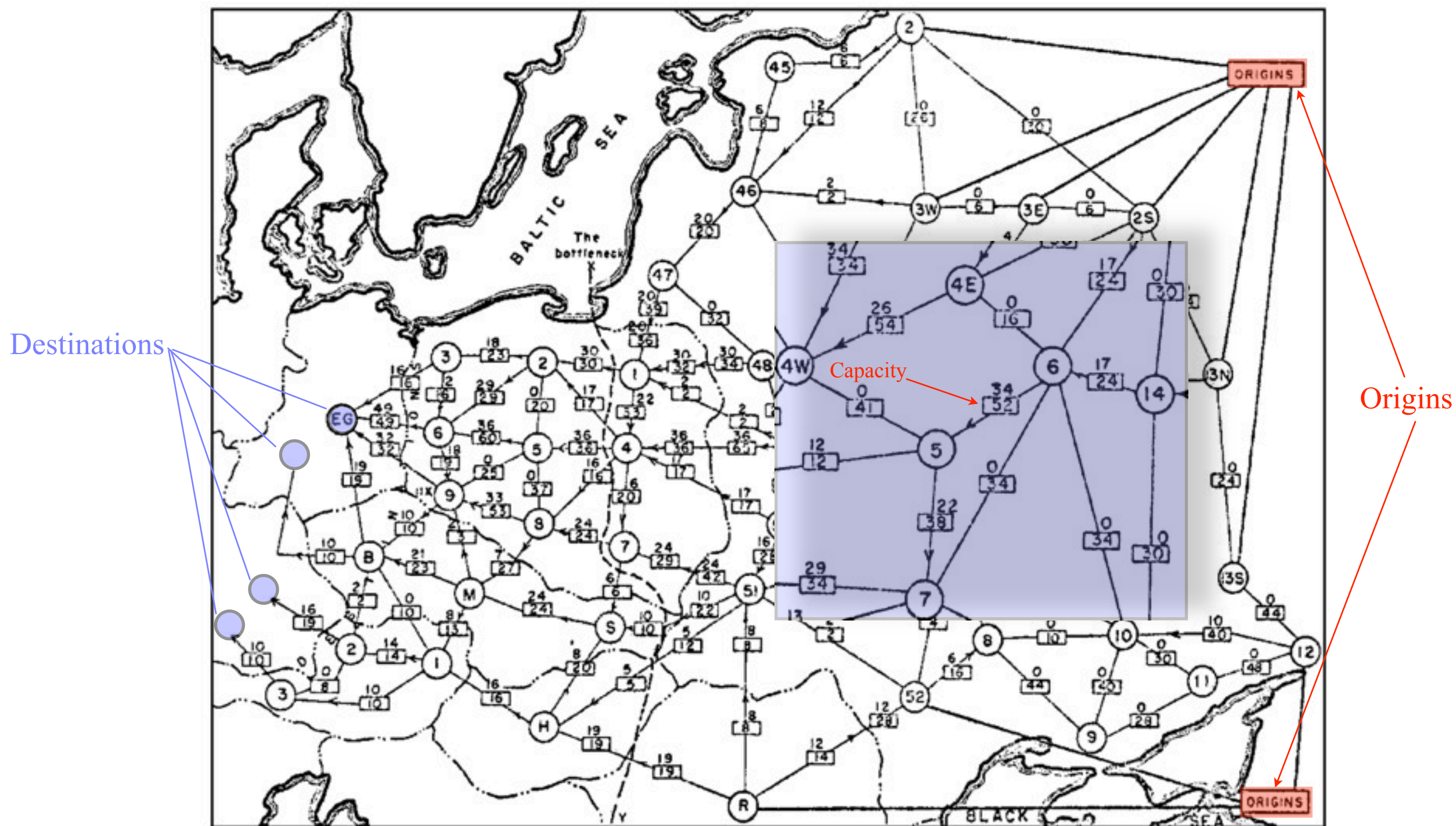


Flows on Graphs

Algorithmique
Fall semester 2011/12



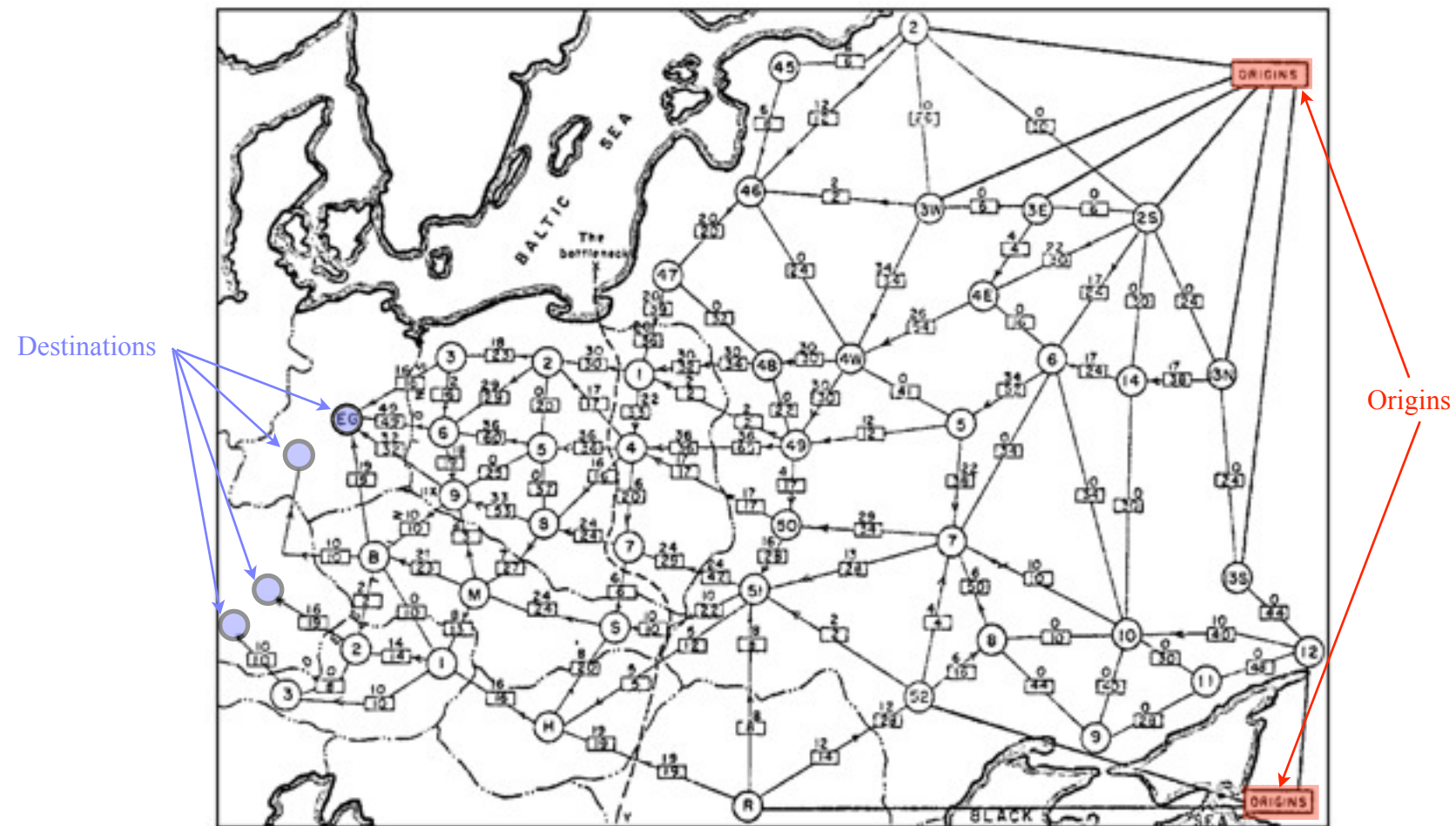
Schematic diagram of the railway network of the Western Soviet Union and East European countries, from Harris & Ross (1955), declassified by the Pentagon in 1999.



Schematic diagram of the railway network of the Western Soviet Union and East European countries, from Harris&Ross (1955), declassified by the Pentagon in 1999.

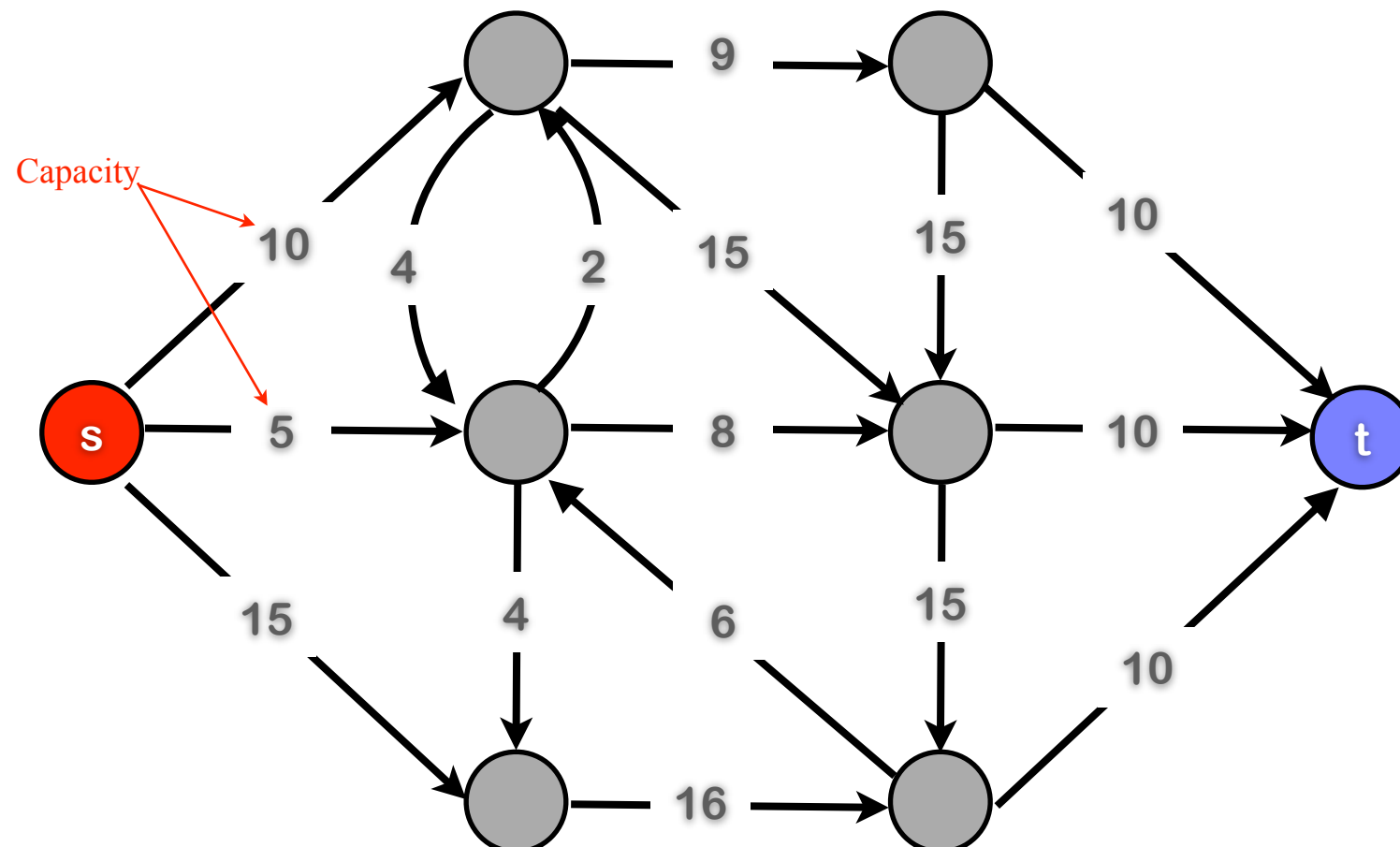
Goal of Soviet Union

Maximize throughput from the “origins” to the destinations.



Problem Formulation

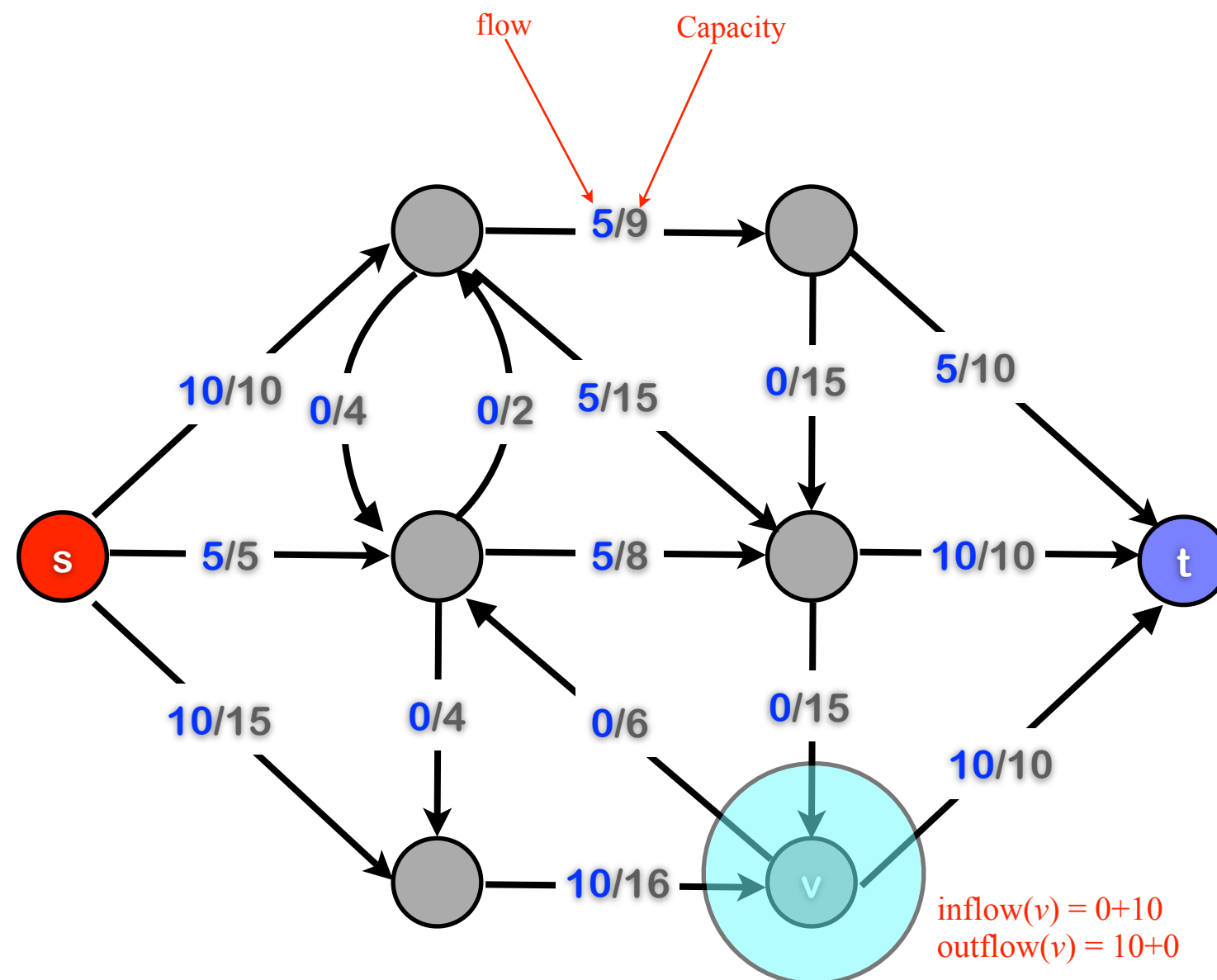
Input: weighted directed graph, source vertex s , sink vertex t
Label of edge e is called the capacity $\text{cap}(e)$ of e .



Problem Formulation

Definition: An s,t -flow f is an assignment of values to edges such that

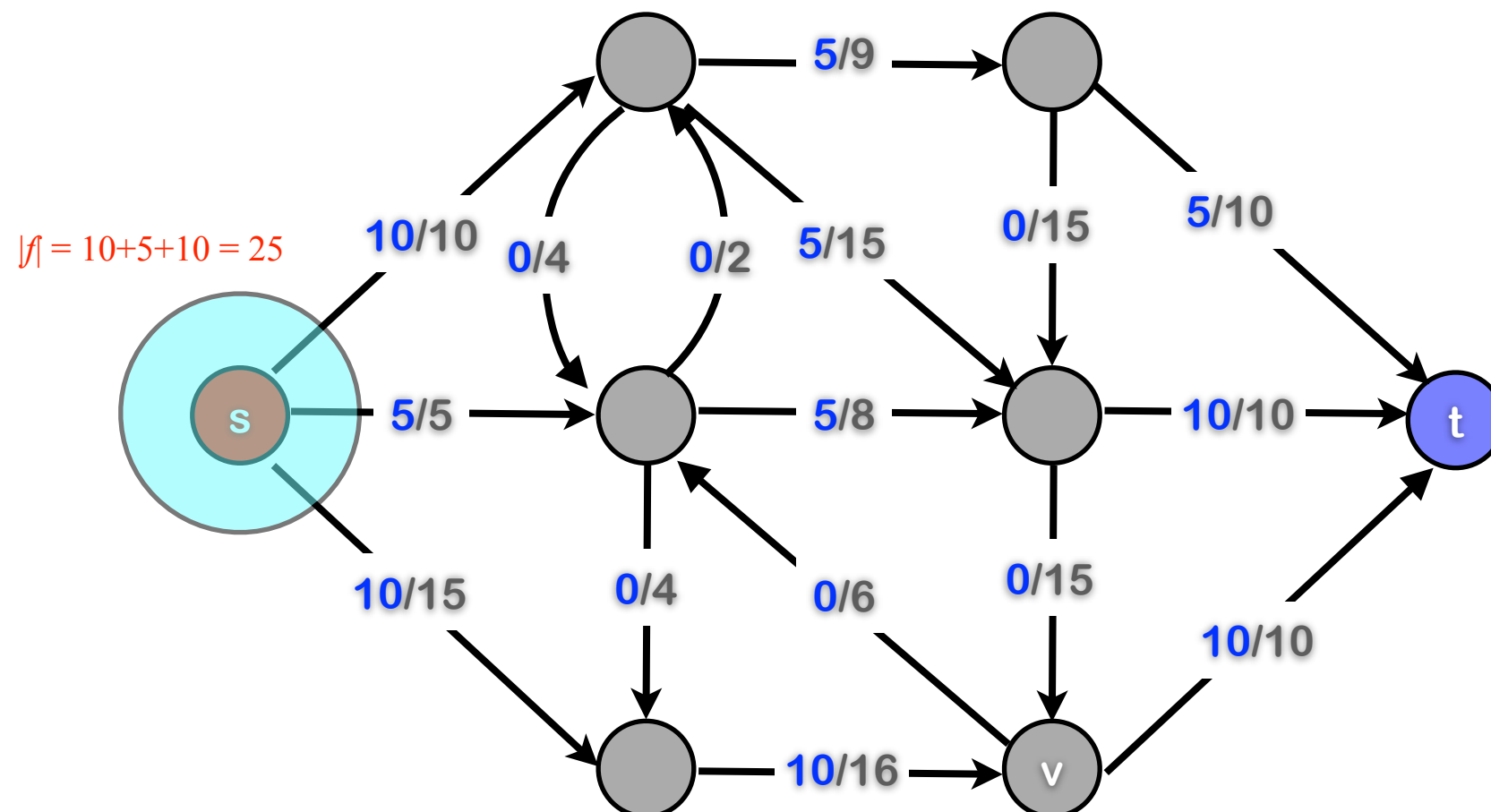
- Capacity constraint: for every edge e , $f(e) \leq \text{cap}(e)$
- Flow constraint: For every node v except s, t , the inflow into v = outflow from v



Problem Formulation

Definition: An s,t -flow f is an assignment of values to edges such that

- Capacity constraint: for every edge e , $f(e) \leq \text{cap}(e)$
- Flow constraint: For every node v except s, t , the inflow into v = outflow from v
- Value of the flow $|f| = \text{outflow}(s) - \text{inflow}(s)$.

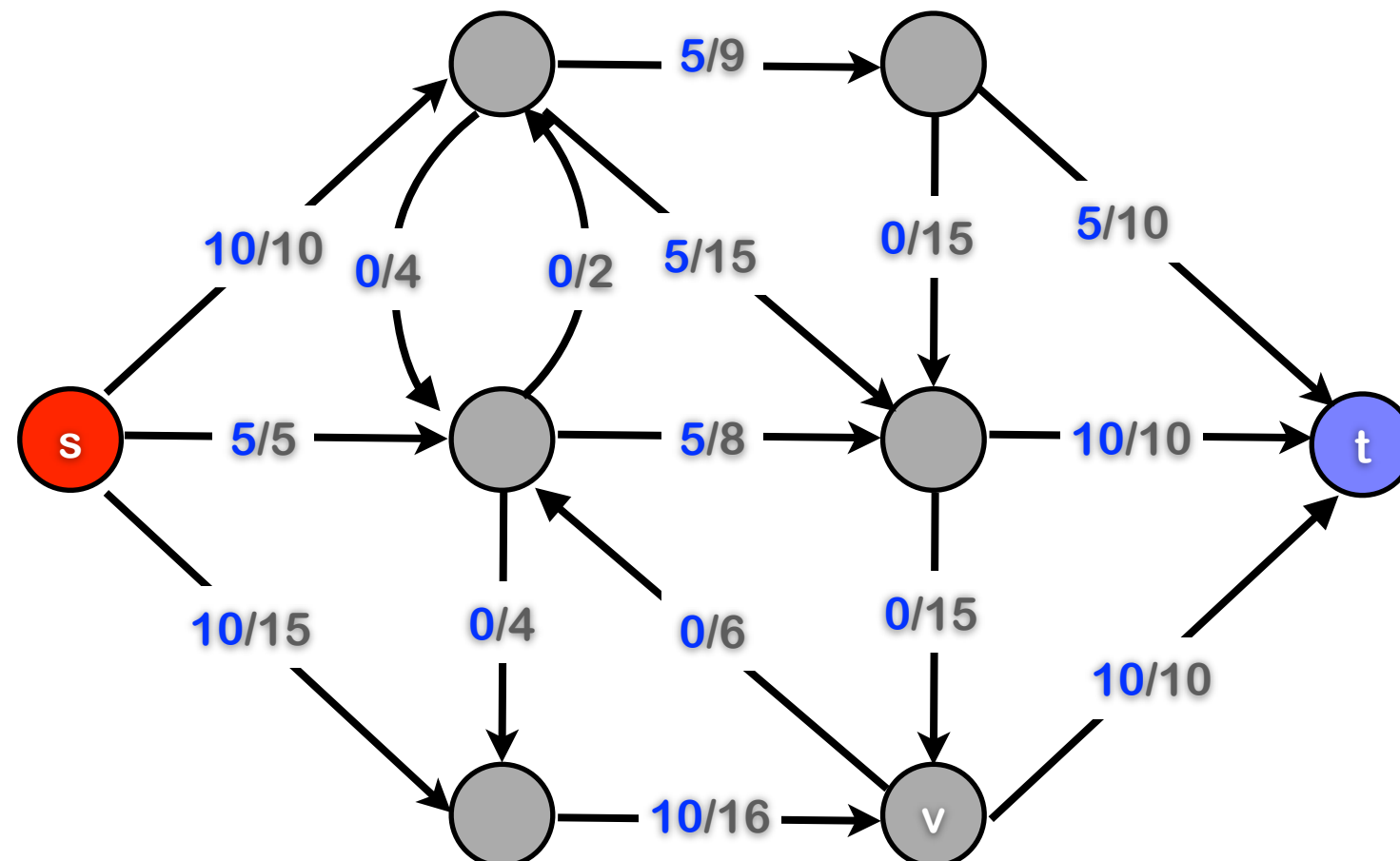


Problem Formulation

Definition: An s,t -flow f is an assignment of values to edges such that

- Capacity constraint: for every edge e , $f(e) \leq \text{cap}(e)$
- Flow constraint: For every node v except s, t , the inflow into v = outflow from v
- Value of the flow $|f| = \text{outflow}(s) - \text{inflow}(s)$.

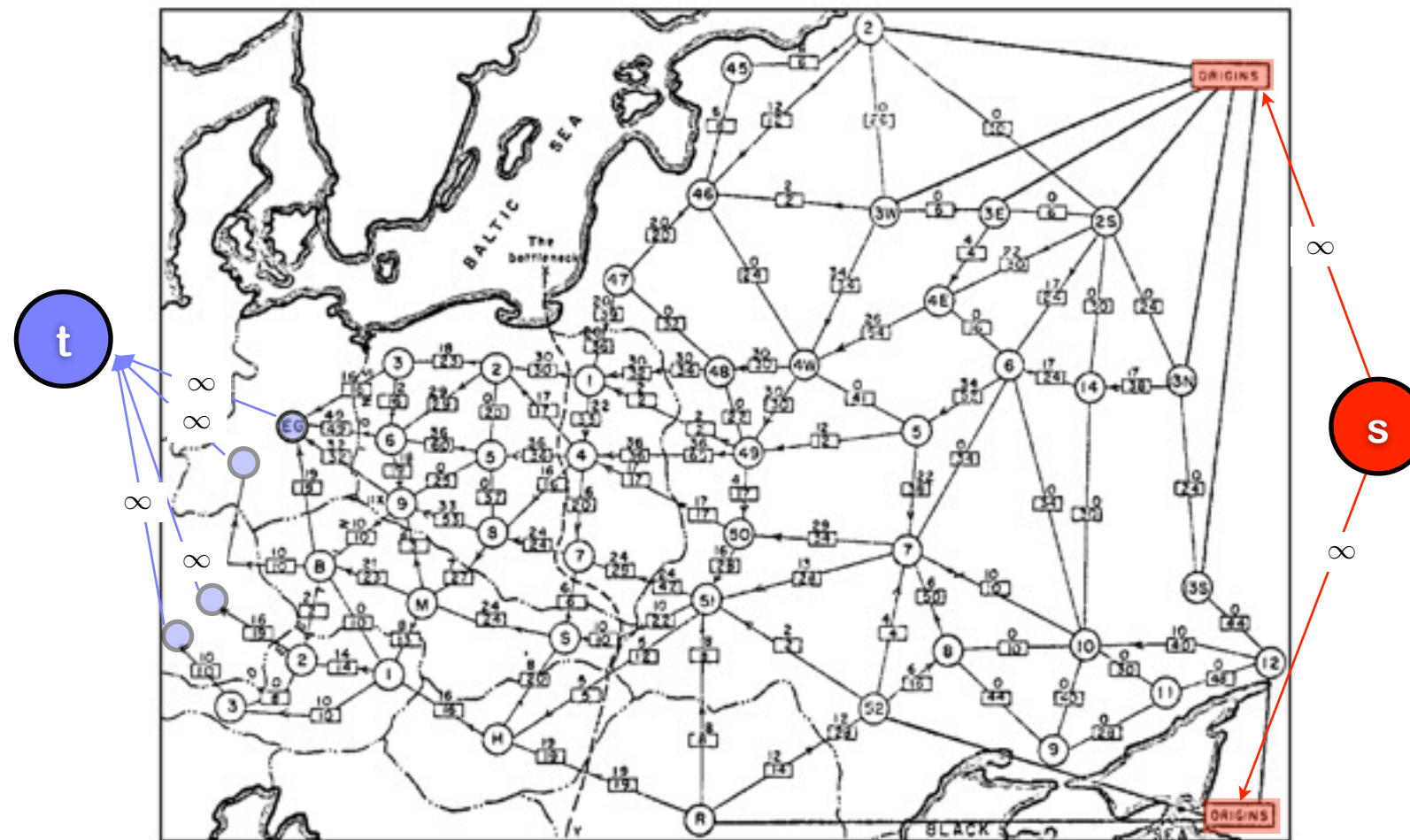
Find flow of maximum value in the network.

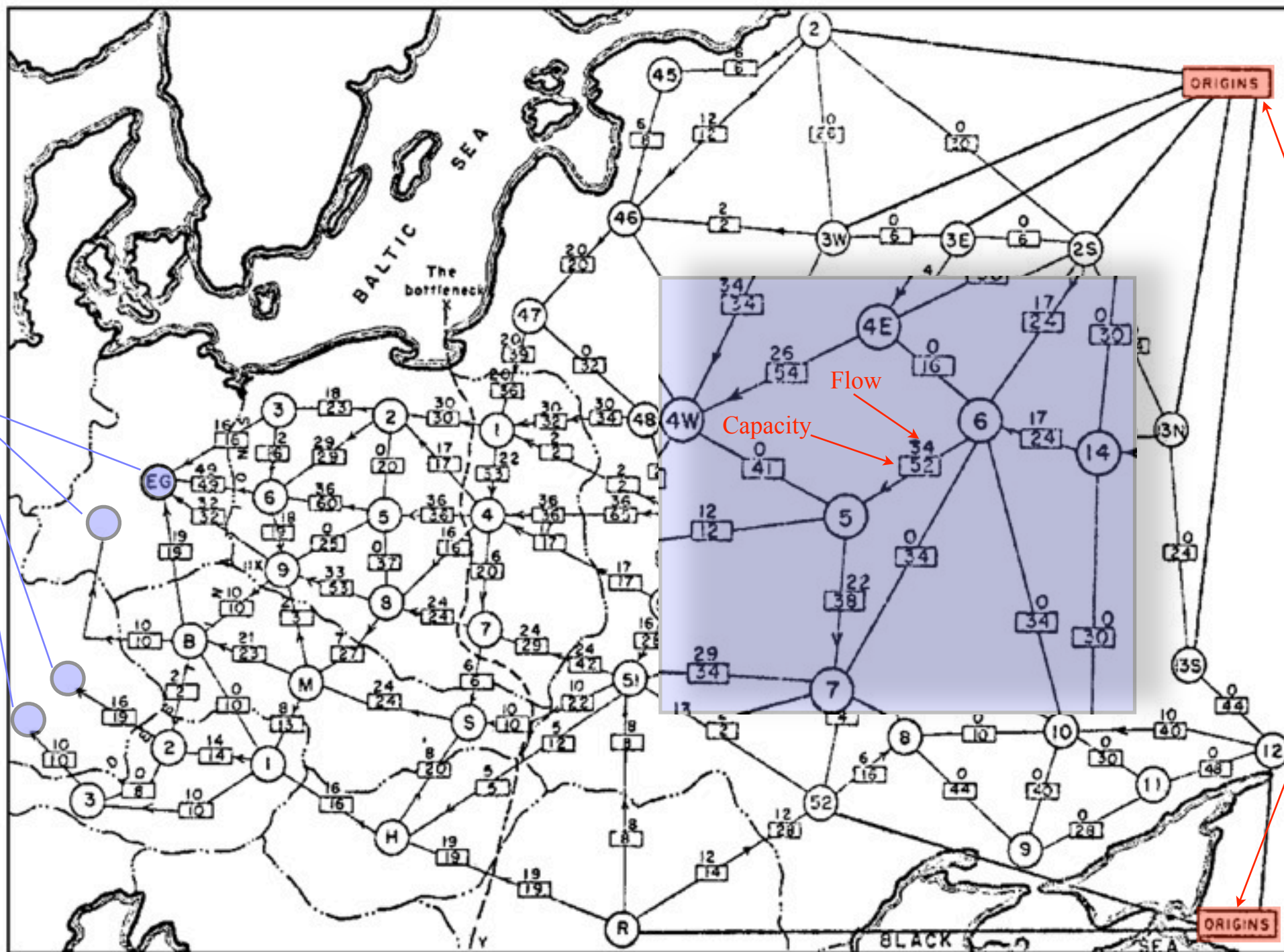


Goal of Soviet Union

Add node s with edges of capacity infinity to the origins, node t with edges from the destinations with capacity infinity.

Find maximum s, t -flow



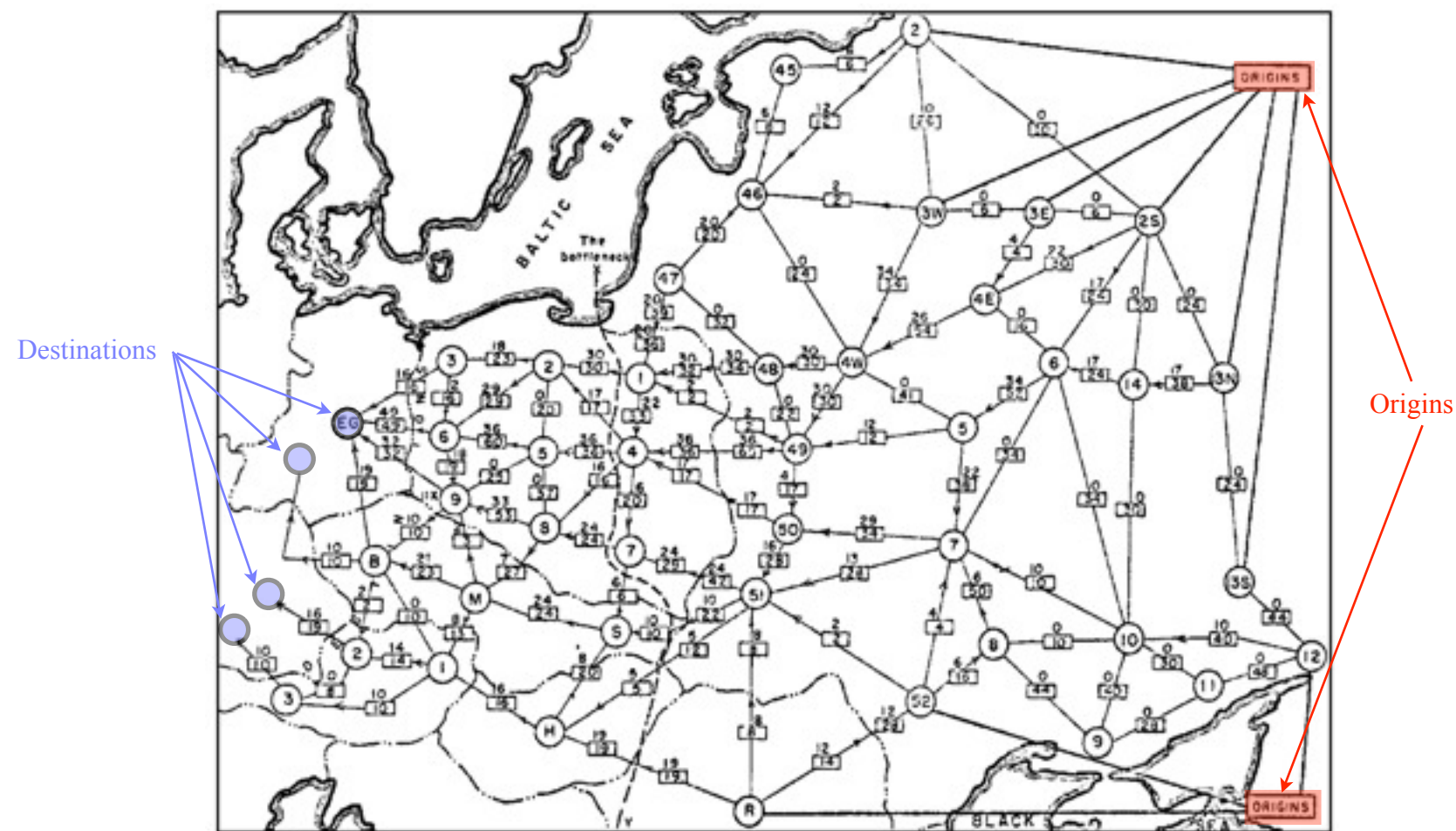


Destinations

Origins

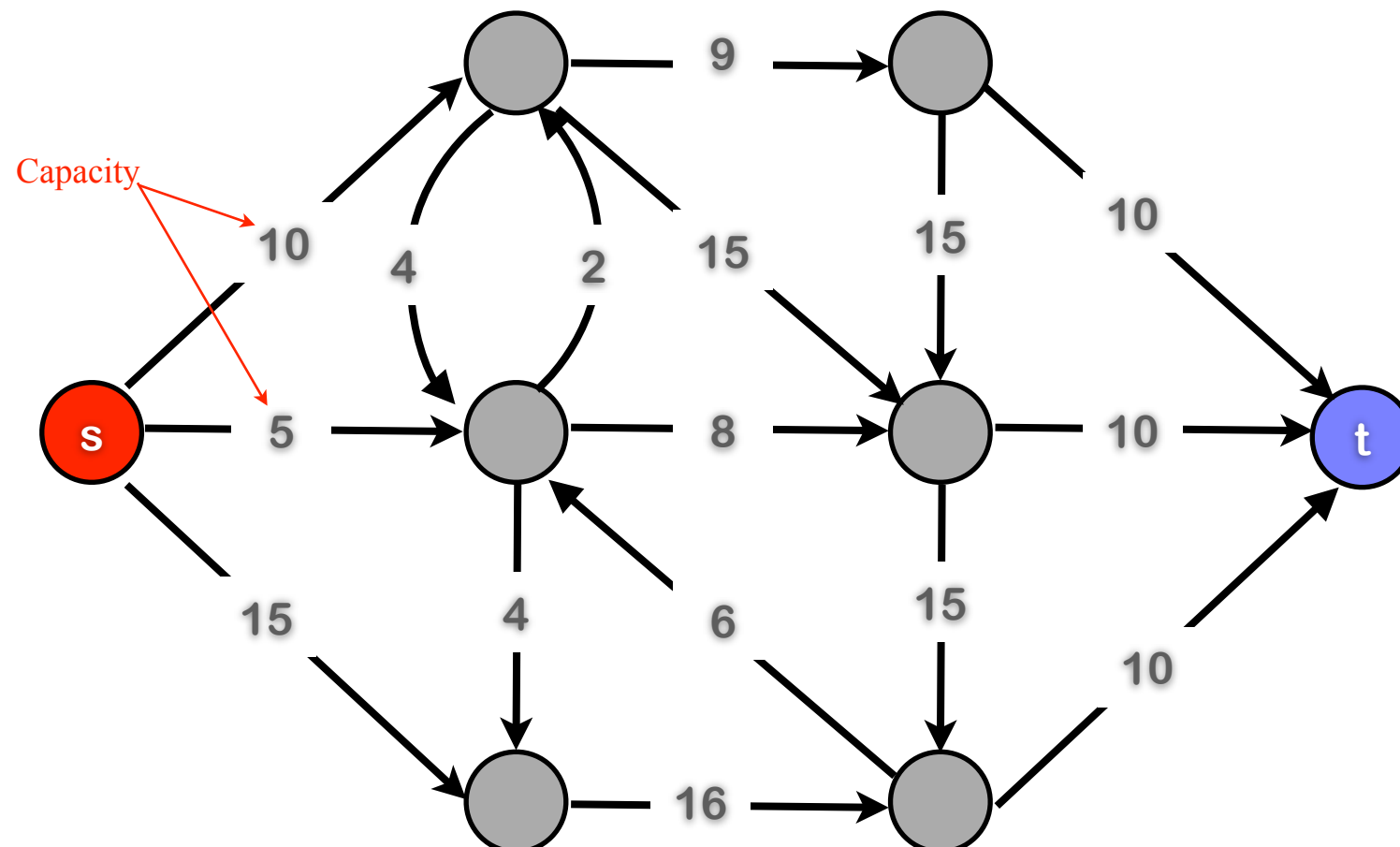
Goal of US Air Force (1950's)

Disrupt flow of goods into satellite countries with least effort.



Problem Formulation

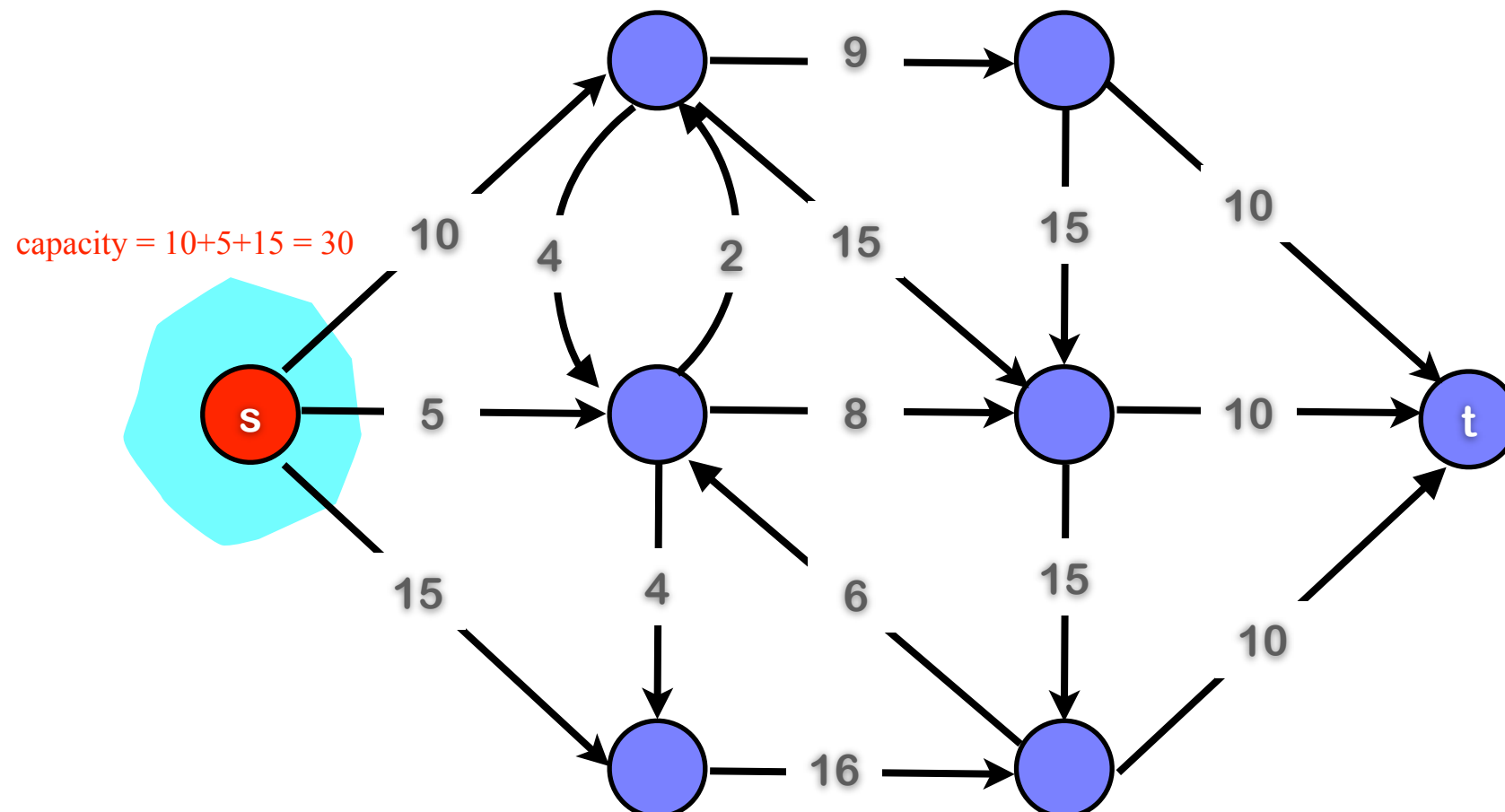
Input: weighted directed graph, source vertex s , sink vertex t
Label of edge e is called the capacity $\text{cap}(e)$ of e .



Problem Formulation

Definition:

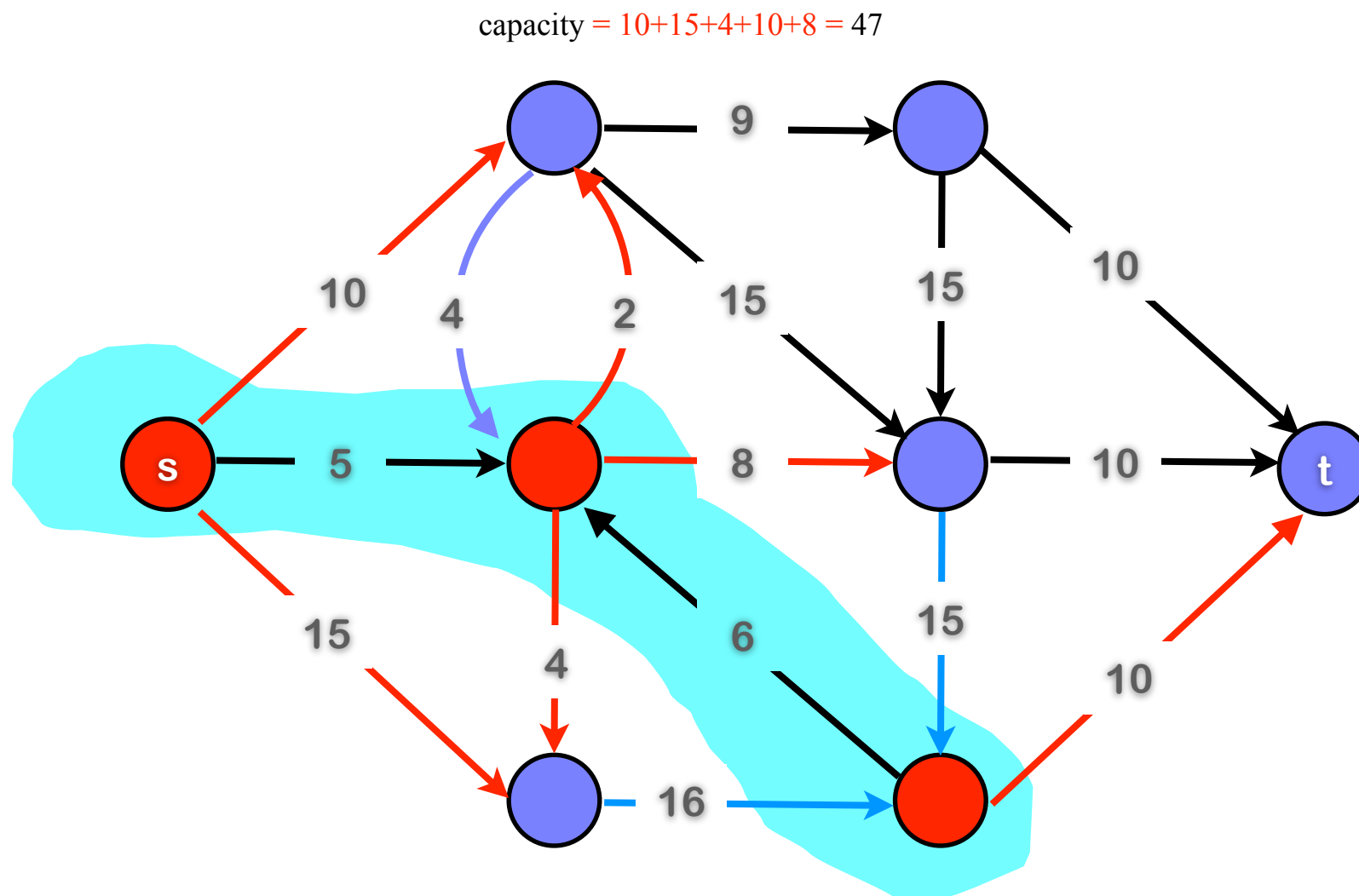
- An s,t -cut C is a partition of the vertices of the graph into two disjoint sets S and T such that s is in S and t is in T .
- The capacity $\text{cap}(C)$ of the cut is the **sum of capacities of edges from S to T**



Problem Formulation

Definition:

- An s,t -cut C is a partition of the vertices of the graph into two disjoint sets S and T such that s is in S and t is in T .
- The capacity $\text{cap}(C)$ of the cut is the **sum of capacities of edges from S to T**

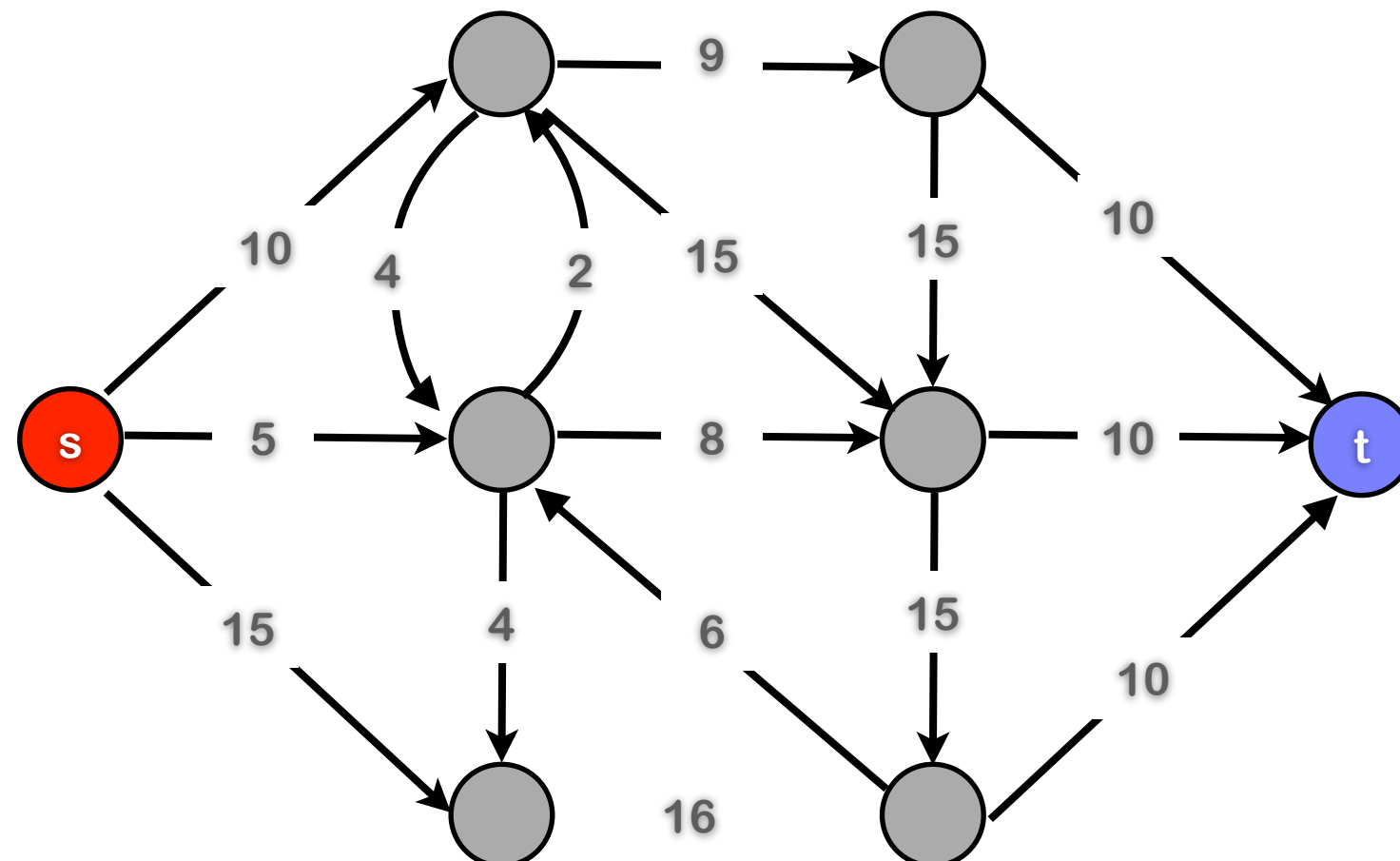


Problem Formulation

Definition:

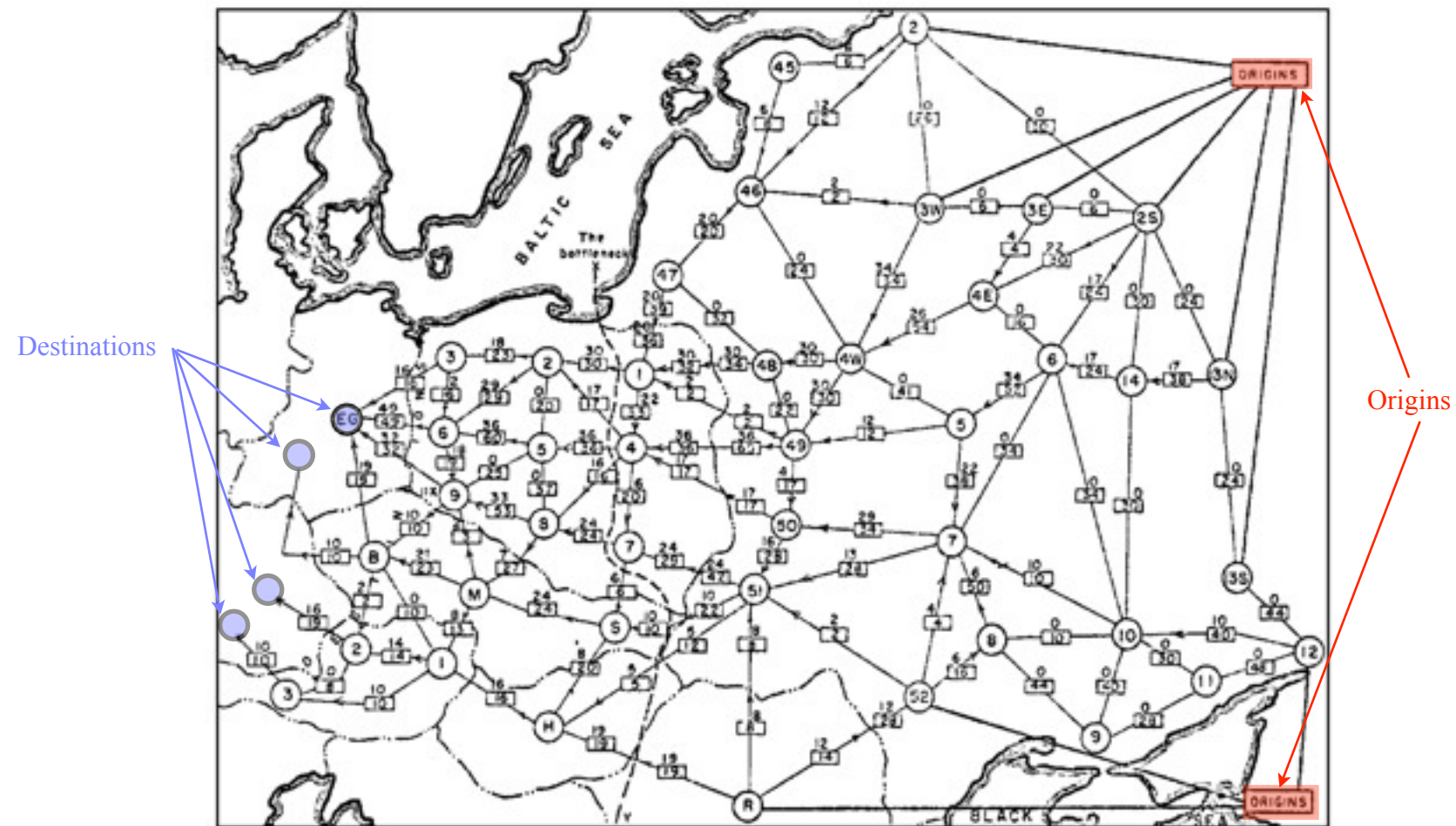
- An s,t -cut C is a partition of the vertices of the graph into two disjoint sets S and T such that s is in S and t is in T .
- The capacity $\text{cap}(C)$ of the cut is the **sum of capacities of edges from S to T**

Find cut of minimal capacity (minimum cut)



Goal of US Air Force (1950's)

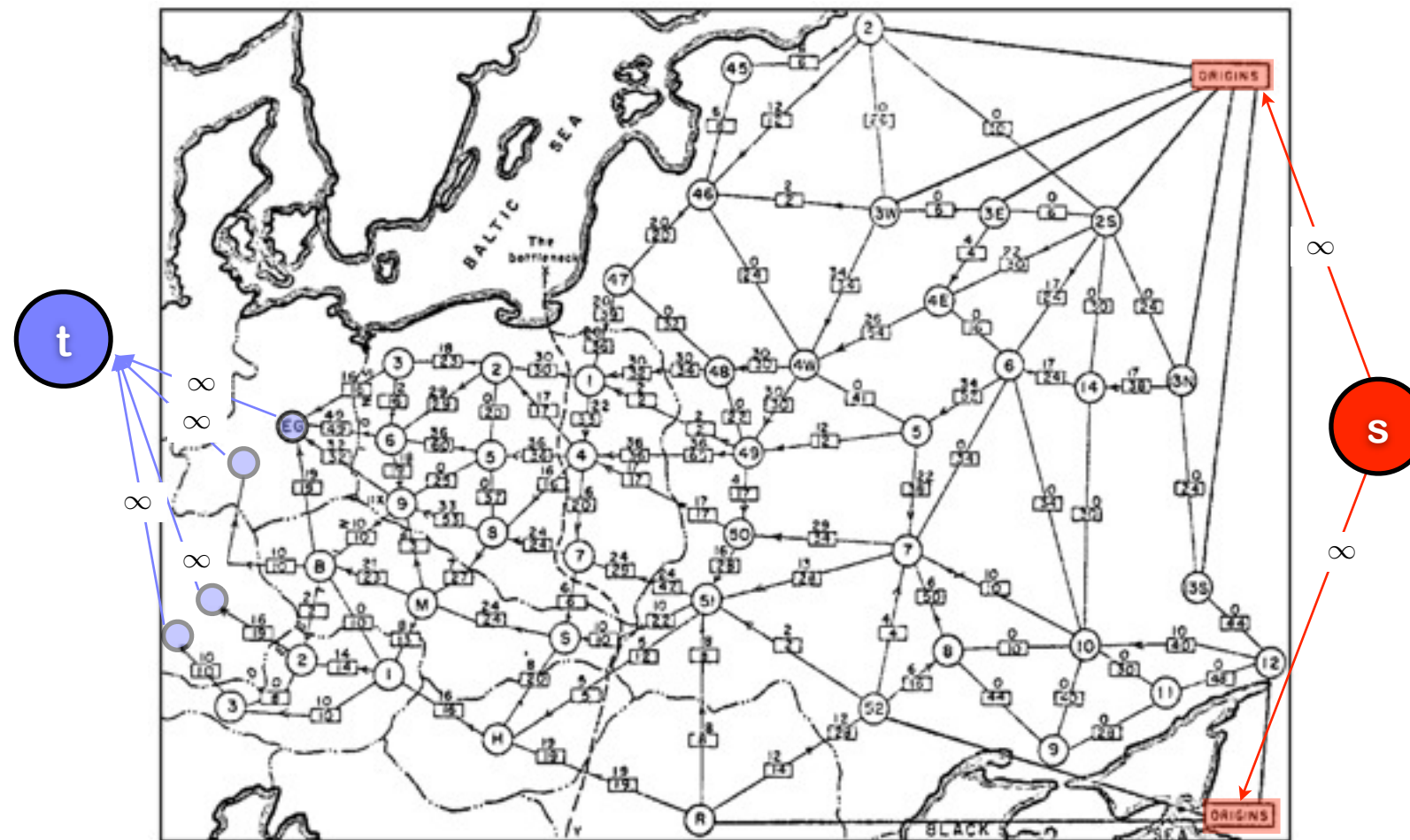
Disrupt flow of goods into satellite countries in the best possible way.



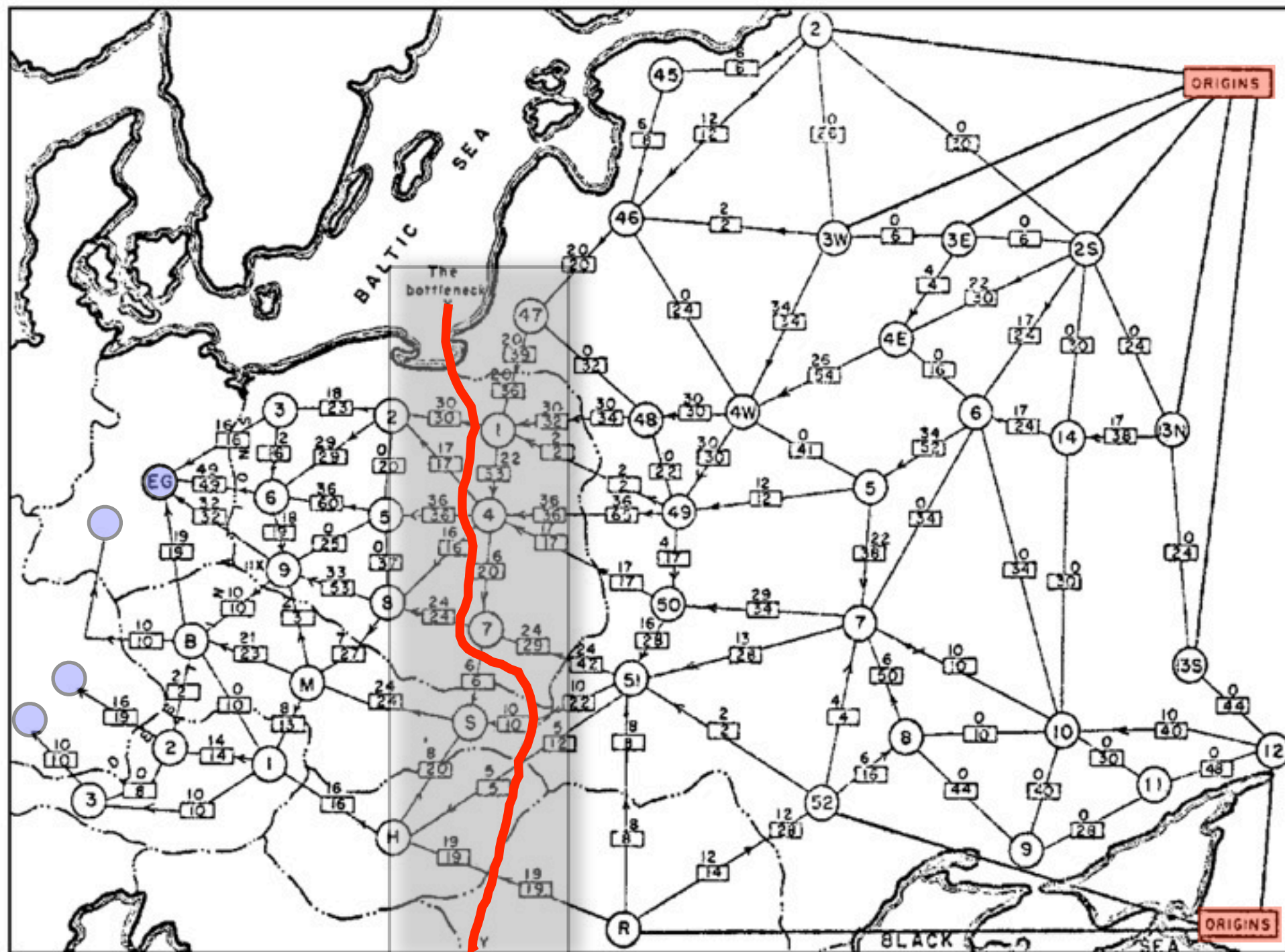
Goal of US Air Force (1950's)

Add node s with edges of capacity infinity to the origins, node t with edges from the destinations with capacity infinity.

Find minimum s,t -cut



Solution



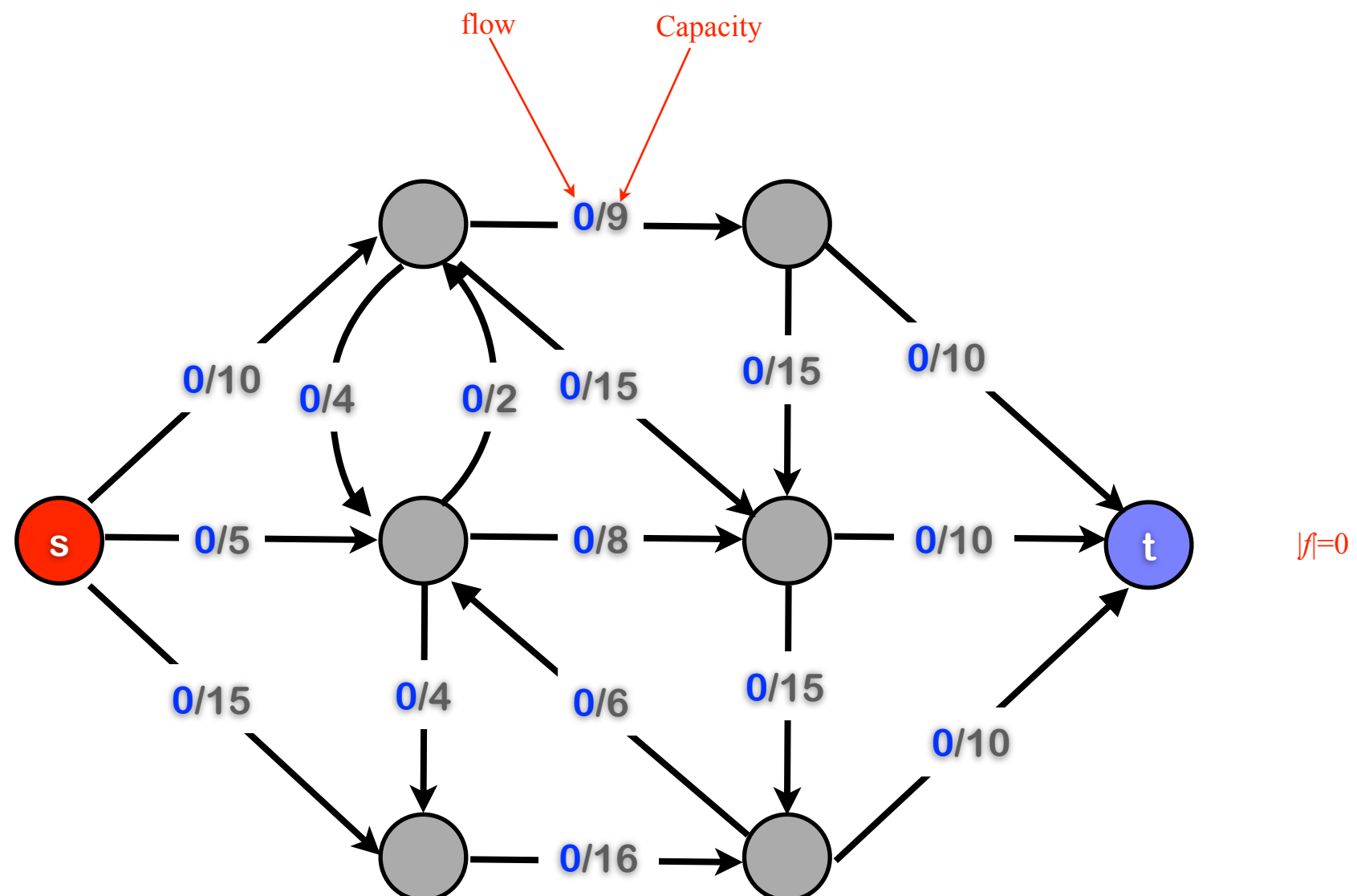
Cut

The Ford-Fulkerson Algorithm

The Ford-Fulkerson Algorithm

Initialization: start with 0-flow

Check that the 0-flow *is* a flow.

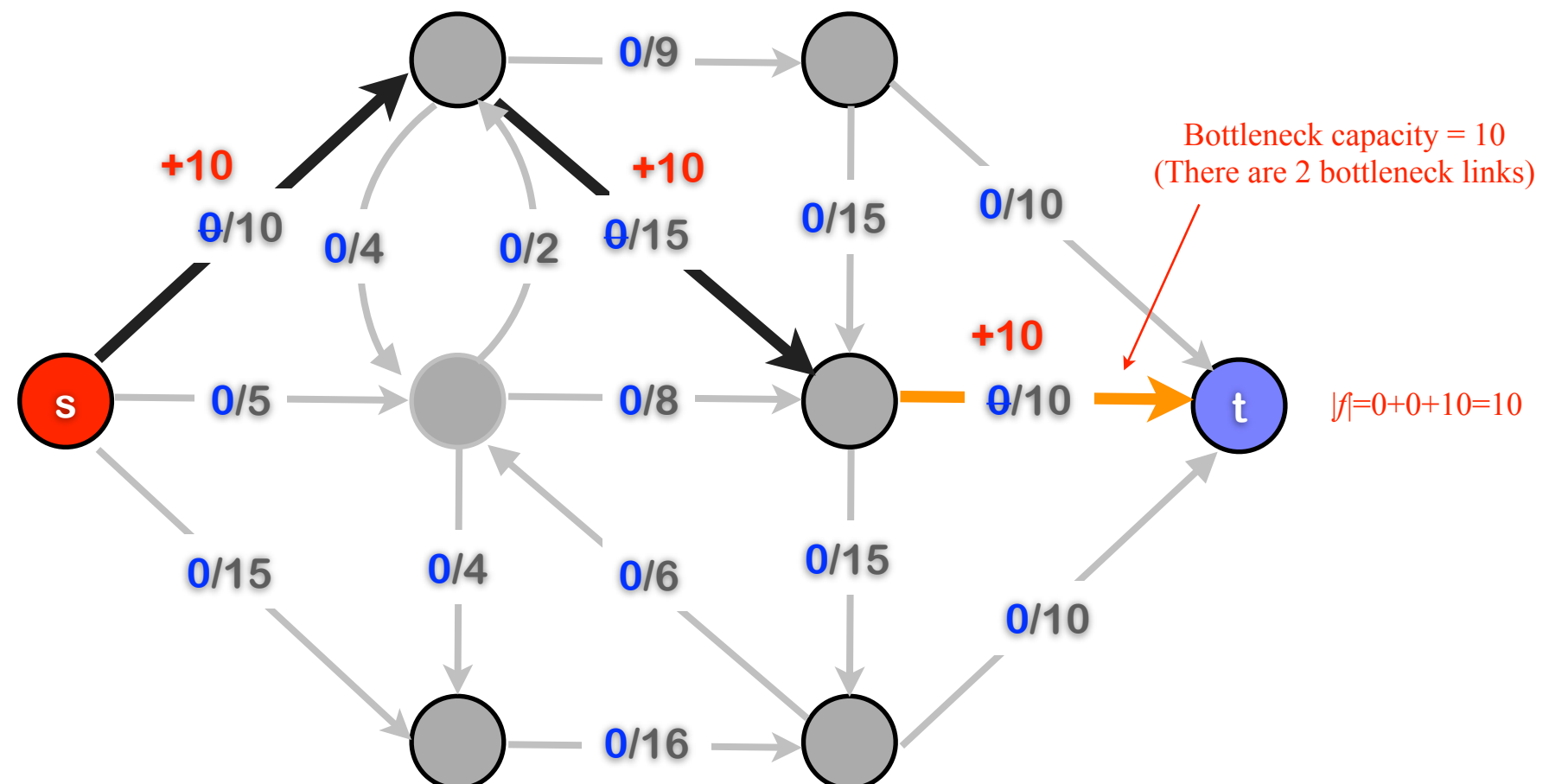


Augmenting Path

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

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

Find smallest capacity on (s,t) path and increase value of flow along the path.

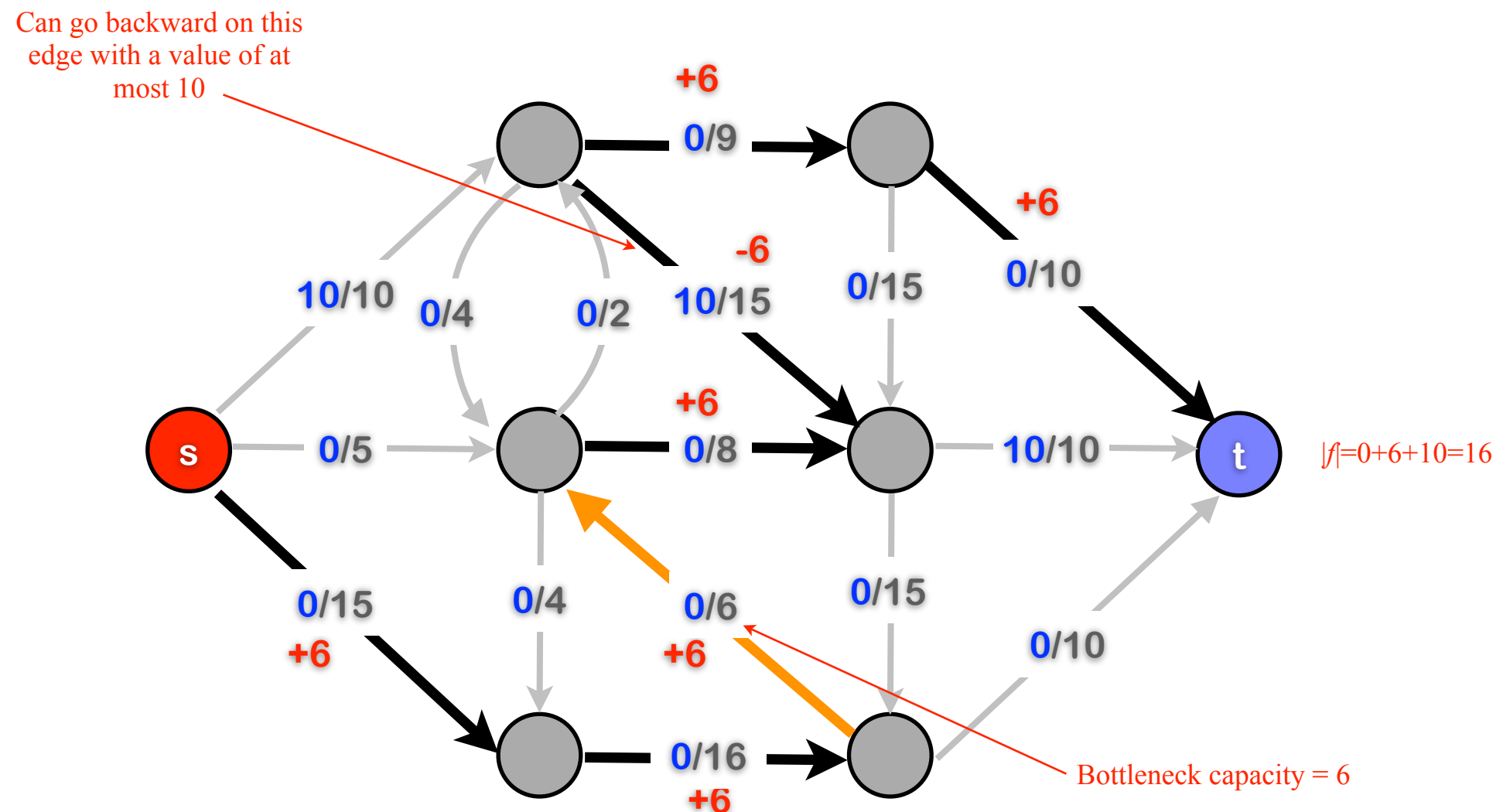


Augmenting Path

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

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

Find smallest capacity on (s,t) path and increase value of flow along the path.

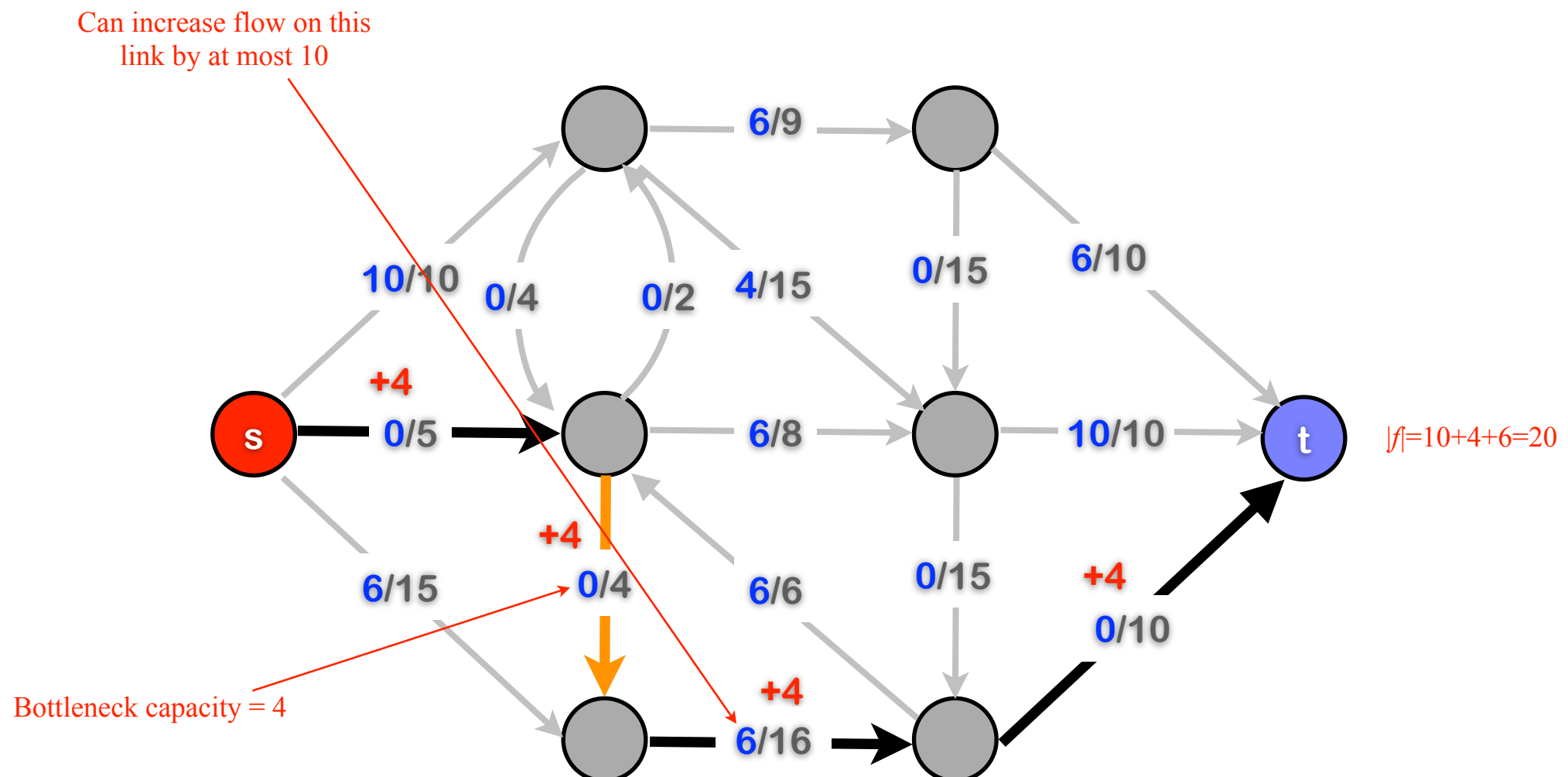


Augmenting Path

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

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

Find smallest capacity on (s,t) path and increase value of flow along the path.

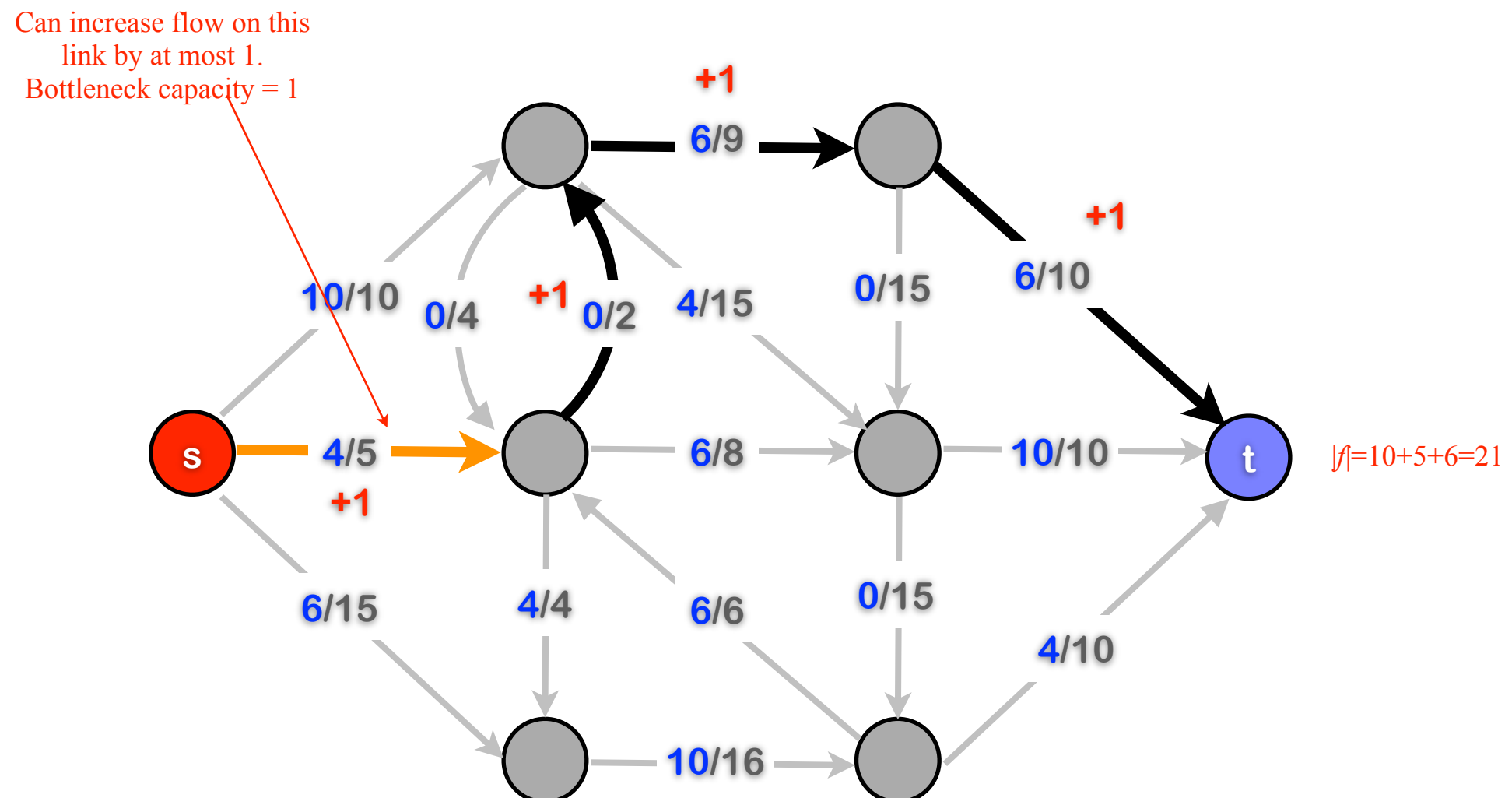


Augmenting Path

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

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

Find smallest capacity on (s,t) path and increase value of flow along the path.

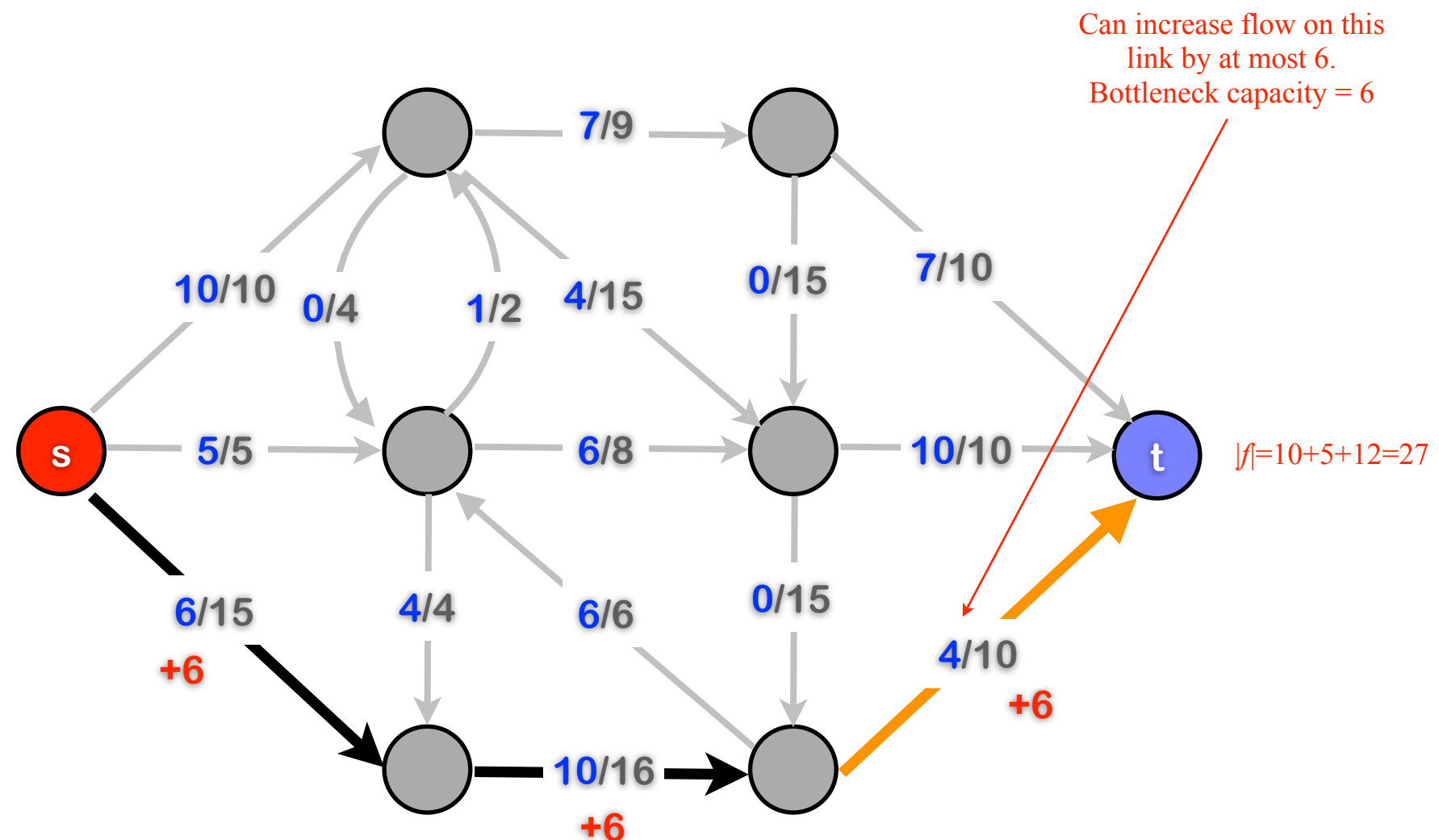


Augmenting Path

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

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

Find smallest capacity on (s,t) path and increase value of flow along the path.

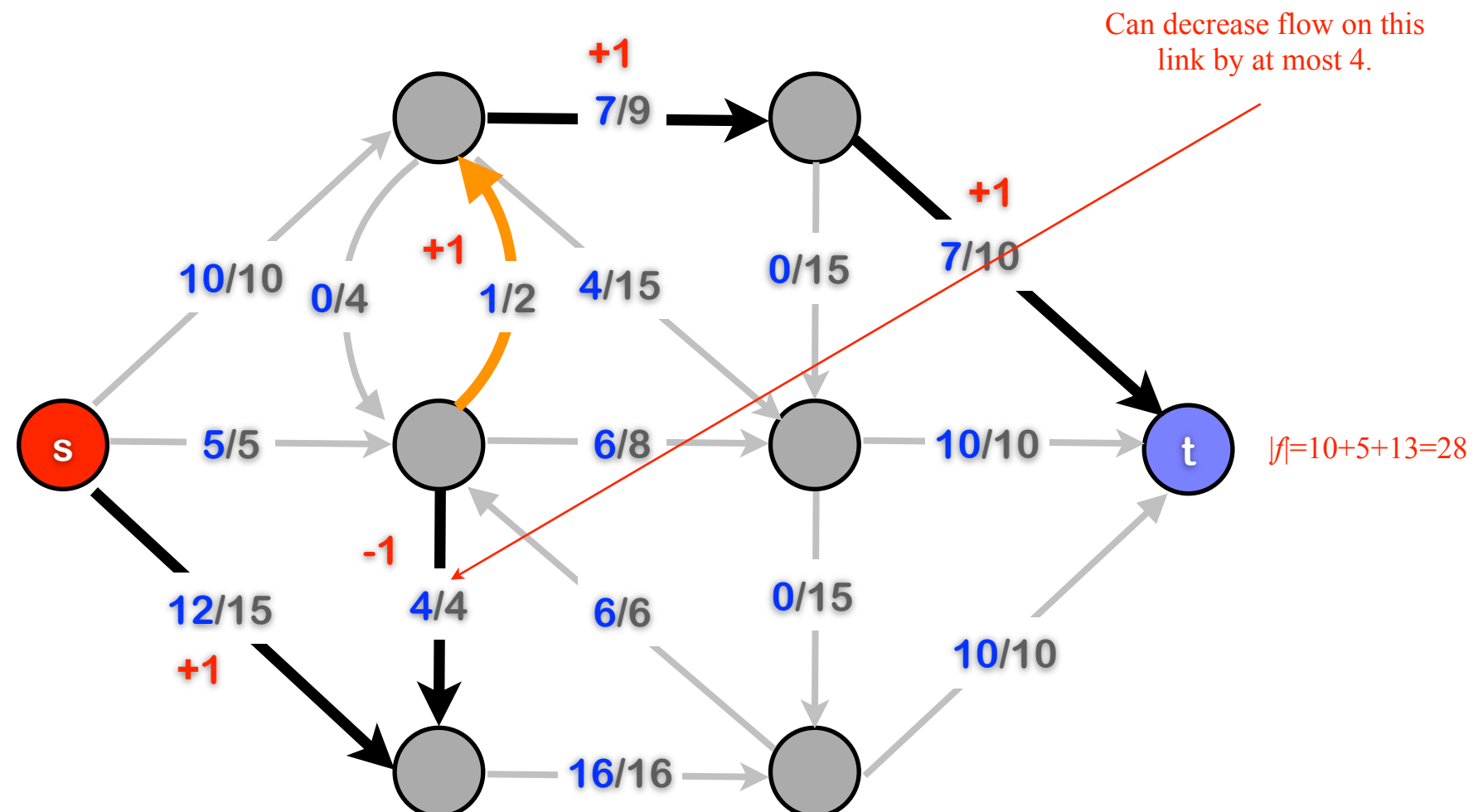


Augmenting Path

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

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

Find smallest capacity on (s,t) path and increase value of flow along the path.

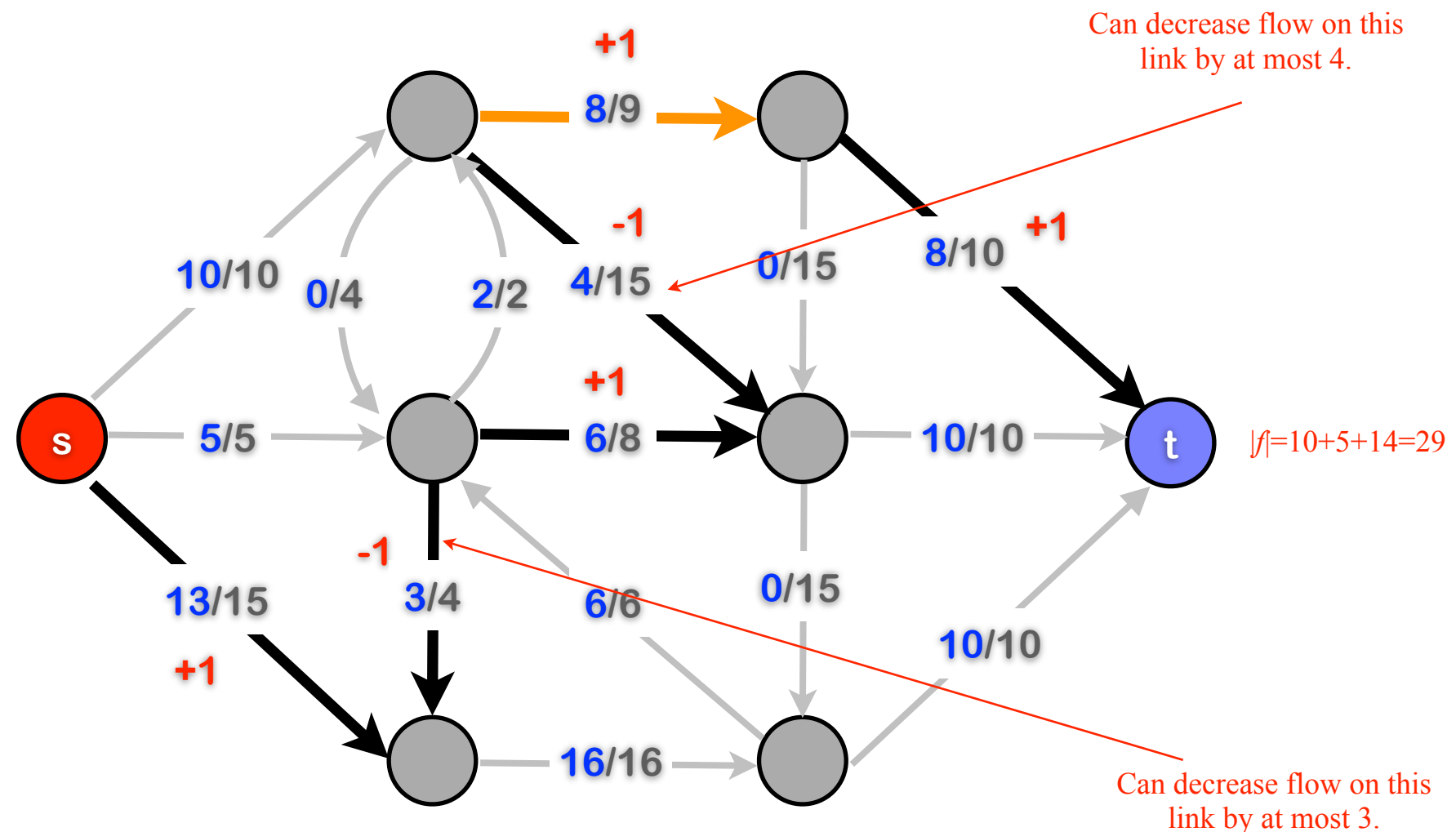


Augmenting Path

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

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

Find smallest capacity on (s,t) path and increase value of flow along the path.

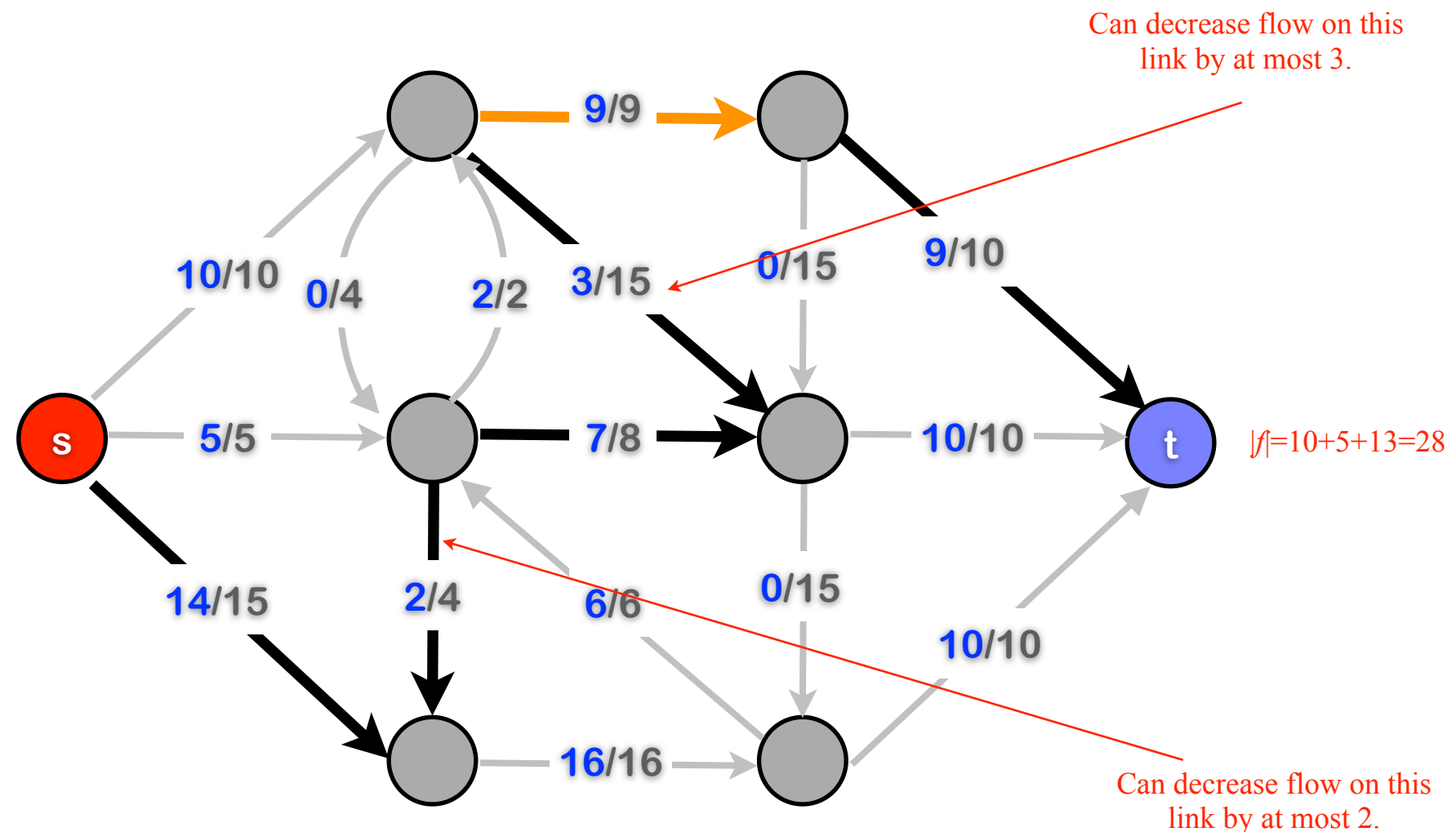


Augmenting Path

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

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

Find smallest capacity on (s,t) path and increase value of flow along the path.



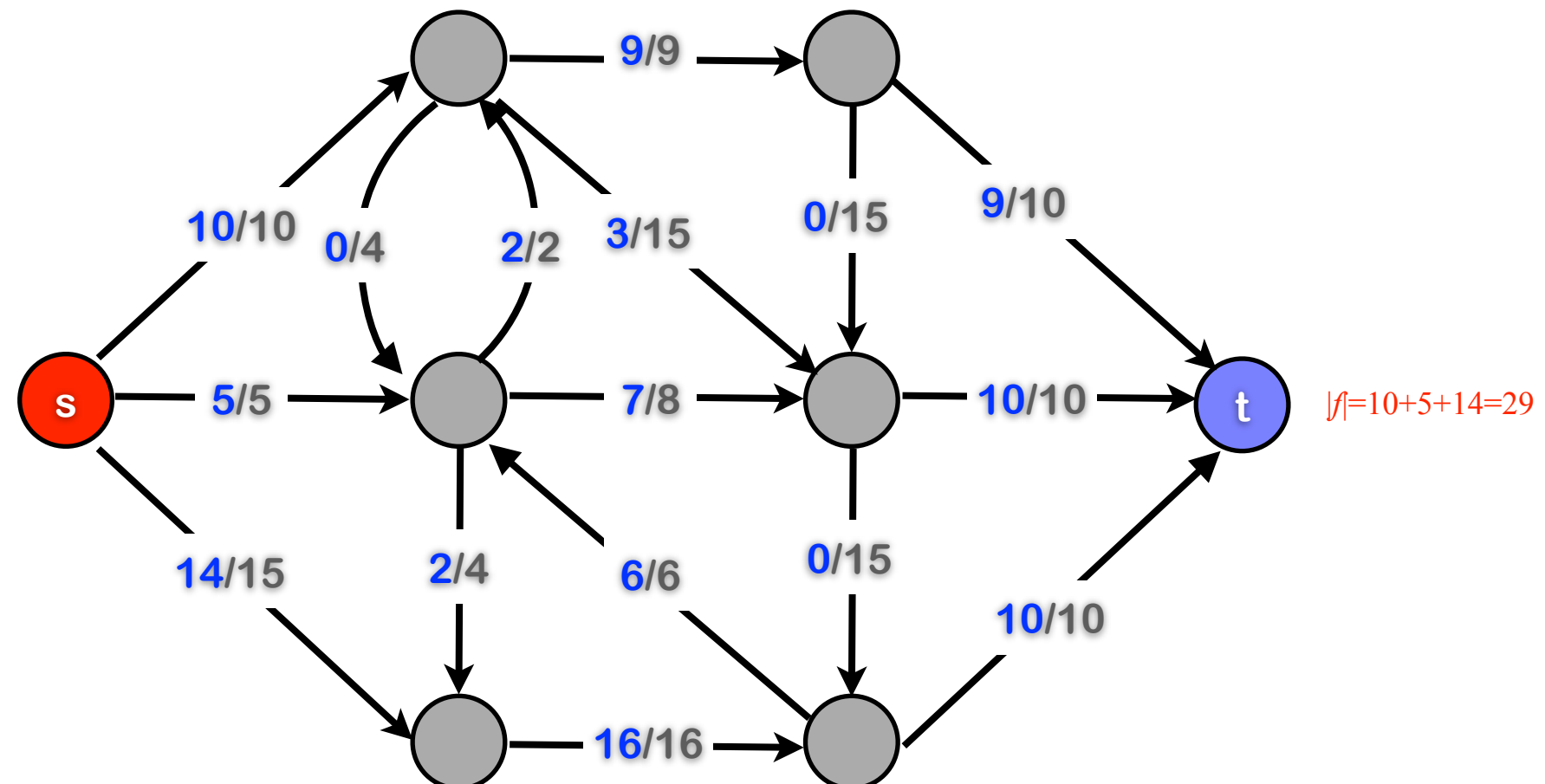
Augmenting Path

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

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

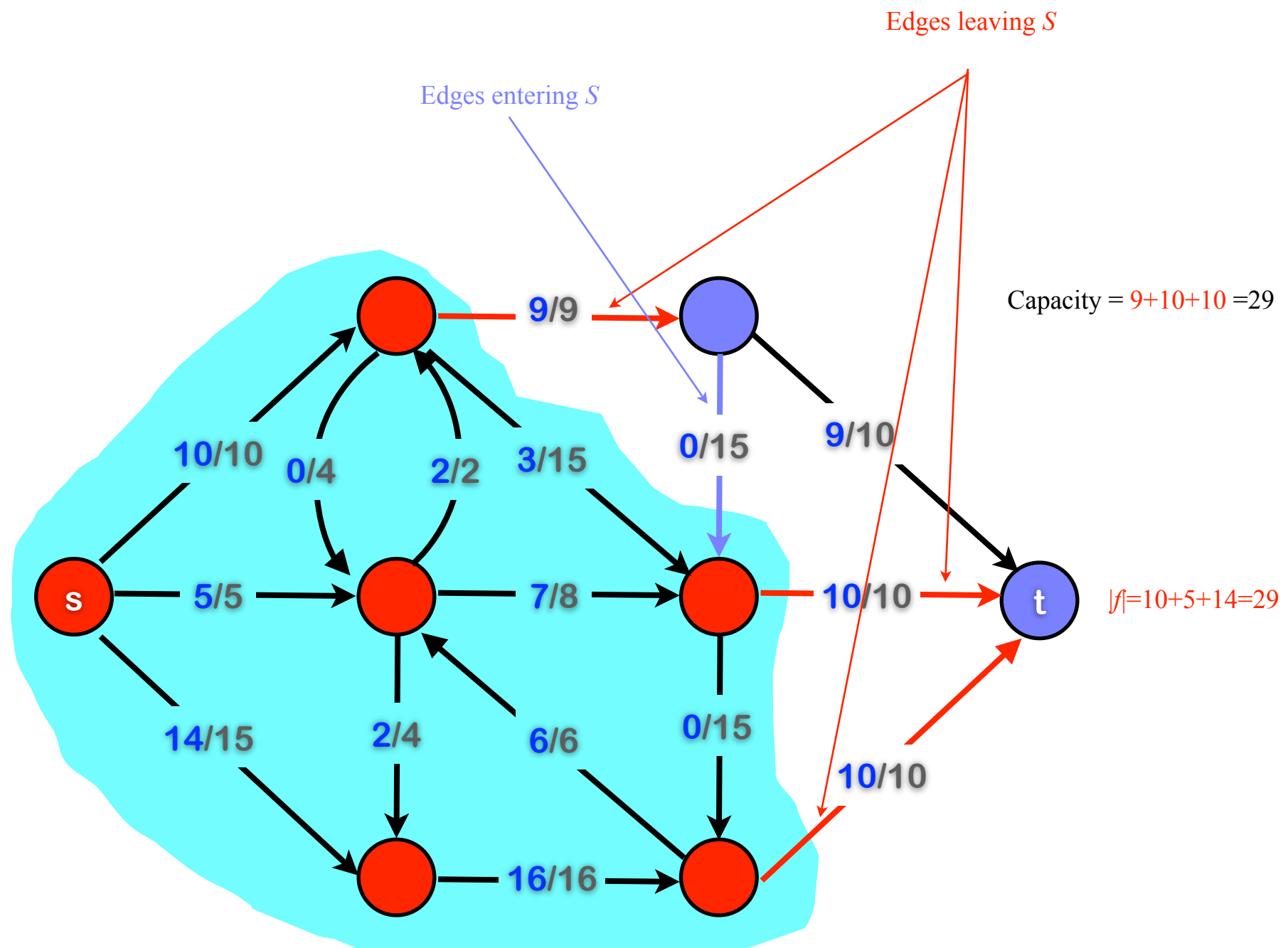
Find smallest capacity on (s,t) path and increase value of flow along the path.

No such path exists at this point.



Augmenting Path

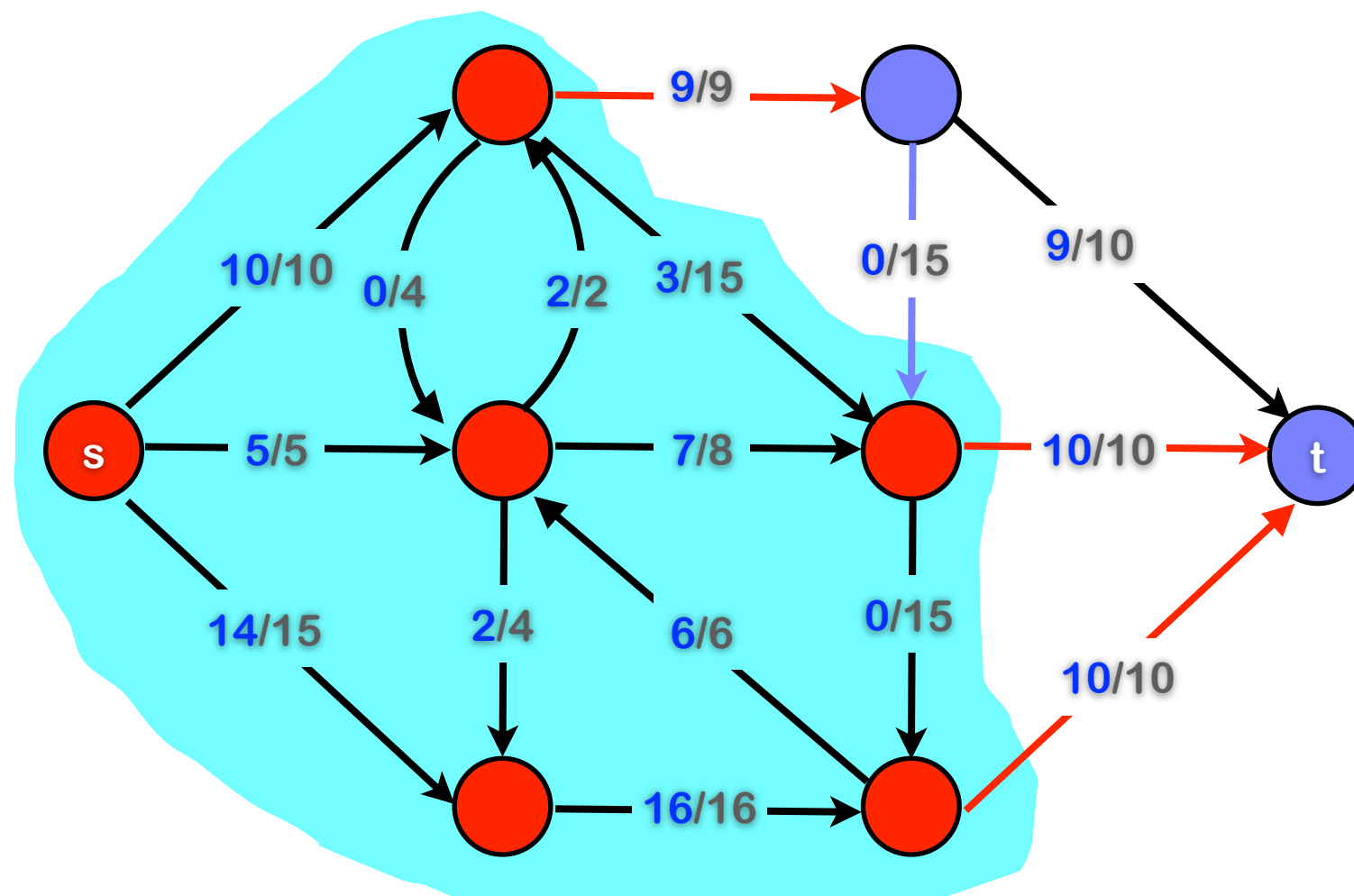
Find node reachable from s in the augmented graph.



Augmenting Path

Find node reachable from s in the augmented graph.

- Capacity of cut = value of flow.
- Edges leaving cut are at capacity.
- Edges entering cut have flow value 0.



Ford-Fulkerson Algorithm

Max-flow

Start with 0-flow.

While (there is an augmenting path from s to t) do

- find augmenting path
- compute bottleneck capacity
- increase flow on the path by bottleneck capacity

When finished, resulting flow is maximal.

Min-cut

If no augmenting path left, then

- Find set of nodes S reachable from s in augmenting graph
- Set $T = V - S$

S and T define minimum cut

Questions:

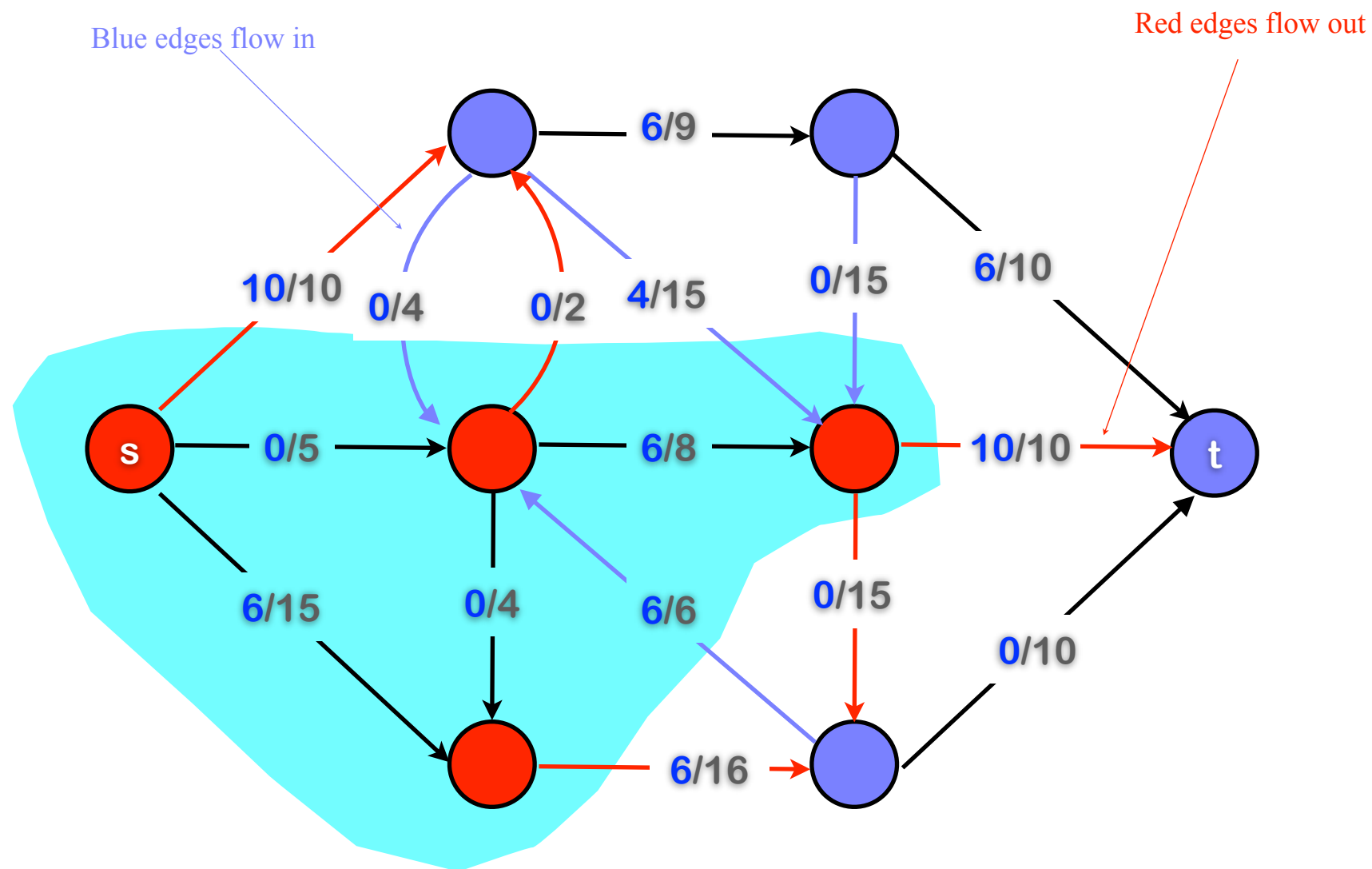
- How do we find an augmenting path?
- If the algorithm terminates, does it so with the correct answers?
- Does it terminate? If so, after how many iterations?

Analysis

Net flow

Definition:

(S, T) an s, t -cut, f flow. The “net flow” of the cut (with respect to f), $\text{netflow}(S, T)$, is the sum of the flows on edges from S to T minus sum of the flows on edges from T to S .



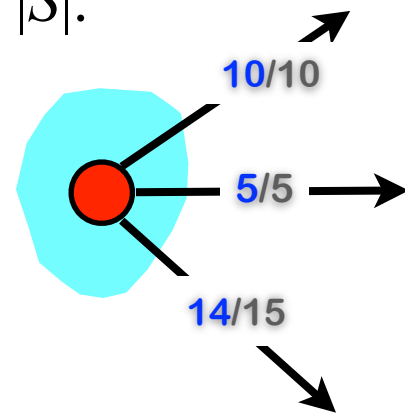
$$\text{netflow}(S, T) = 10+0+0+6+10-(0+4+0+6)=16 = |f|$$

Net flow

(S, T) an s, t -cut, f flow: $\text{netflow}(S, T) = |f|$

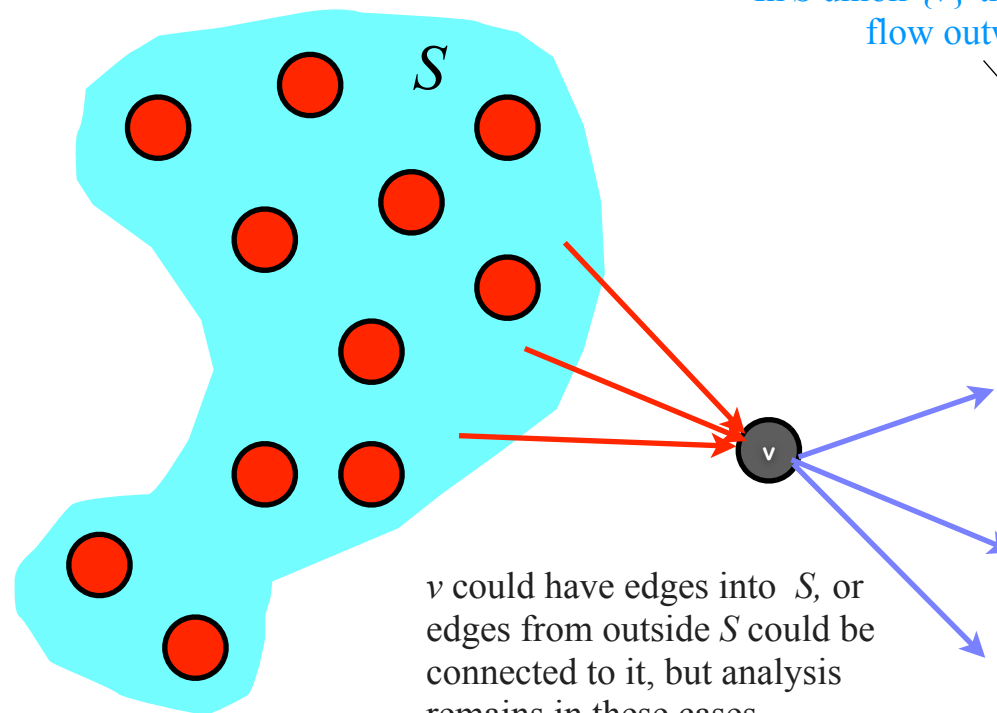
Proof: Induction on $|S|$.

$|S| = 1, S = \{s\}$:



$$\text{netflow}(\{s\}, V - \{s\}) = |f|$$

$|S| + 1$:



In S union $\{v\}$ the blue edges flow outwards

In S union $\{v\}$ the red edges don't flow outwards anymore

Adding v to S :

$$\text{new netflow} = \text{old netflow}$$

- flow on red edges

+ flow on blue edges

$$= \text{old netflow} = |f|$$

By definition of flow

$\} = 0$

Weak Duality

(S, T) an s, t -cut, f flow: $|f| \leq \text{Capacity}(S, T)$

Proof: $|f| = \text{netflow}(S, T) =$

sum of the flows on edges from S to T minus sum of the flows on edges from T to S

\leq Sum of capacities of edges from S to T ≥ 0

since $f(e) \leq \text{cap}(e)$

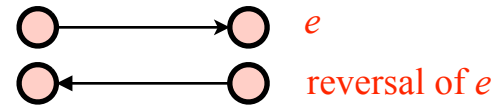
\leq Sum of capacities of edges from S to T $= \text{Capacity}(S, T)$

Residual Graph

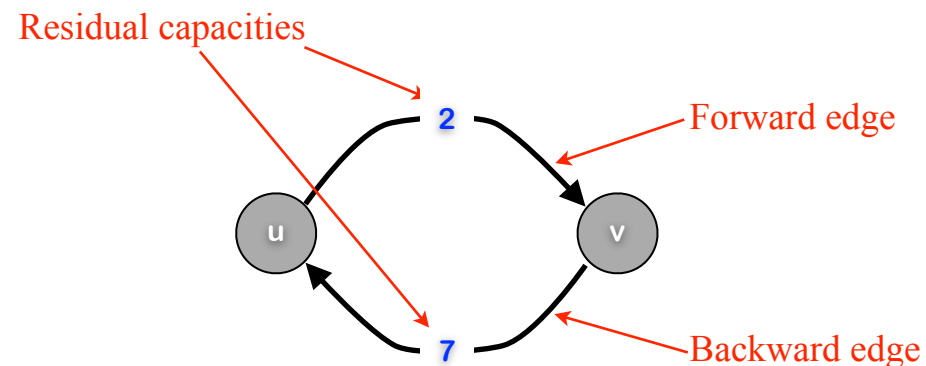
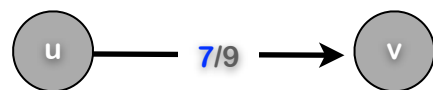
Residual graph with respect to f :

Vertices are the same as in the original graph.

Some edges are the same as in the original graph, some are reversals of edges in the original graph.



- If an edge e has flow value $f(e) < \text{cap}(e)$, then replicate e and give it capacity $\text{cap}(e) - f(e)$ \longleftarrow We can still put this much flow on the edge
- If an edge e has flow value $f(e) > 0$, then add the reversal of e to the residual graph and give it capacity $f(e)$ \longleftarrow We can decrease flow on this edge by at most this much

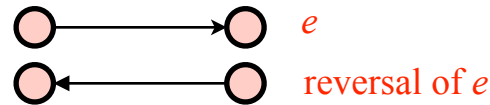



Residual Graph

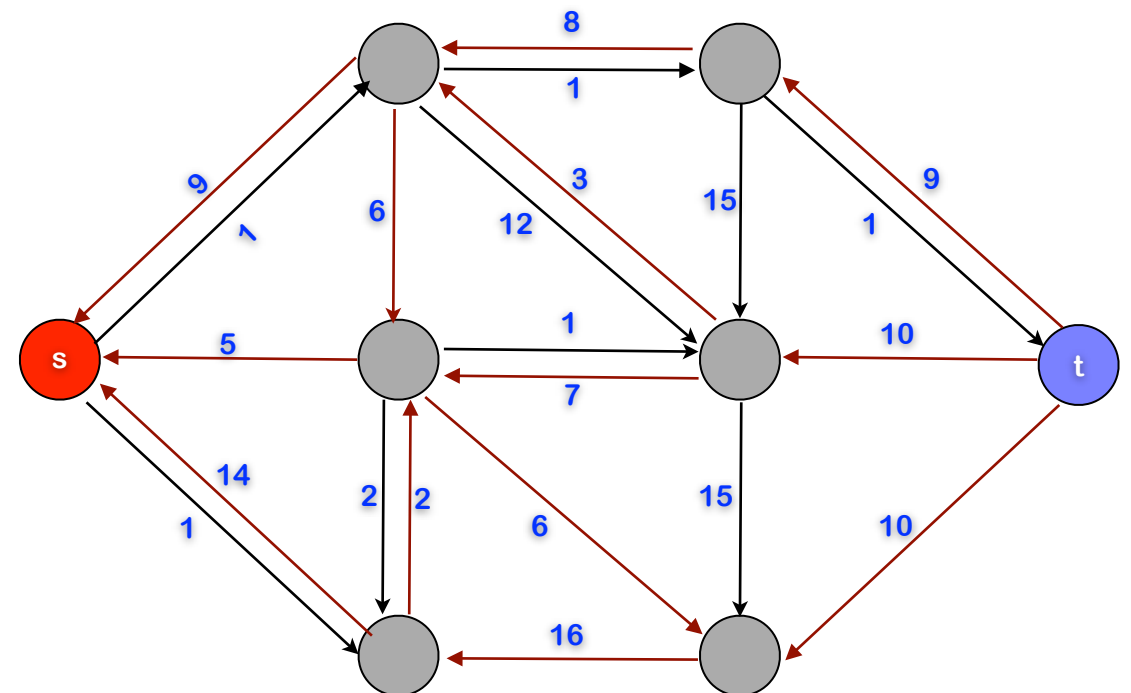
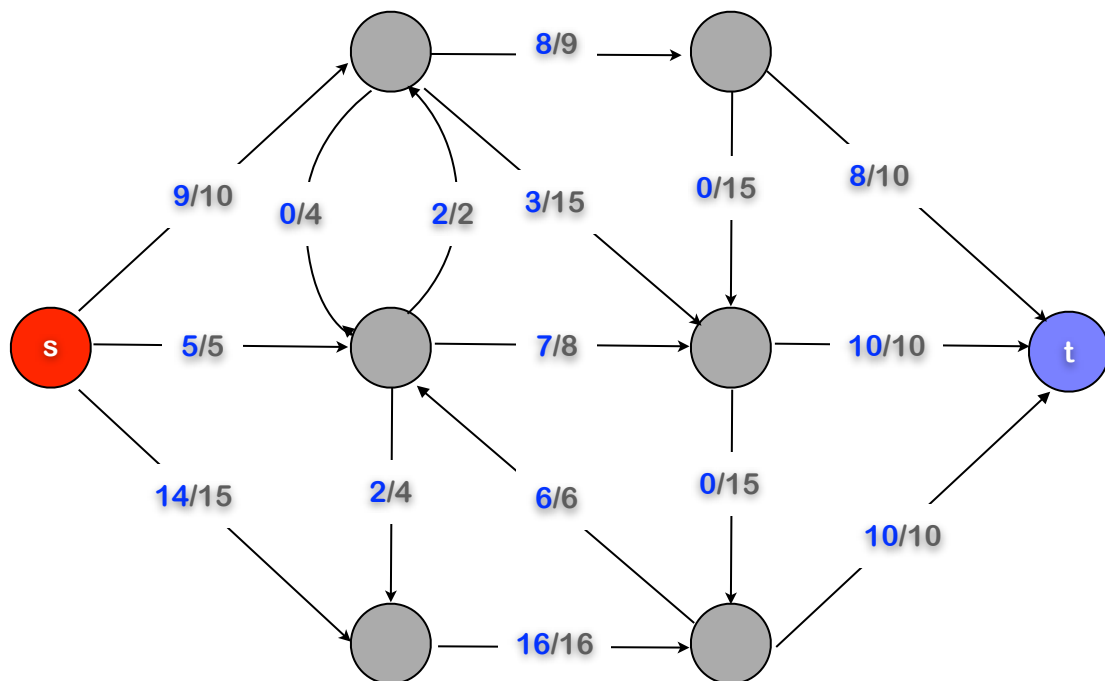
Residual graph with respect to f :

Vertices are the same as in the original graph.

Some edges are the same as in the original graph, some are reversals of edges in the original graph.



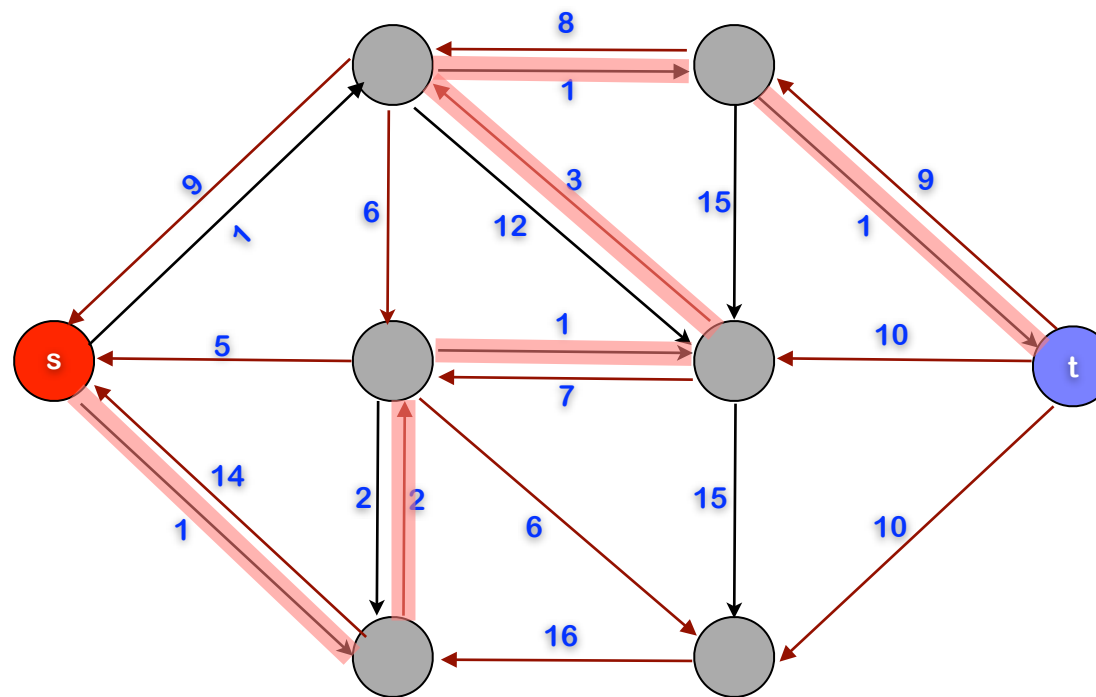
- If an edge e has flow value $f(e) < \text{cap}(e)$, then replicate e and give it capacity $\text{cap}(e) - f(e)$ \longleftarrow We can still put this much flow on the edge
- If an edge e has flow value $f(e) > 0$, then add the reversal of e to the residual graph and give it capacity $f(e)$ \longleftarrow We can decrease flow on this edge by at most this much



Augmenting Path

Path from s to t in the residual graph.

Called “augmenting” because we can augment value of flow by smallest capacity on the path.



MaxFlow-MinCut Theorem

MaxFlow-MinCut-Theorem

Let $G = (V, E)$ be a graph, c an assignment of capacities, s, t nodes in V , and f a flow.

The following are equivalent:

- (a) f is a maximal flow
- (b) The residual graph with respect to f does not contain an augmenting path
- (c) There is a cut (S, T) in G such that $|f| = \text{Capacity}(S, T)$

Proof:

(a) \implies (b): we show contrapositive: $\neg(b) \implies \neg(a)$

If the residual graph has an augmenting path, then we can increase value of f by smallest capacity on the path, so f is not a maximal flow.

MaxFlow-MinCut Theorem

MaxFlow-MinCut-Theorem

Let $G = (V, E)$ be a graph, c an assignment of capacities, s, t nodes in V , and f a flow.

The following are equivalent:

- (a) f is a maximal flow
- (b) The residual graph with respect to f does not contain an augmenting path
- (c) There is a cut (S, T) in G such that $|f| = \text{Capacity}(S, T)$

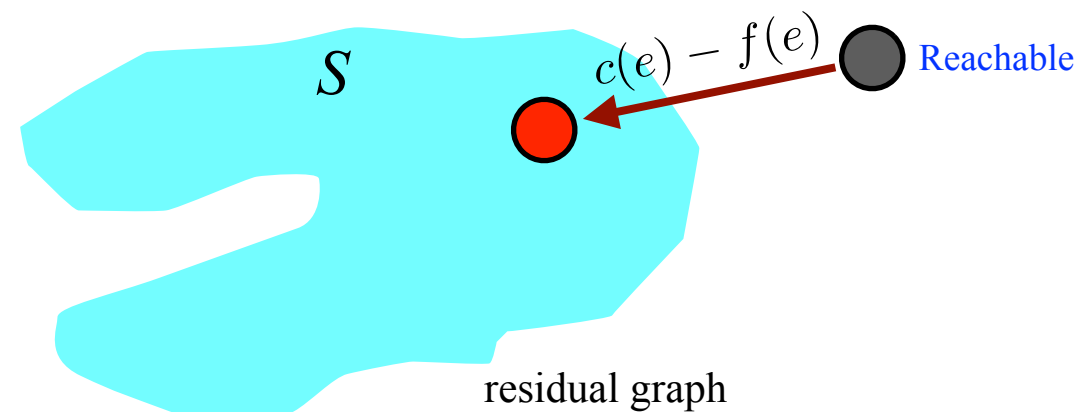
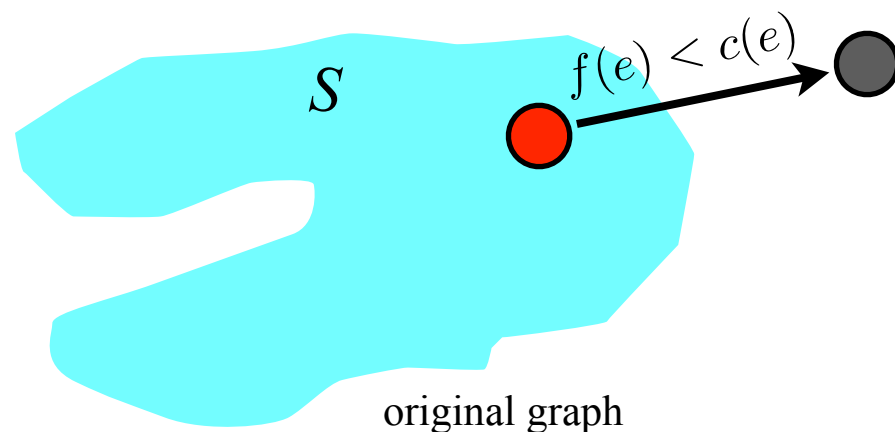
Proof:

(b) \implies (c):

S = set of nodes reachable from s in the residual graph, $T = V - S$. Then (S, T) is an s, t -cut.

Every edge flowing out of S in the original graph must be at capacity:

Otherwise we can reach a node outside of S in the residual graph.



MaxFlow-MinCut Theorem

MaxFlow-MinCut-Theorem

Let $G = (V, E)$ be a graph, c an assignment of capacities, s, t nodes in V , and f a flow.

The following are equivalent:

- (a) f is a maximal flow
- (b) The residual graph with respect to f does not contain an augmenting path
- (c) There is a cut (S, T) in G such that $|f| = \text{Capacity}(S, T)$

Proof:

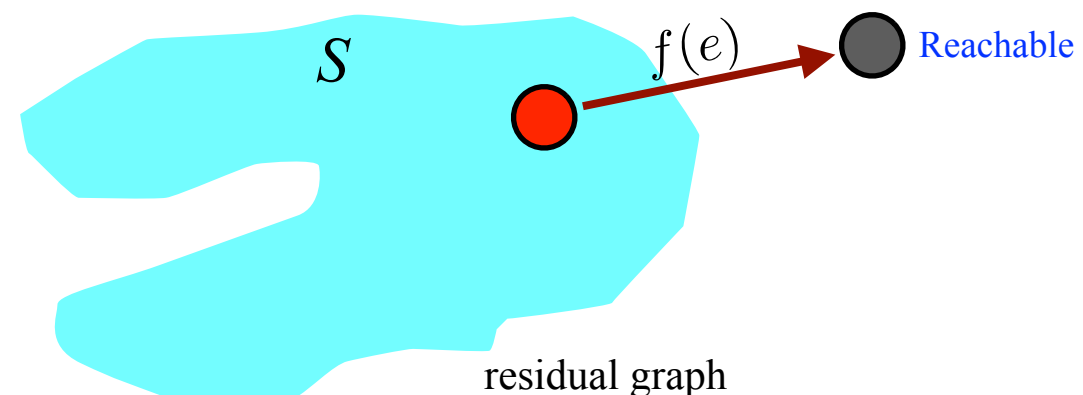
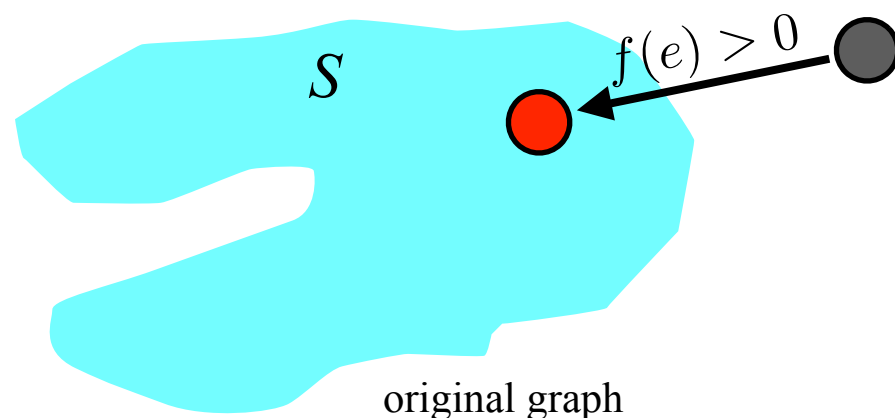
(b) \implies (c):

S = set of nodes reachable from s in the residual graph, $T = V - S$. Then (S, T) is an s, t -cut.

Every edge flowing out of S in the original graph must be at capacity

Every edge flowing into S in the original graph must have flow 0

Otherwise we can reach a node outside of S in the residual graph.



MaxFlow-MinCut Theorem

MaxFlow-MinCut-Theorem

Let $G = (V, E)$ be a graph, c an assignment of capacities, s, t nodes in V , and f a flow.

The following are equivalent:

- (a) f is a maximal flow
- (b) The residual graph with respect to f does not contain an augmenting path
- (c) There is a cut (S, T) in G such that $|f| = \text{Capacity}(S, T)$

Proof:

(b) \implies (c):

S = set of nodes reachable from s in the residual graph, $T = V - S$. Then (S, T) is an s, t -cut.

Every edge flowing out of S in the original graph must be at capacity

Every edge flowing into S in the original graph must have flow 0

$$\text{netflow}(S, T) = |f| = \text{sum of capacities of outflowing edges} = \text{Capacity}(S, T)$$

MaxFlow-MinCut Theorem

MaxFlow-MinCut-Theorem

Let $G = (V, E)$ be a graph, c an assignment of capacities, s, t nodes in V , and f a flow.

The following are equivalent:

- (a) f is a maximal flow
- (b) The residual graph with respect to f does not contain an augmenting path
- (c) There is a cut (S, T) in G such that $|f| = \text{Capacity}(S, T)$

Proof:

(c) \implies (a):

Weak duality: $|f|$ is always less than or equal to capacity of a cut

If value of flow equal to the capacity of some cut, then it cannot be further improved (otherwise weak duality would be violated).

So f is maximal flow.

Ford-Fulkerson Algorithm

Max-flow

Start with 0-flow.

While (there is an augmenting path from s to t) do

- find augmenting path
- compute bottleneck capacity
- increase flow on the path by bottleneck capacity

When finished, resulting flow is maximal.

Min-cut

If no augmenting path left, then

- Find set of nodes S reachable from s in augmenting graph
- Set $T = V - S$

S and T define minimum cut

Questions:

- How do we find an augmenting path? **BFS works well.**
- If the algorithm terminates, does it so with the correct answers? **Yes.**
- Does it terminate? If so, after how many iterations?

Integer Capacities

If the capacities are integers, then the maximum flow is integer as well.

Proof:

Induction:

- Start: original flow is integer valued (0-flow), capacities in residual graph are integer valued (equal to original capacities)
- Step: Value of flow is augmented by some capacity in residual graph. Induction hypothesis: prior flow is integer valued, capacities on residual graph are integer valued. So, new flow is integer valued.

Number of iterations of the algorithm is at most equal to the value of maximum flow, if capacities are integers.

Proof:

Value of flow increases by at least one in every iteration.

Ford-Fulkerson Algorithm

Max-flow

Start with 0-flow.

While (there is an augmenting path from s to t) do

- find augmenting path
- compute bottleneck capacity
- increase flow on the path by bottleneck capacity

When finished, resulting flow is maximal.

Min-cut

If no augmenting path left, then

- Find set of nodes S reachable from s in augmenting graph
- Set $T = V - S$

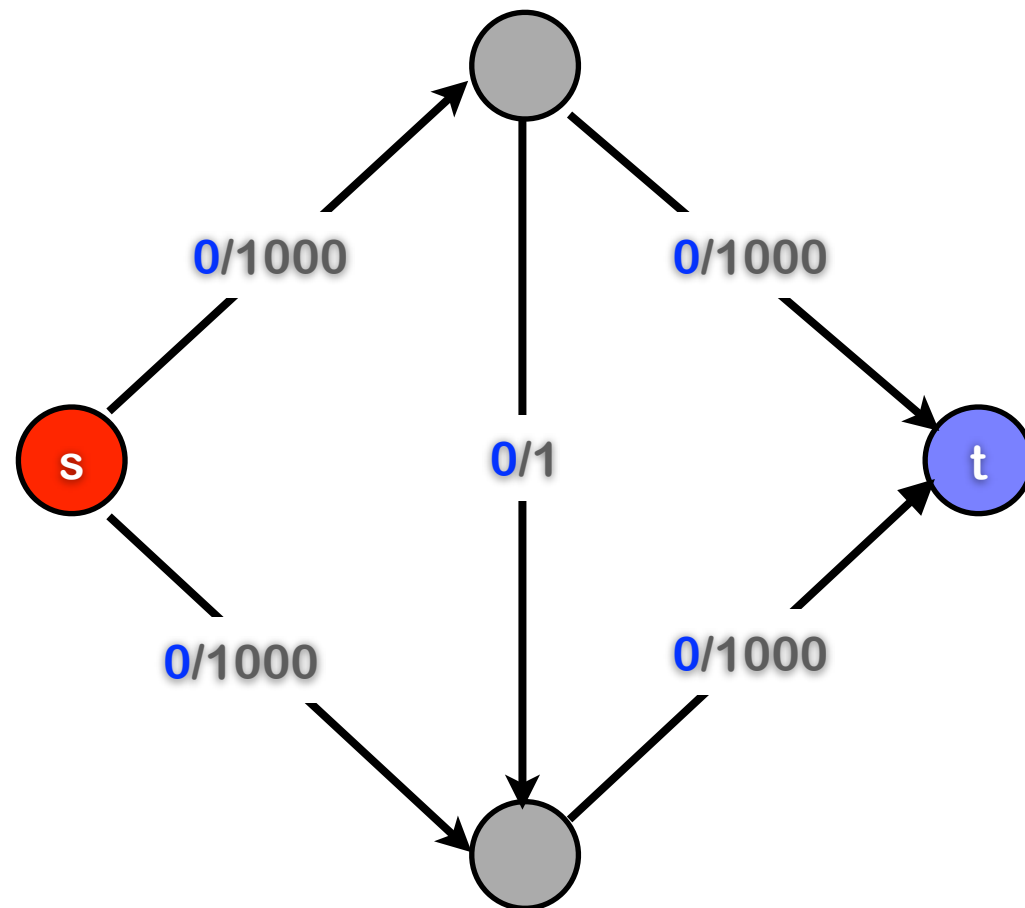
S and T define minimum cut

Questions:

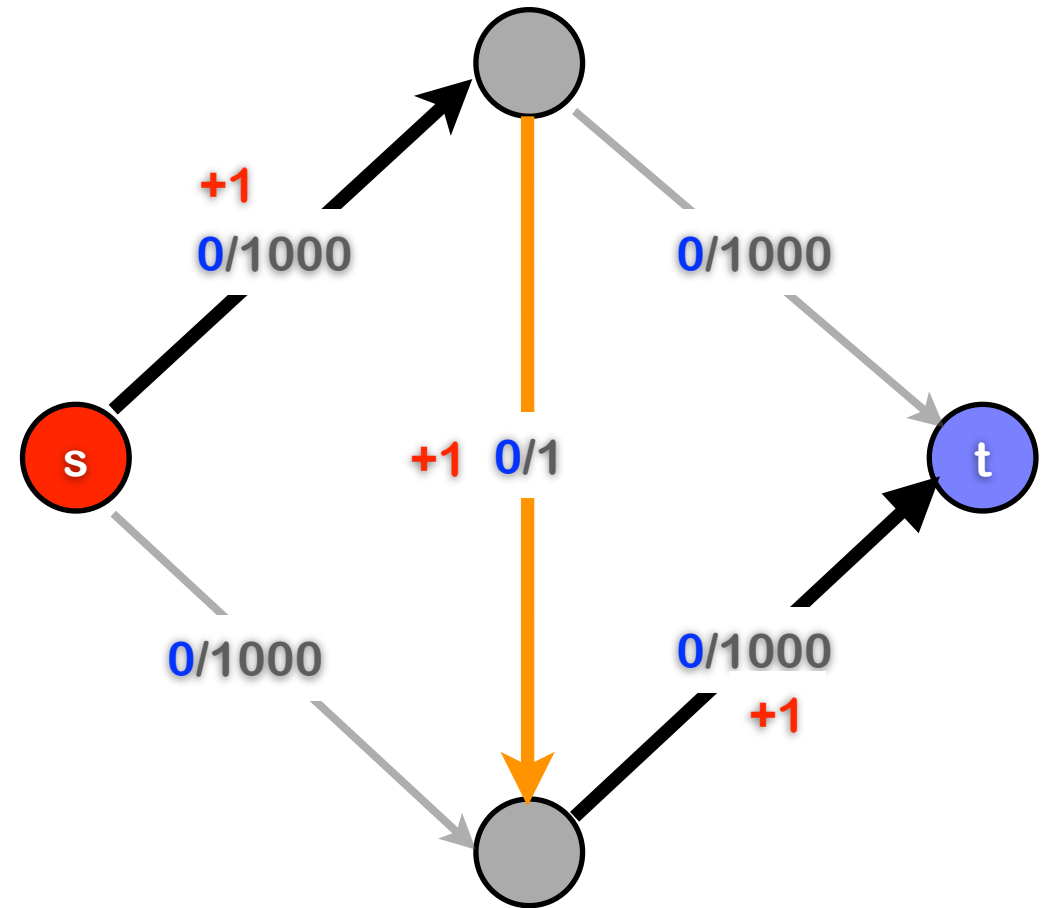
- How do we find an augmenting path? **BFS works well.**
- If the algorithm terminates, does it so with the correct answers? **Yes.**
- Does it terminate? If so, after how many iterations?

Yes, if capacities are integers

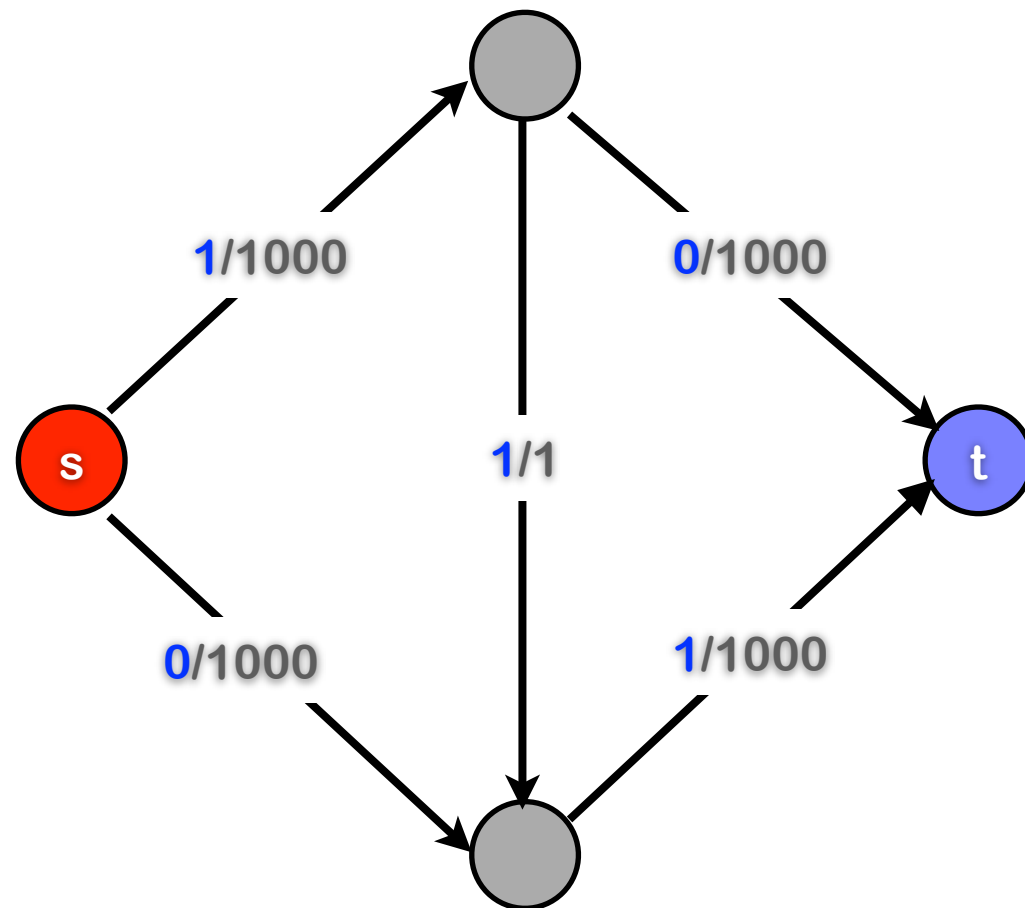
Bad News



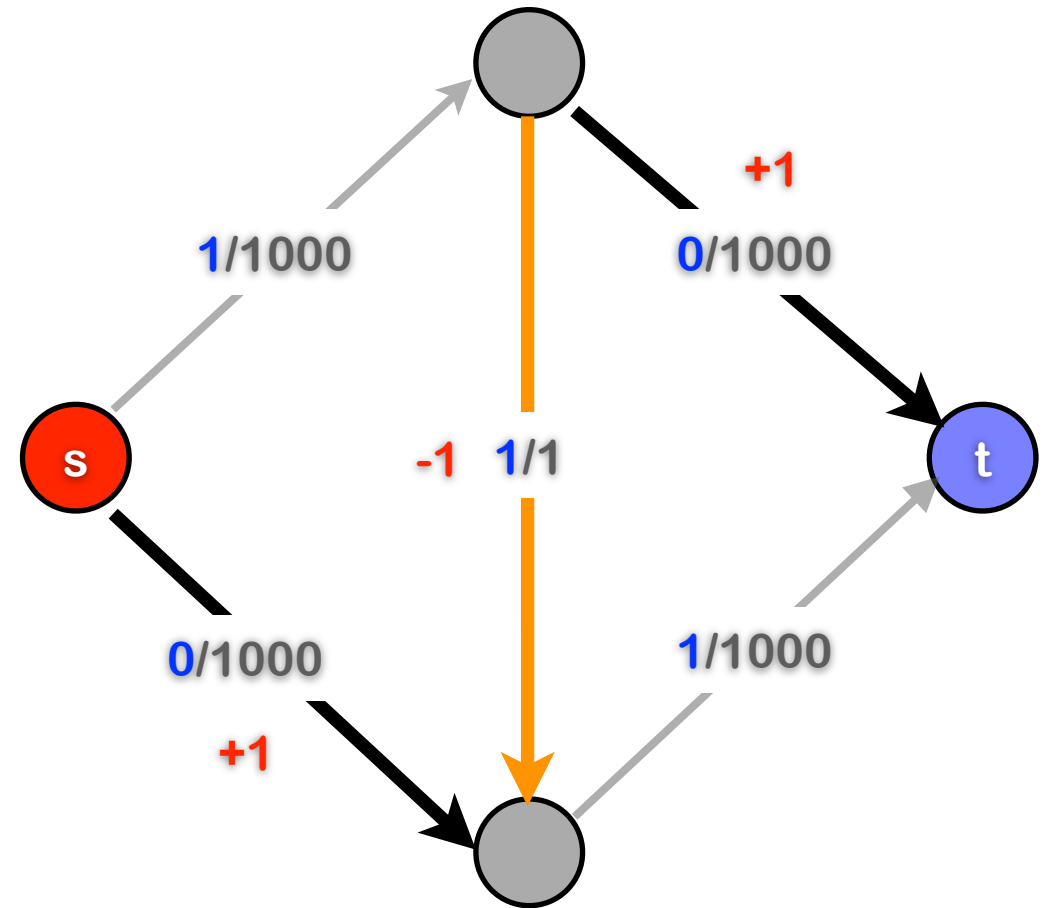
$$|f| = 0$$



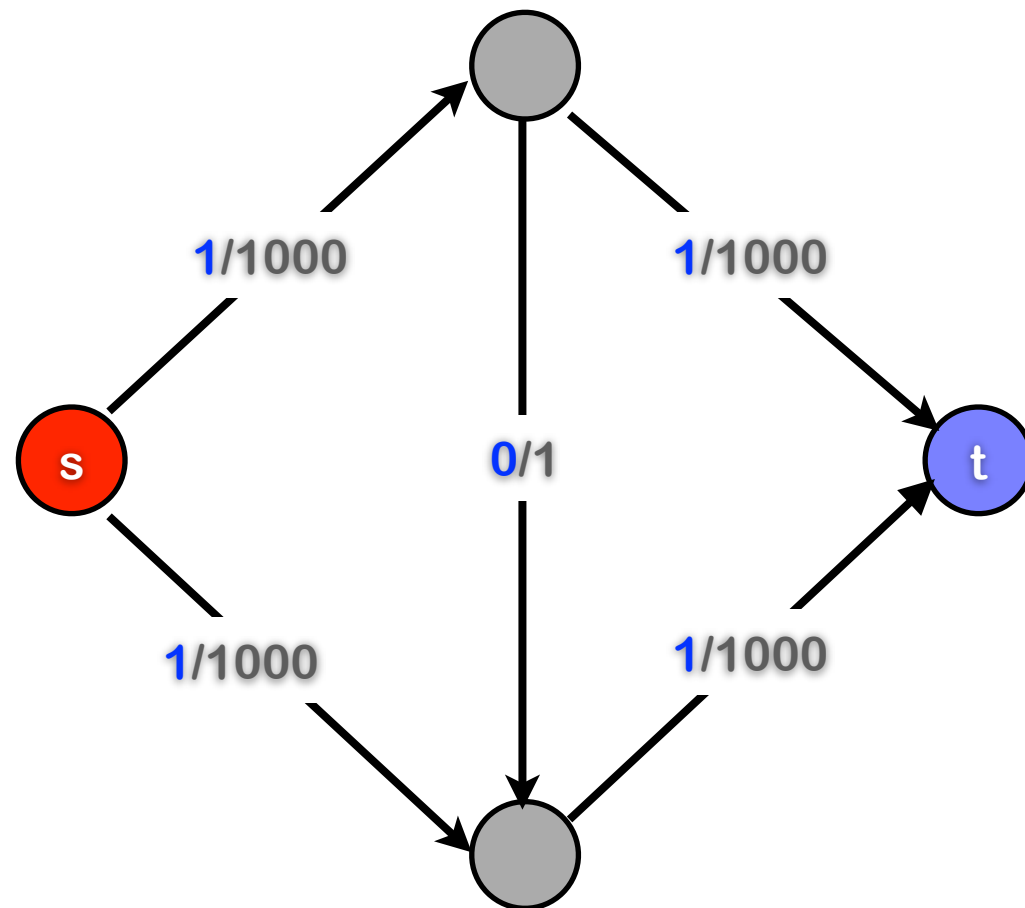
Bad News



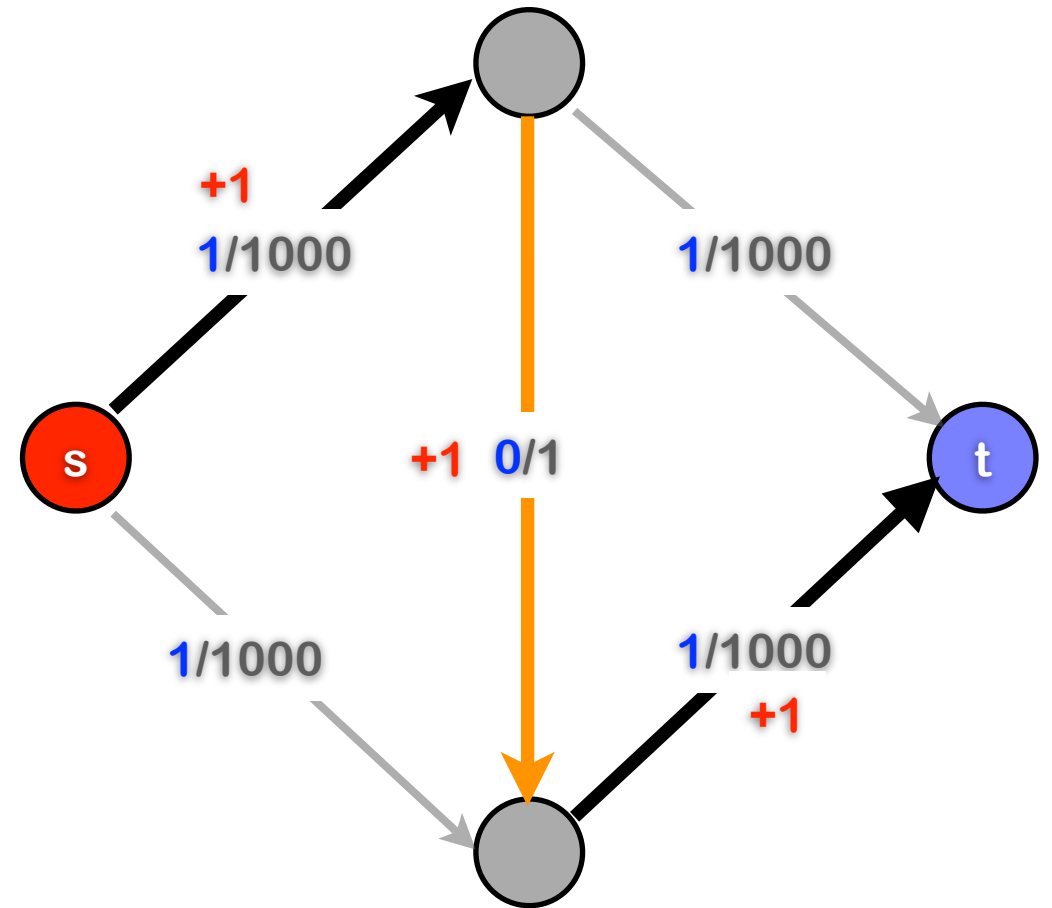
$$|f| = 1$$



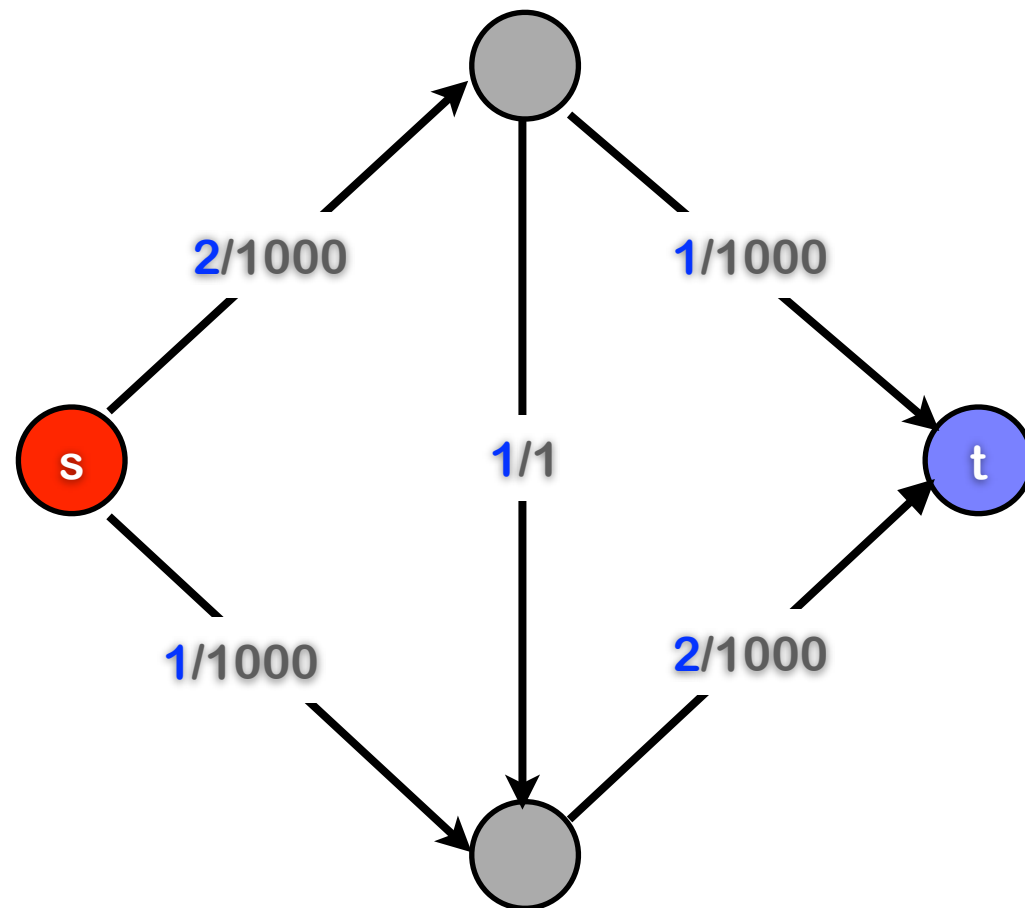
Bad News



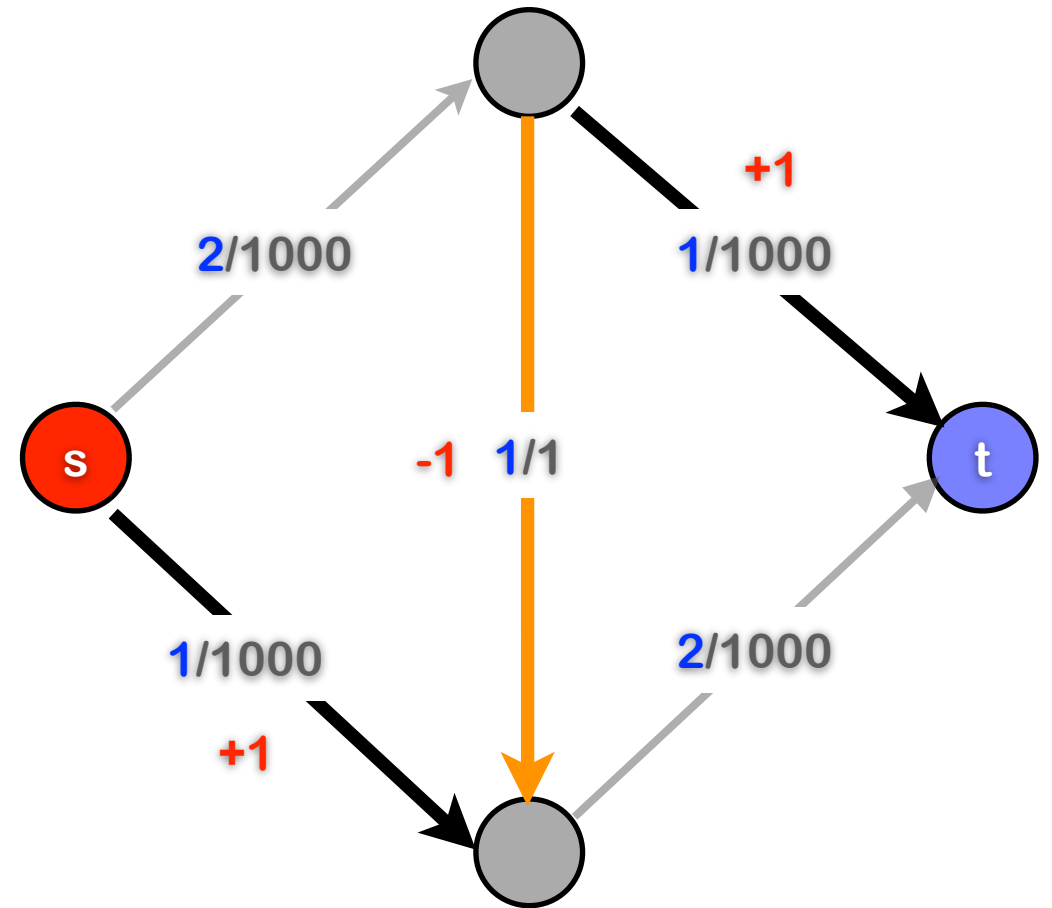
$$|f| = 2$$



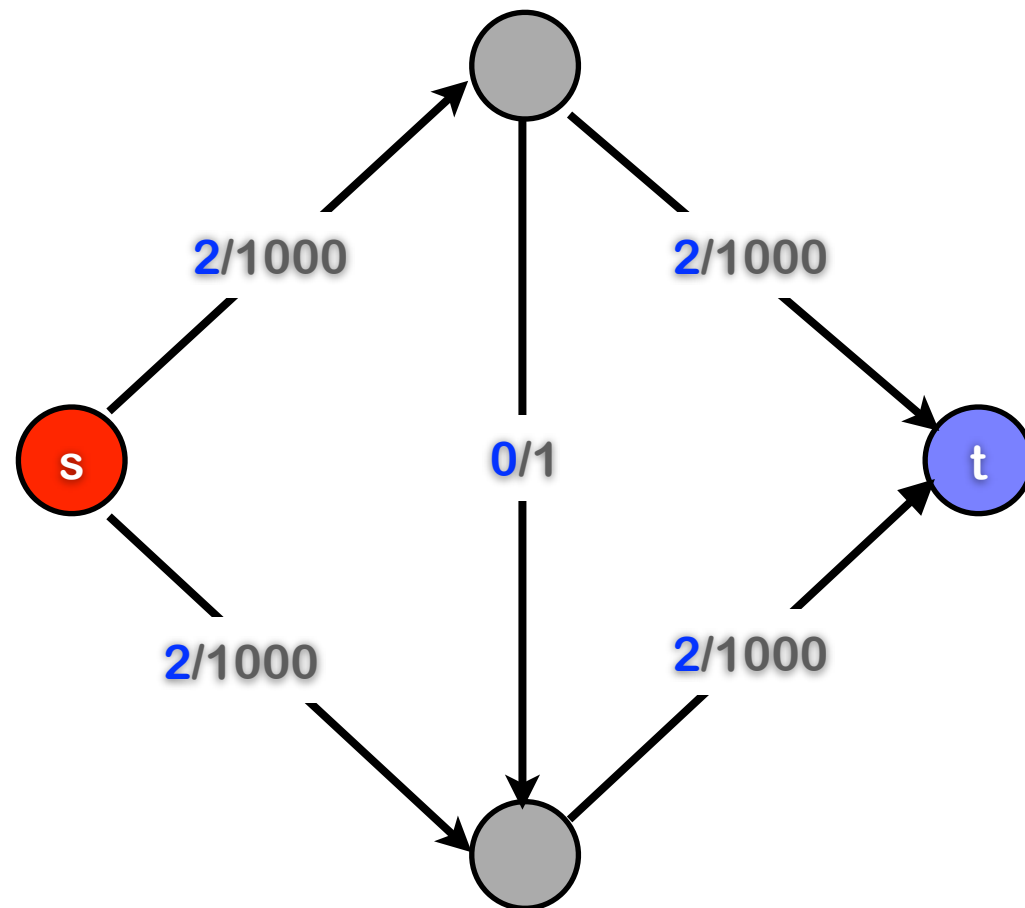
Bad News



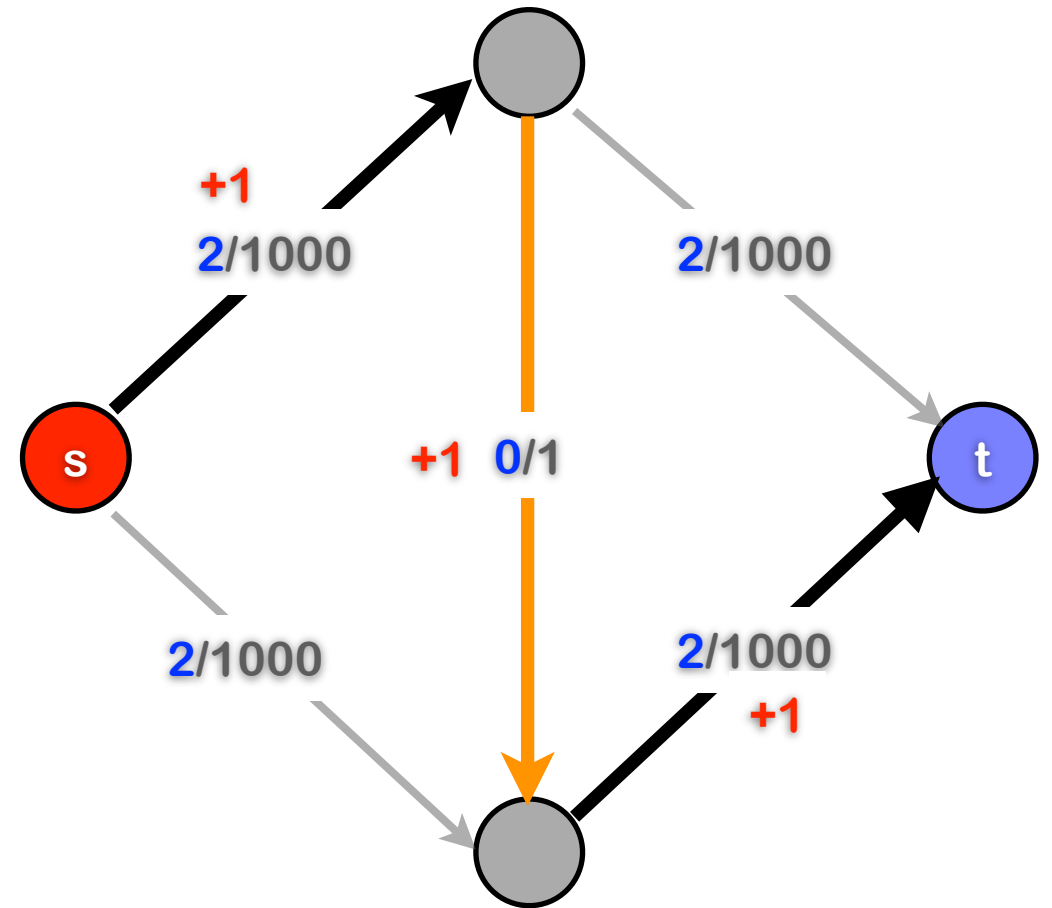
$$|f| = 3$$



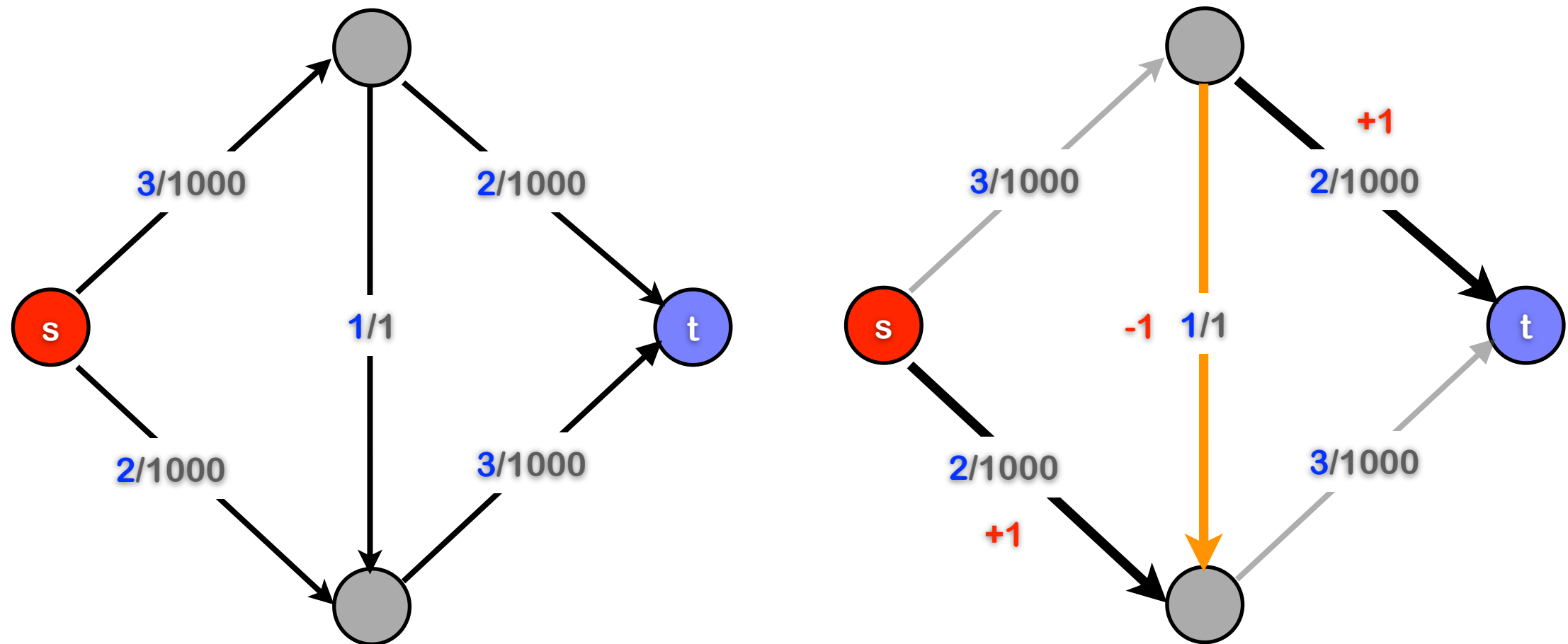
Bad News



$$|f| = 4$$



Bad News



$$|f| = 5$$

Flow value increases by exactly one at every step.

Number of steps can become VERY large.

Even more Bad News

Algorithm may not even converge to the max flow if capacities not integer.

Good News

If we either take the **shortest path** or the **fattest path**, then this will not happen if the capacities are integers. Without proof

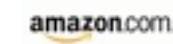
Augmenting path	Number of iterations	Implementation
BFS shortest path	$\leq \frac{1}{2} E \cdot V $	Queue
Fattest path	$\leq E \log(E \cdot U)$	Priority queue

U is the maximum flow value

★ Fattest path implementation: choose augmenting path with largest bottleneck value

Application: Bipartite Matching

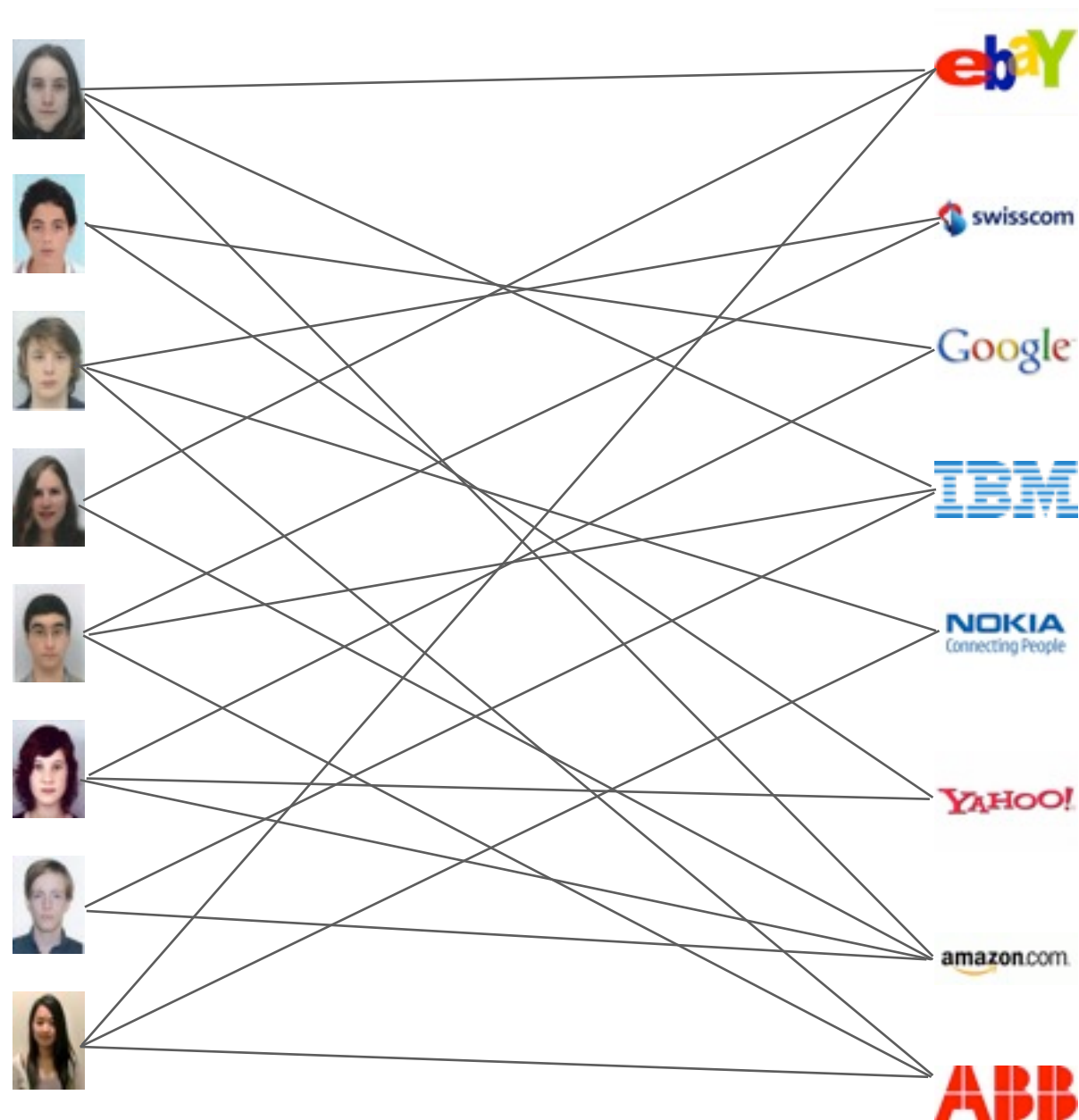
N students apply for M jobs



Bipartite Matching

N students apply for M jobs

Each get several offers

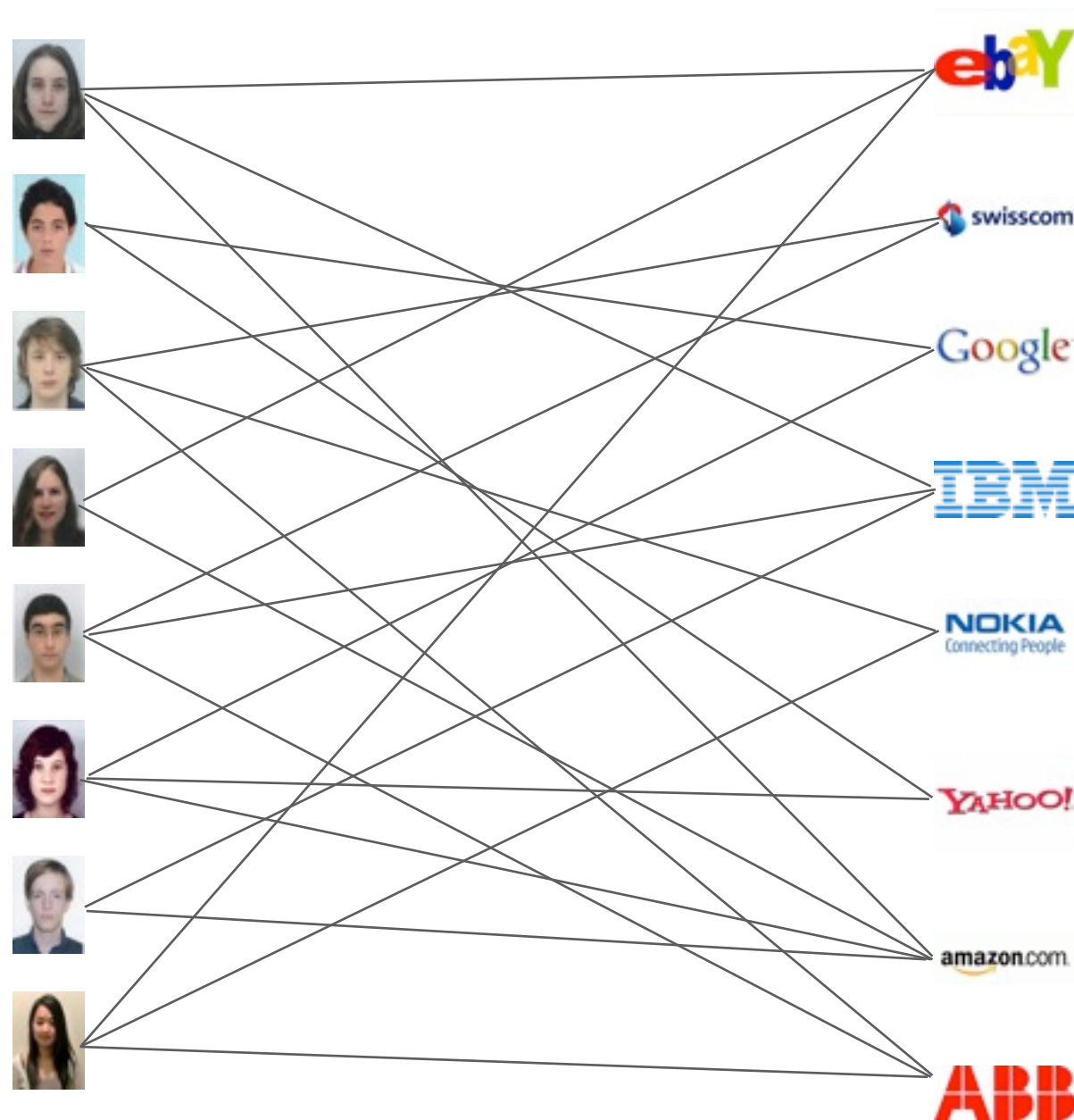


Bipartite Matching

N students apply for M jobs

Each get several offers

Is there a way to match all students to jobs? Obviously, M has to be at least equal to N .

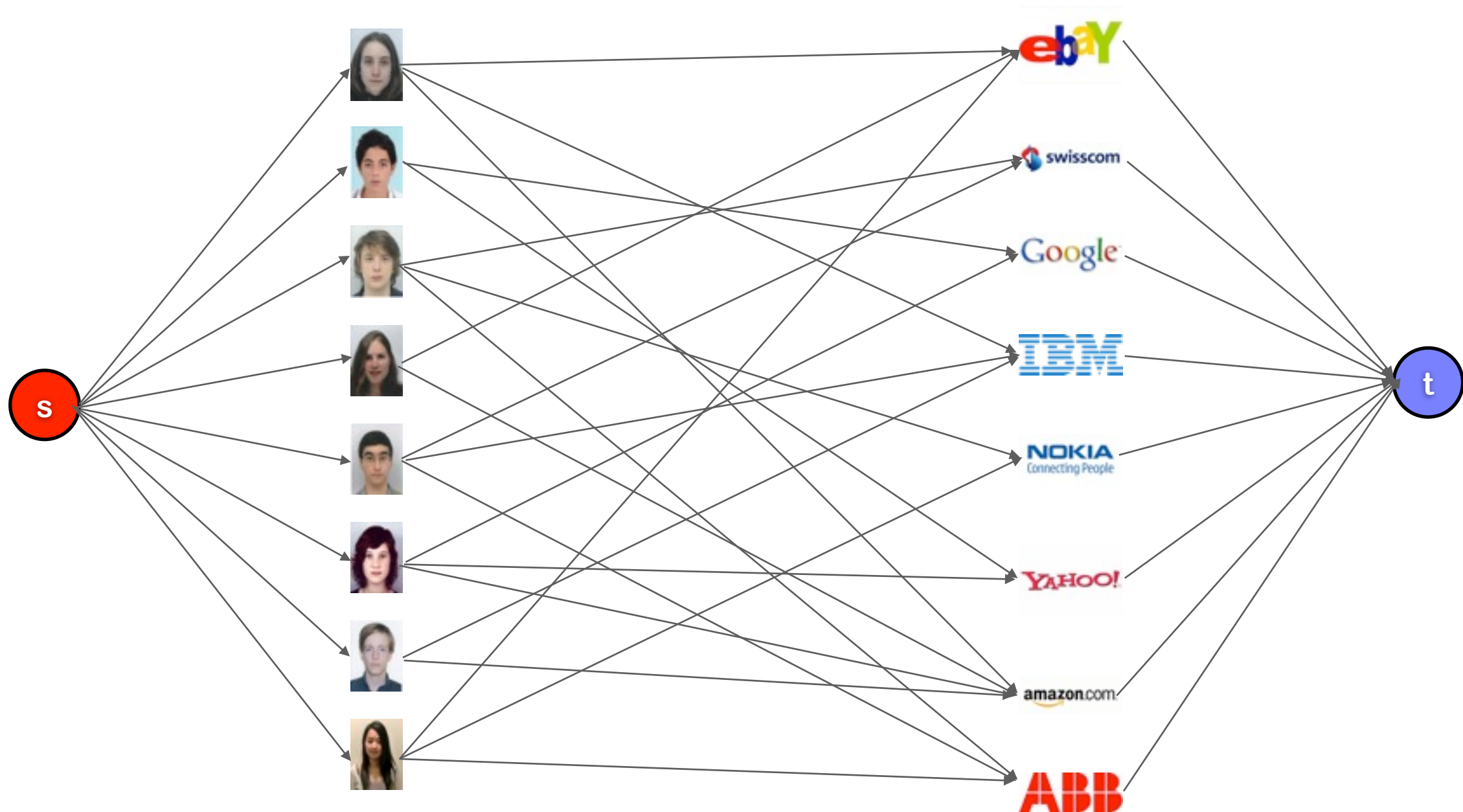


Bipartite Matching

Add node s to the left, node t to the right with edges going from s to students and edges going from jobs to t .

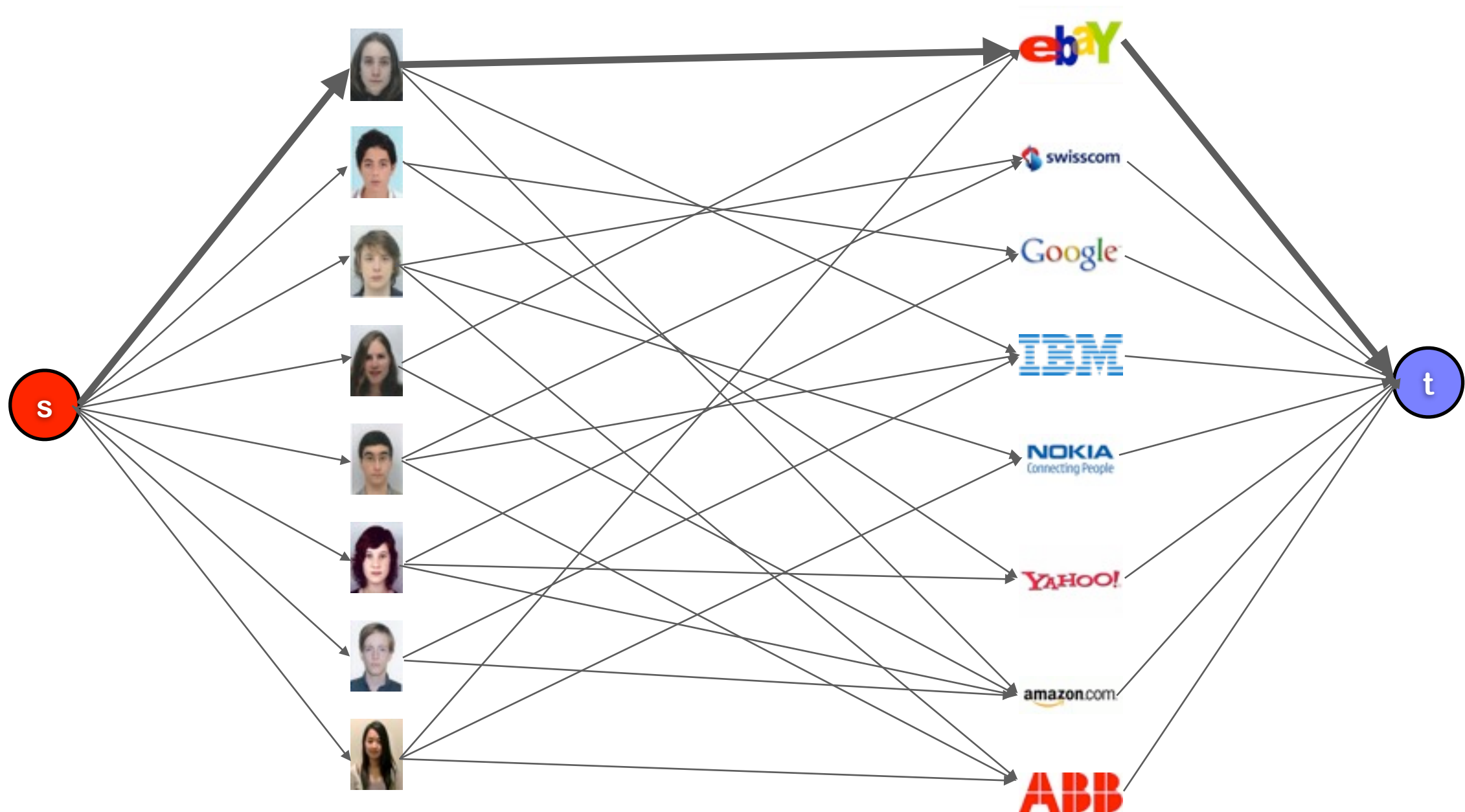
All edges have capacity one.

Direction is from left to right.

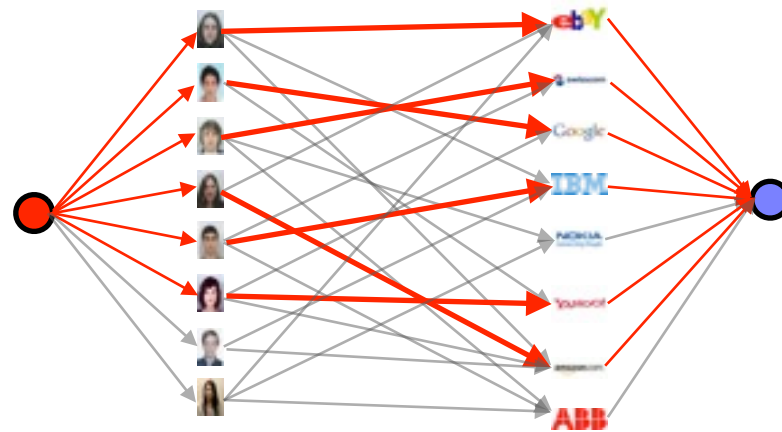
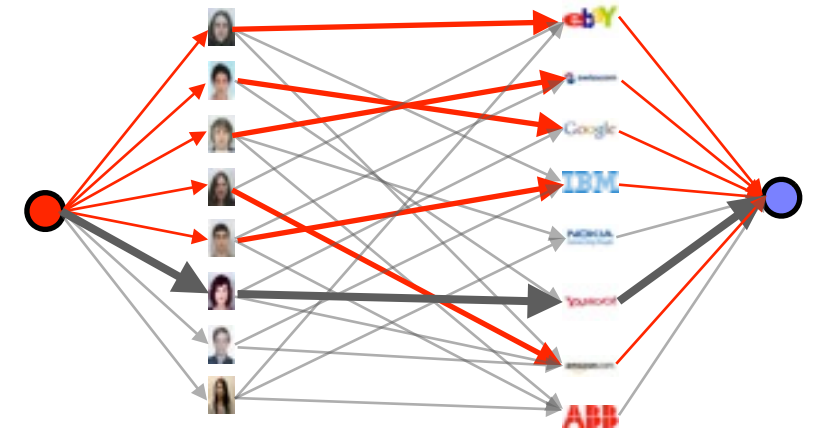
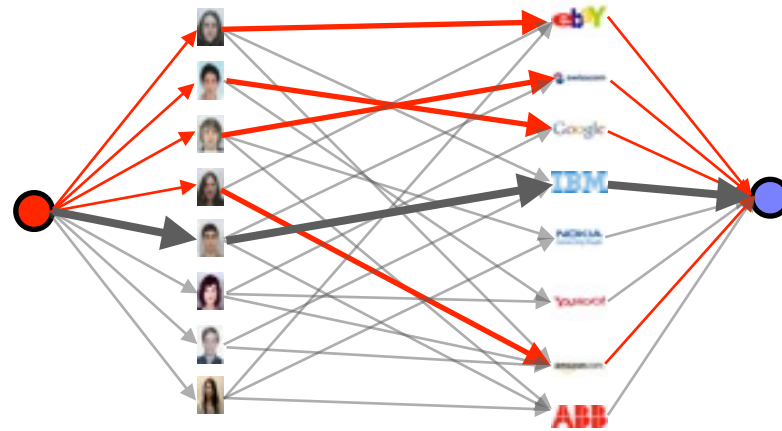
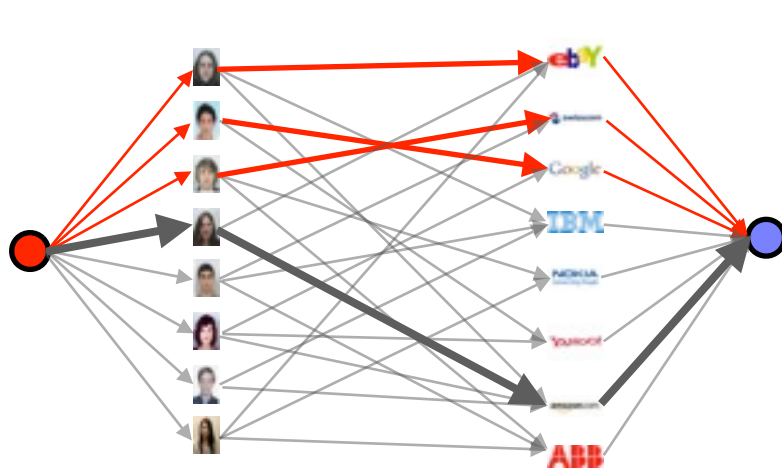
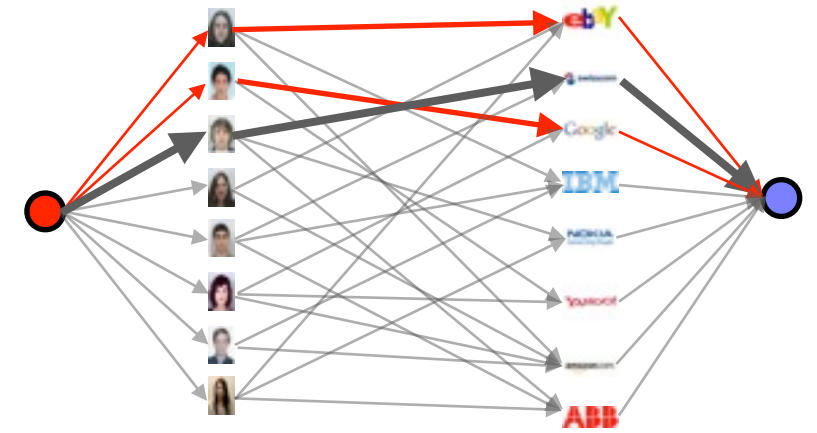
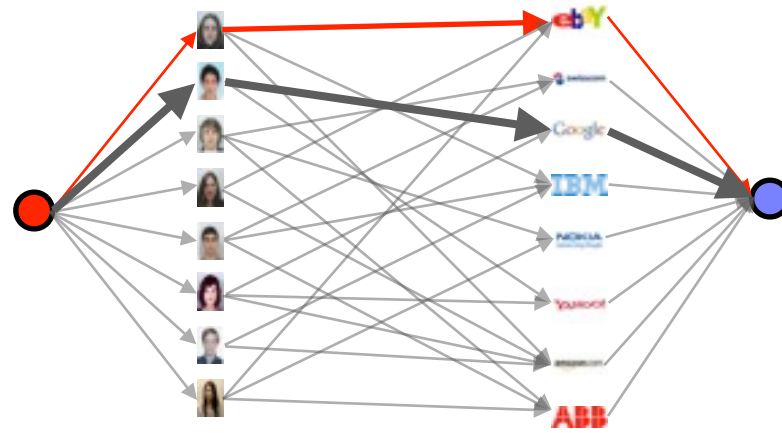
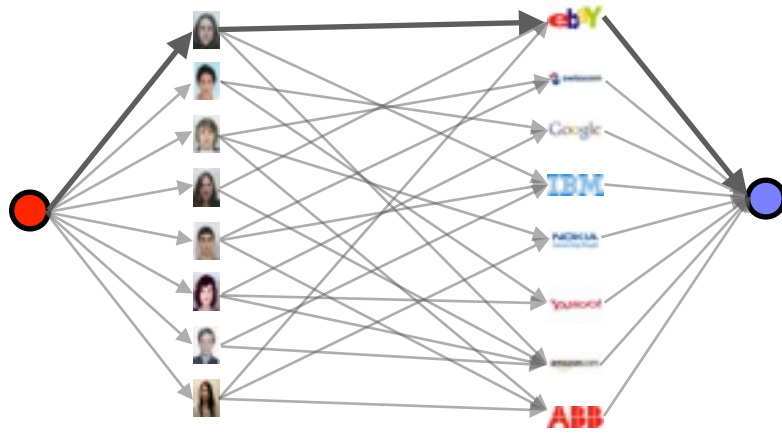


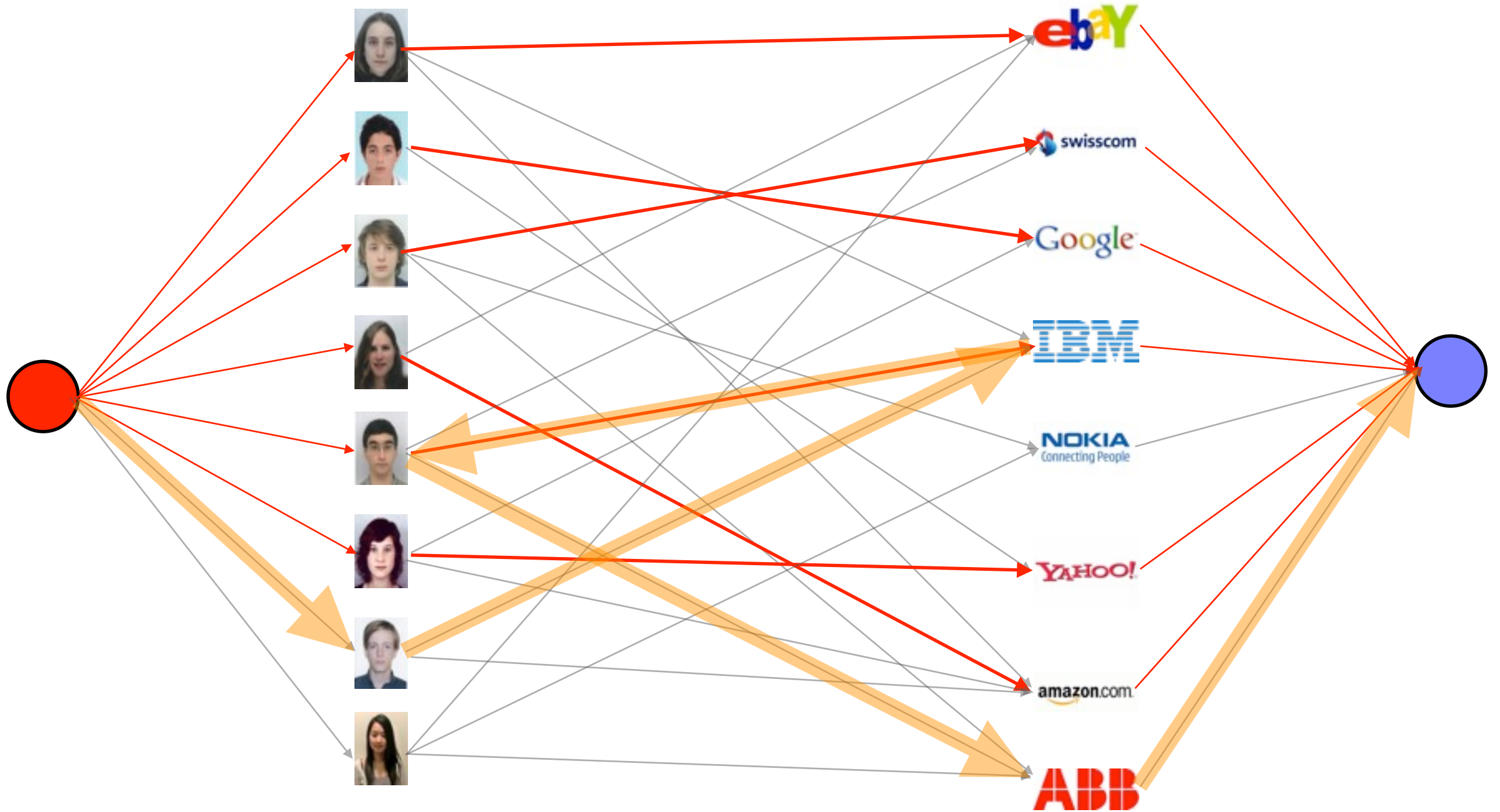
Bipartite Matching

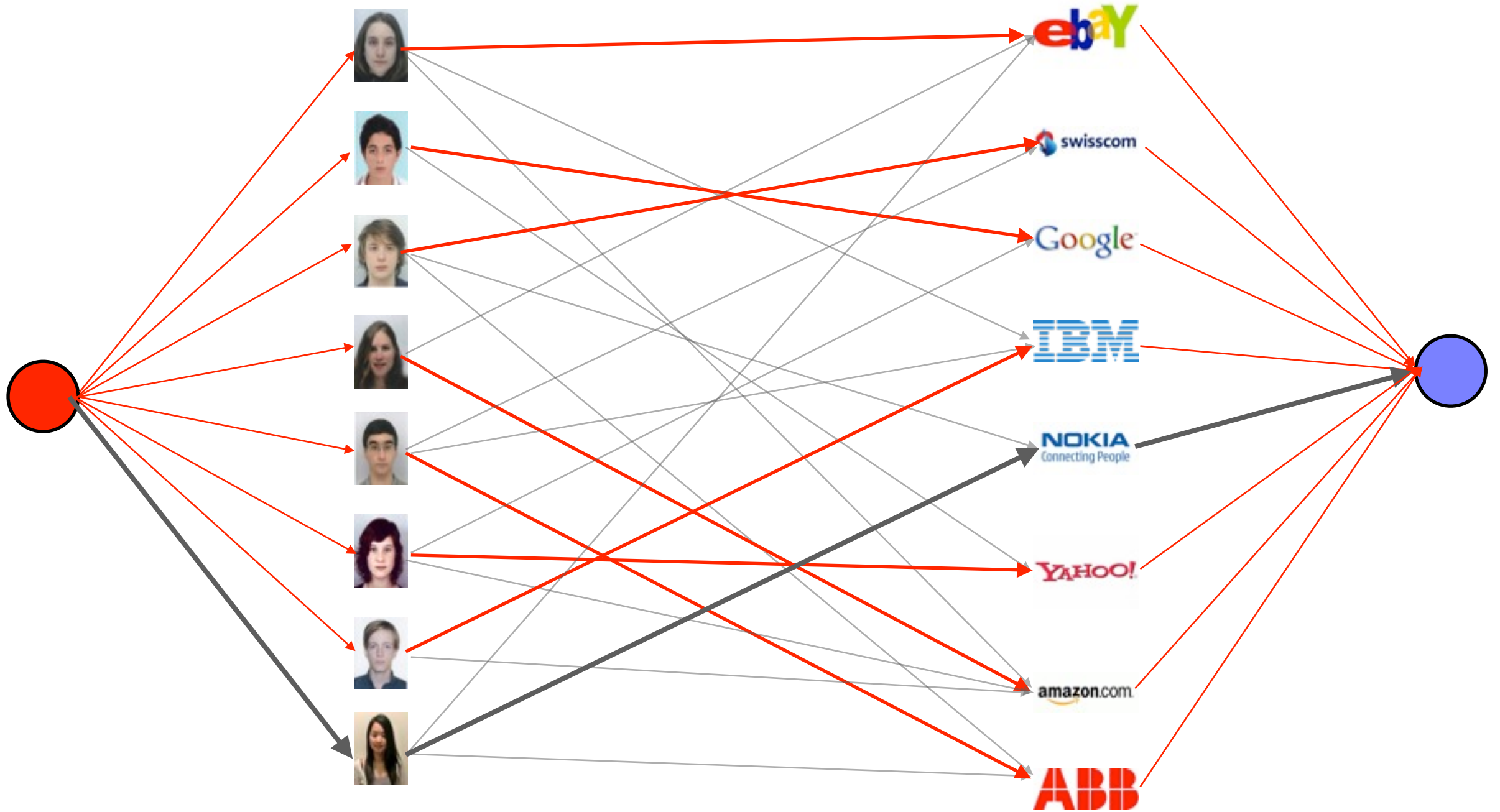
Run the Ford-Fulkerson algorithm

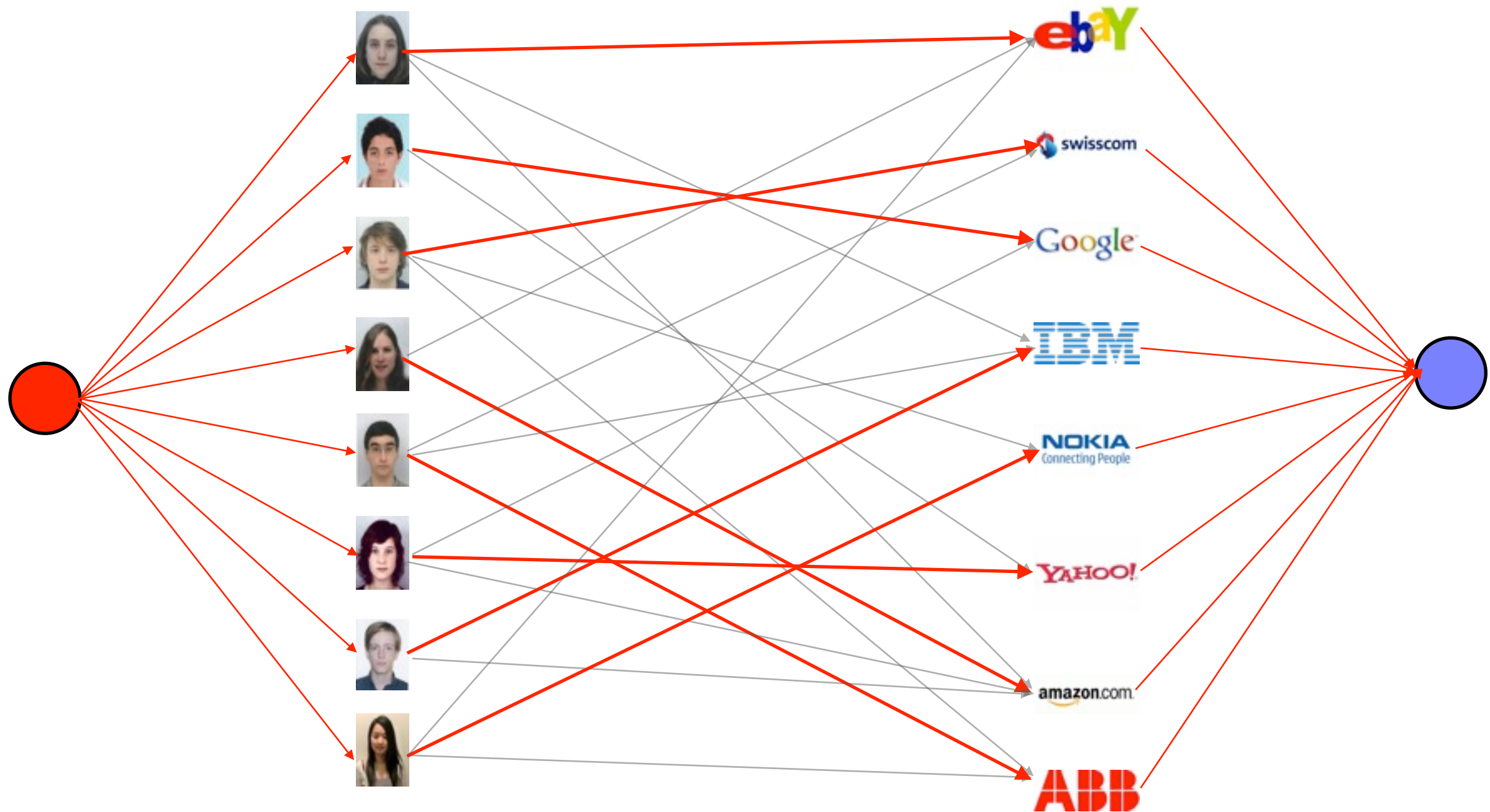


Red edges have flow 1, others have flow 0

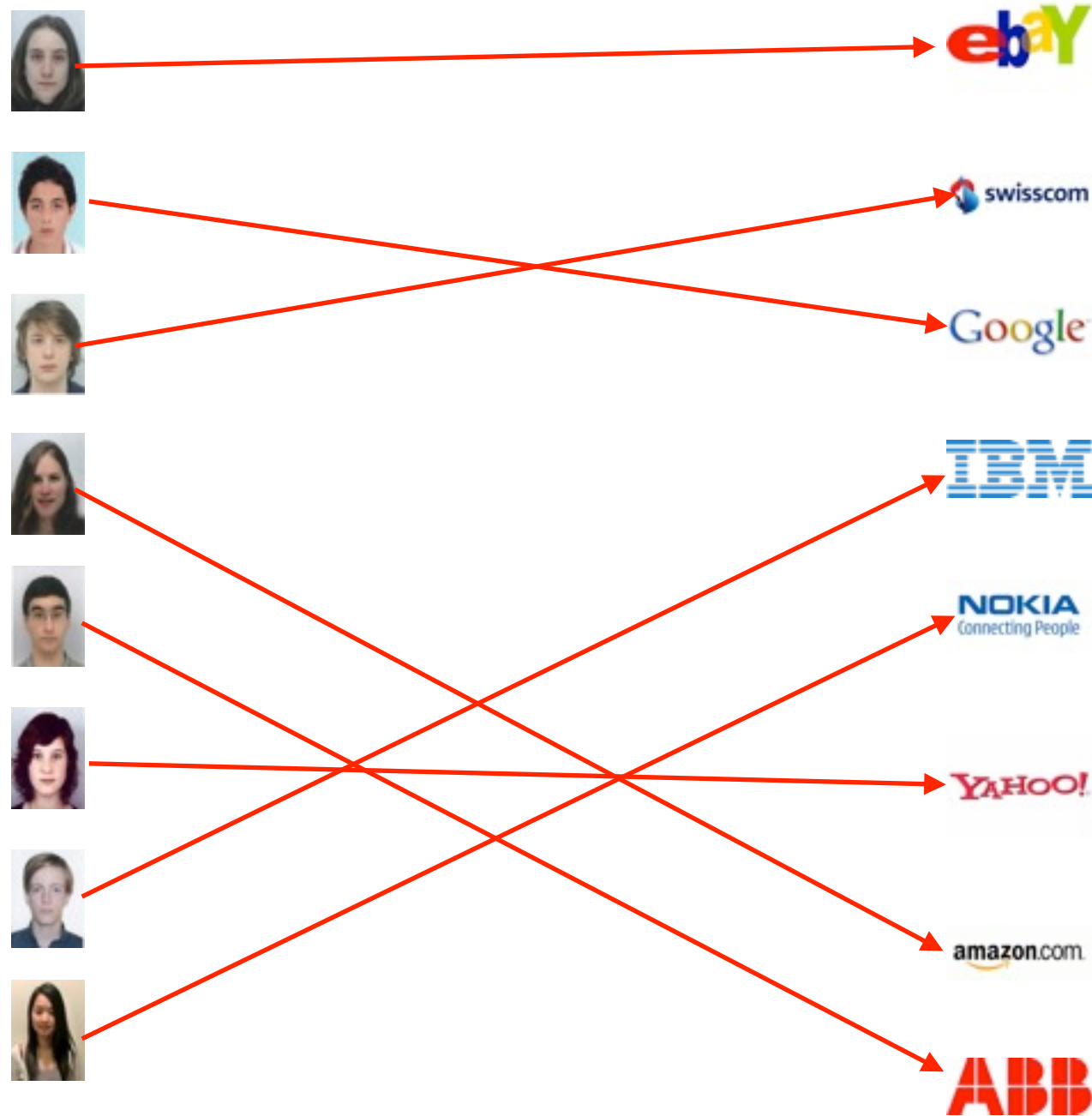








Matching is complete.



Matching is complete.

Why does it Work?

Every matching is a flow:

Put flow 1 on

- The edges of the matching
- Edges from s to matched student nodes
- Edges from matched job nodes to t

Put flow 0 on all other edges.

Works because flow conservation is equivalent to: no student is matched more than once, no job is matched more than once.

Why does it Work?

Every flow during the algorithm is a matching:

Flows are integer valued, so value on every edge is 0 or 1.

Edges between students and jobs with flow 1 are a matching, by flow conservation:

- There cannot be more than one edge with flow 1 from a student node
- There cannot be more than one edge with flow 1 into a job node

So, maximum flow is maximum matching!