

Network Flow II

Inge Li Gørtz

KT 7.3, 7.5, 7.6

Network Flow

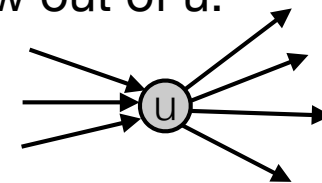
- Network flow:

- graph $G=(V,E)$.
- Special vertices s (source) and t (sink).
- Every edge e has a capacity $c(e) \geq 0$.

- Flow:

- **capacity constraint:** every edge e has a flow $0 \leq f(e) \leq c(e)$.
- **flow conservation:** for all $u \neq s, t$: flow into u equals flow out of u .

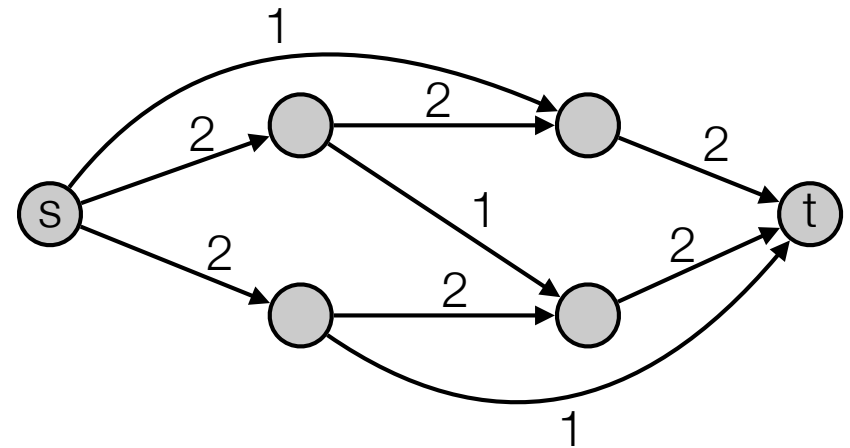
$$\sum_{v:(v,u) \in E} f(v,u) = \sum_{v:(u,v) \in E} f(u,v)$$



- Value of flow f is the sum of flows out of s minus sum of flows into s :

$$v(f) = \sum_{v:(s,v) \in E} f(e) - \sum_{v:(v,s) \in E} f(e) = f^{out}(s) - f^{in}(s)$$

- **Maximum flow problem:** find s - t flow of maximum value

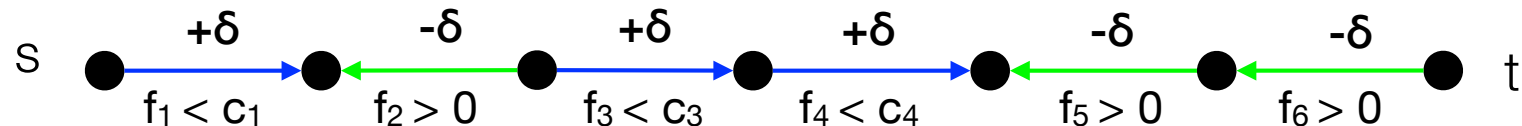


Today

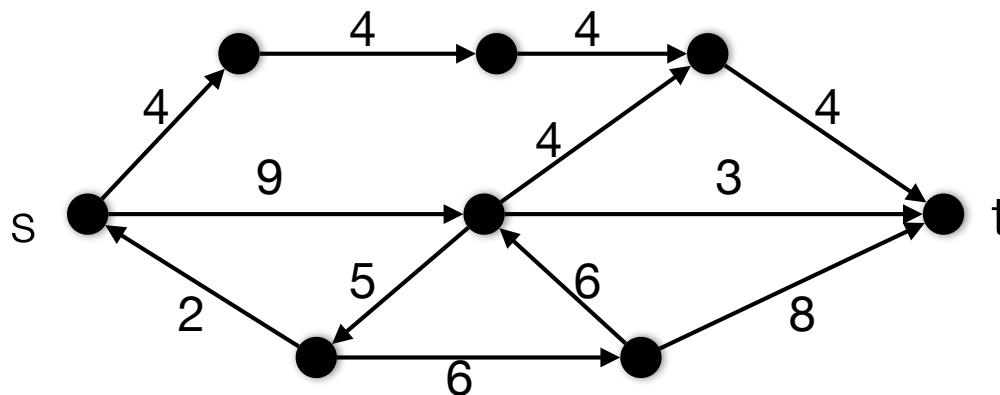
- Applications
- Finding good augmenting paths. Edmonds-Karp and scaling algorithm.

Ford-Fulkerson

- Find (any) augmenting path and use it.
- Augmenting path (definition different than in CLRS): s-t path where
 - **forward** edges have leftover capacity
 - **backwards** edges have positive flow

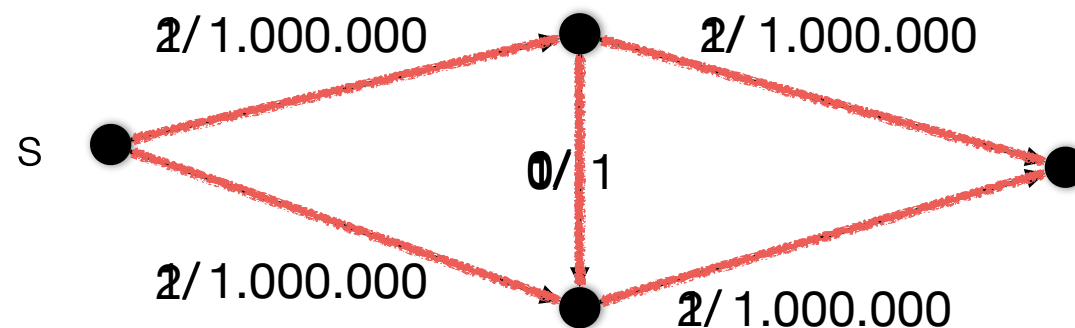


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$
- To find augmenting path use DFS or BFS:



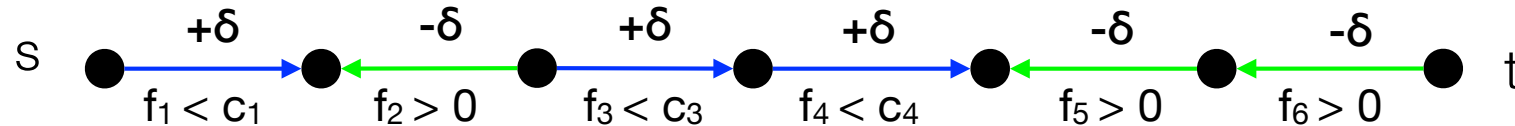
Ford-Fulkerson

- Integral capacities:
 - Each augmenting path increases flow with at least 1.
 - At most $v(f)$ iterations
 - Find augmenting path via DFS/BFS: $O(m)$
 - Total running time: $O(m \cdot v(f))$
- **Lemma.** If all the capacities are integers, then there is a maximum flow where the flow on every edge is an integer.
- Bad example for Ford-Fulkerson:

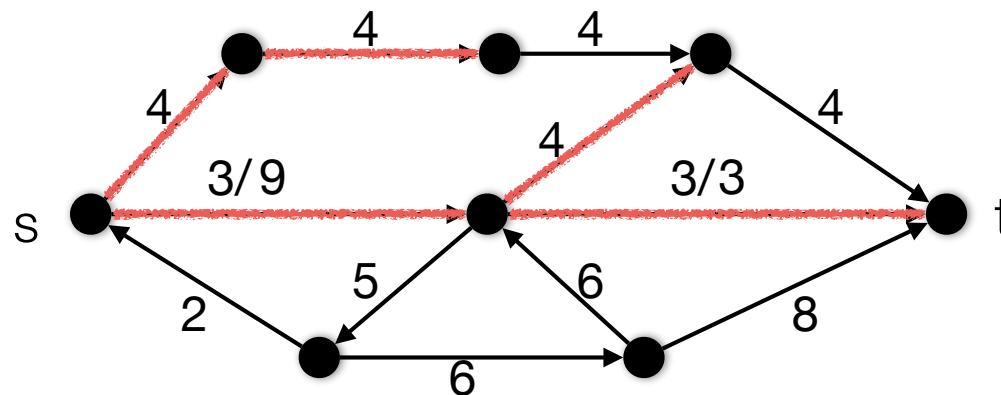


Edmonds-Karp

- Find *shortest augmenting path* and use it.
- Augmenting path (definition different than in CLRS): s-t path where
 - forward* edges have leftover capacity
 - backwards* edges have positive flow

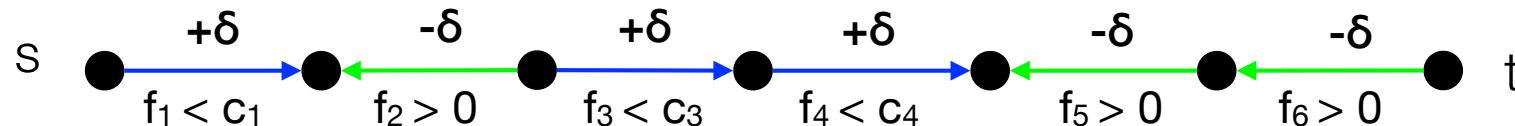


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$
- To find augmenting path use **BFS**:

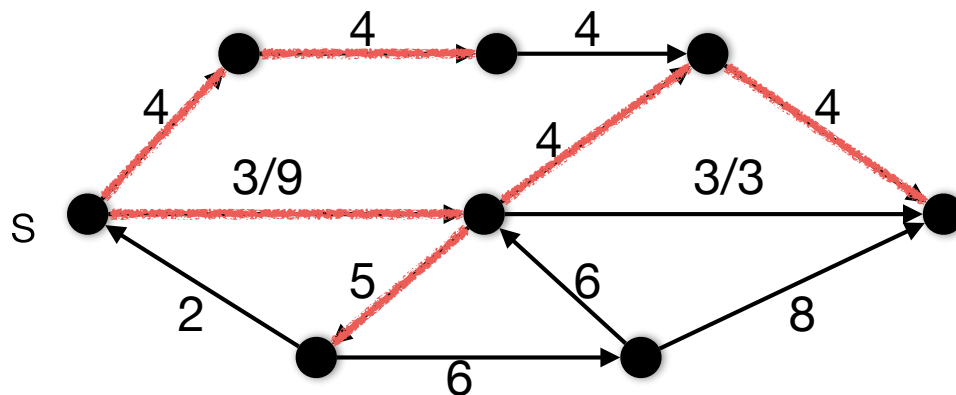


Edmonds-Karp

- Find *shortest augmenting path* and use it.
- Augmenting path (definition different than in CLRS): s-t path where
 - forward* edges have leftover capacity
 - backwards* edges have positive flow

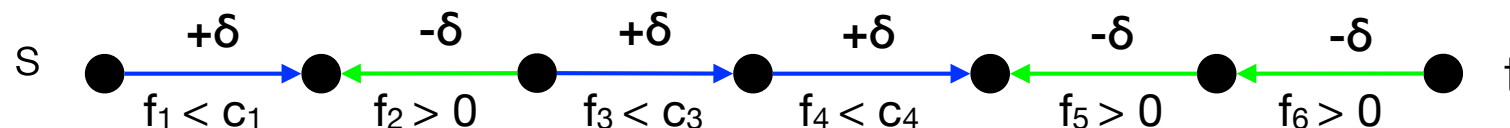


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$
- To find augmenting path use *BFS*:

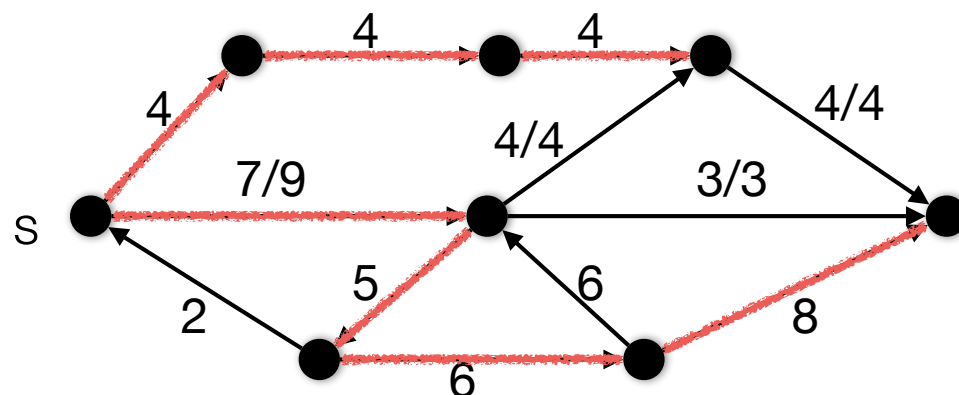


Edmonds-Karp

- Find *shortest augmenting path* and use it.
- Augmenting path (definition different than in CLRS): s-t path where
 - forward* edges have leftover capacity
 - backwards* edges have positive flow

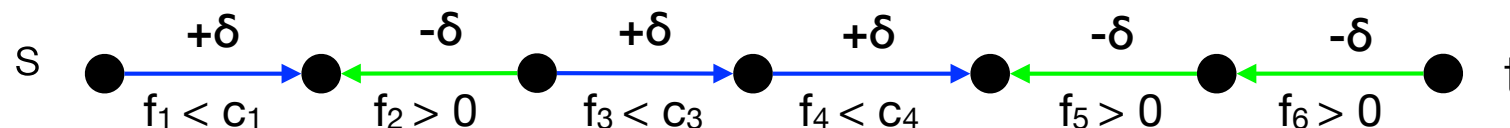


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$
- To find augmenting path use *BFS*:

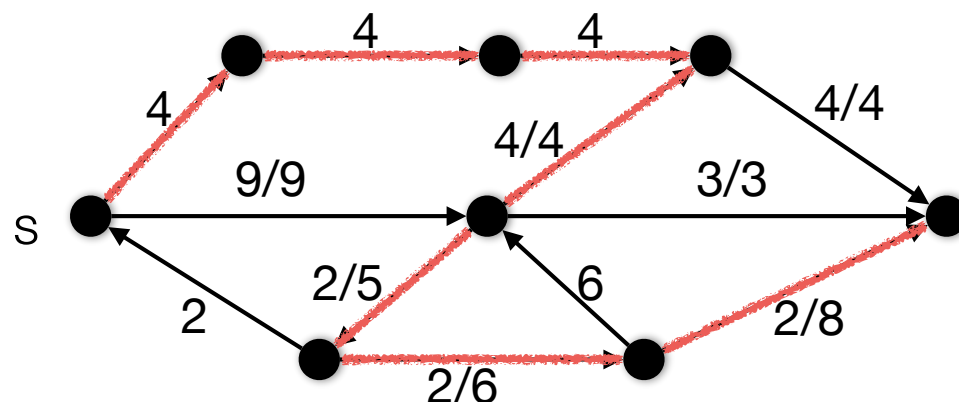


Edmonds-Karp

- Find *shortest augmenting path* and use it.
- Augmenting path (definition different than in CLRS): s-t path where
 - forward* edges have leftover capacity
 - backwards* edges have positive flow

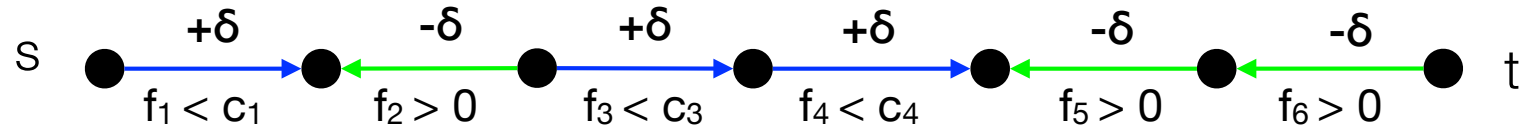


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$
- To find augmenting path use *BFS*:

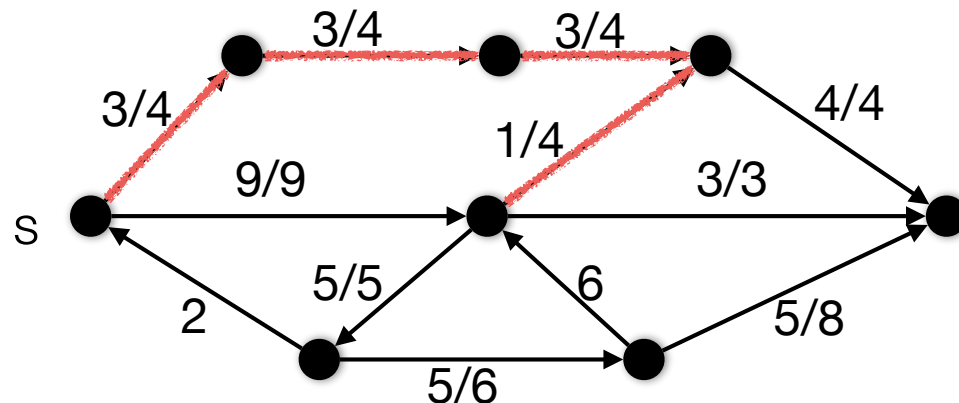


Edmonds-Karp

- Find *shortest augmenting path* and use it.
- Augmenting path (definition different than in CLRS): s-t path where
 - forward* edges have leftover capacity
 - backwards* edges have positive flow

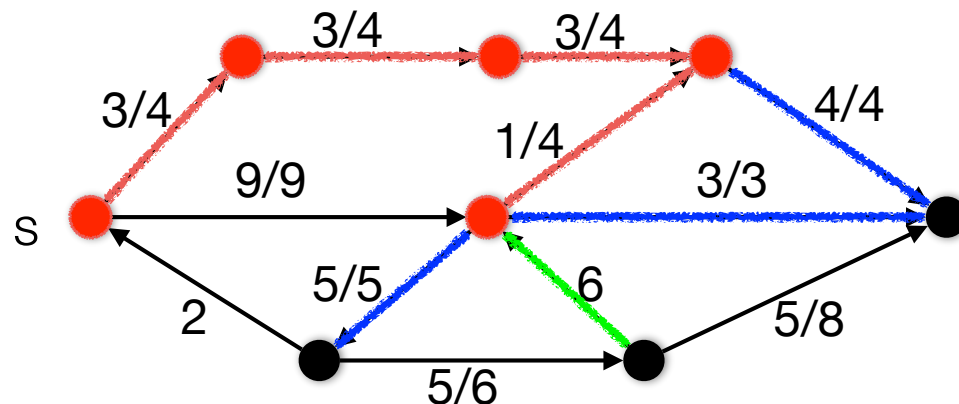


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$
- To find augmenting path use *BFS*:



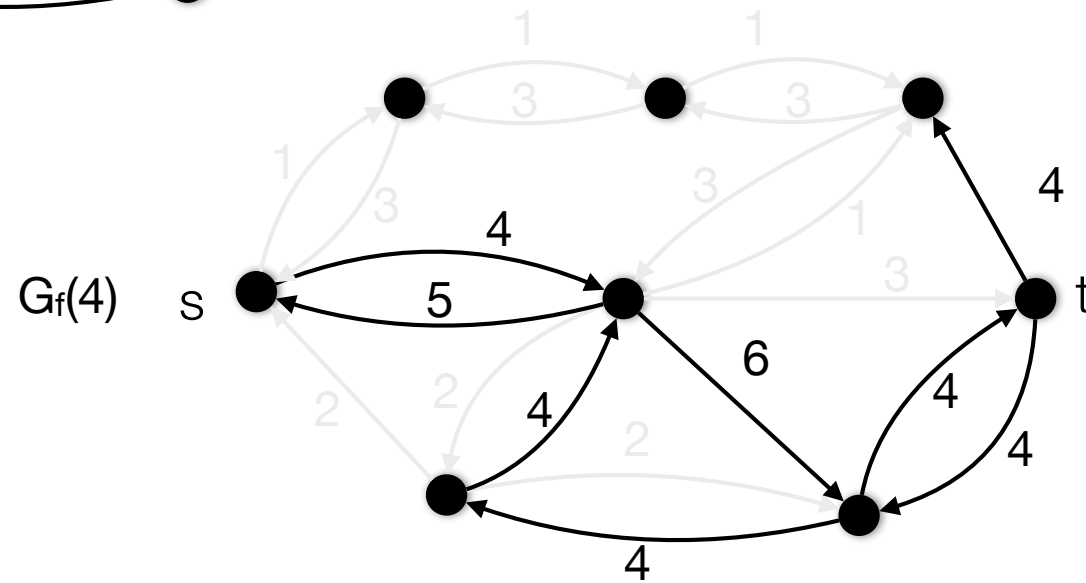
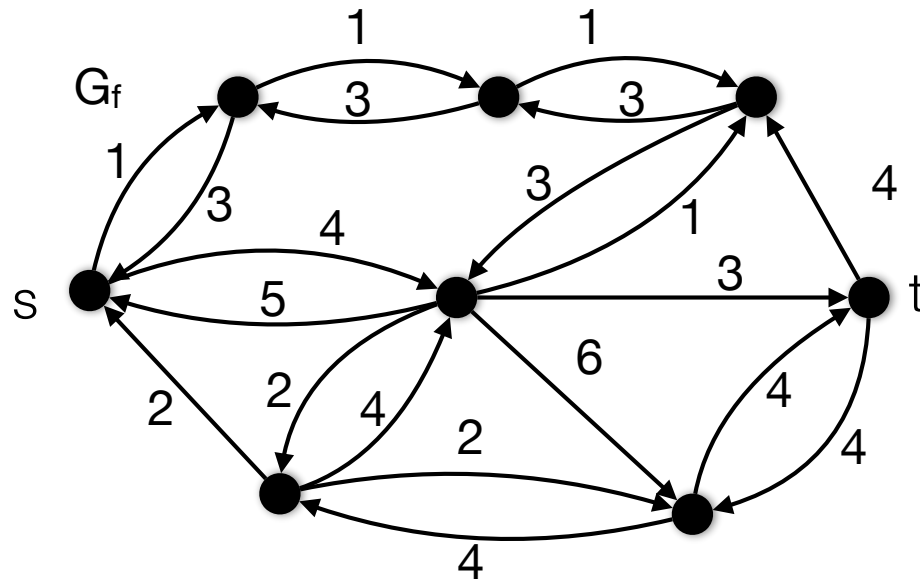
Find a minimum cut

- When there are no more augmenting s-t paths:
- Find all augmenting paths from s.
- The nodes S that can be reached by these augmenting paths form the left side of a minimum cut.
 - **edges out** of S have $c_e = f_e$.
 - **edges into** S have $f_e = 0$.
 - Capacity of the cut equals the flow.



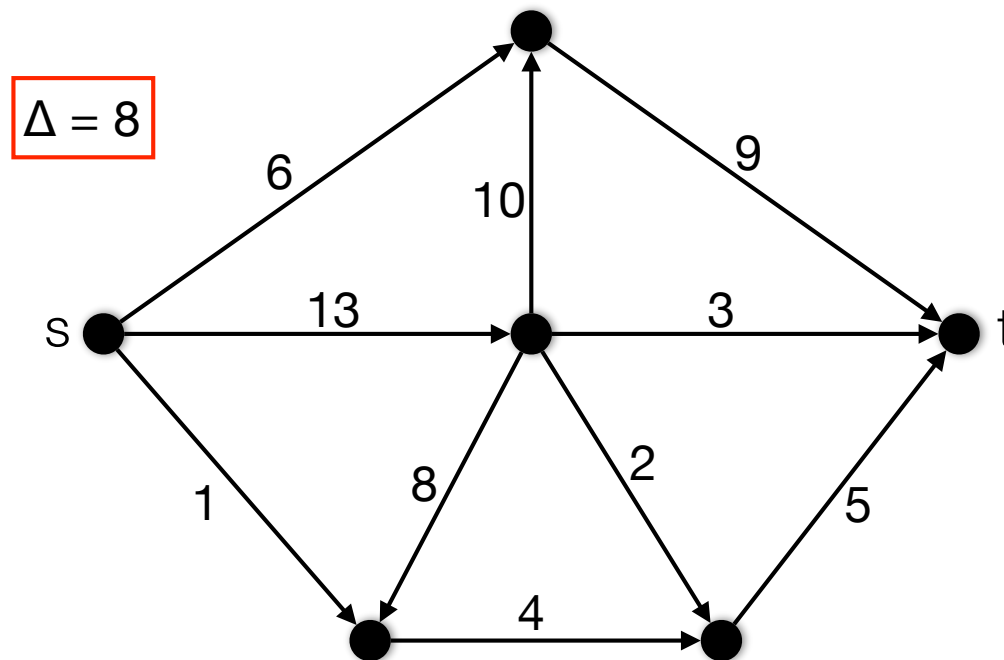
Scaling algorithm

- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_f(\Delta)$.
- Example: $\Delta = 4$



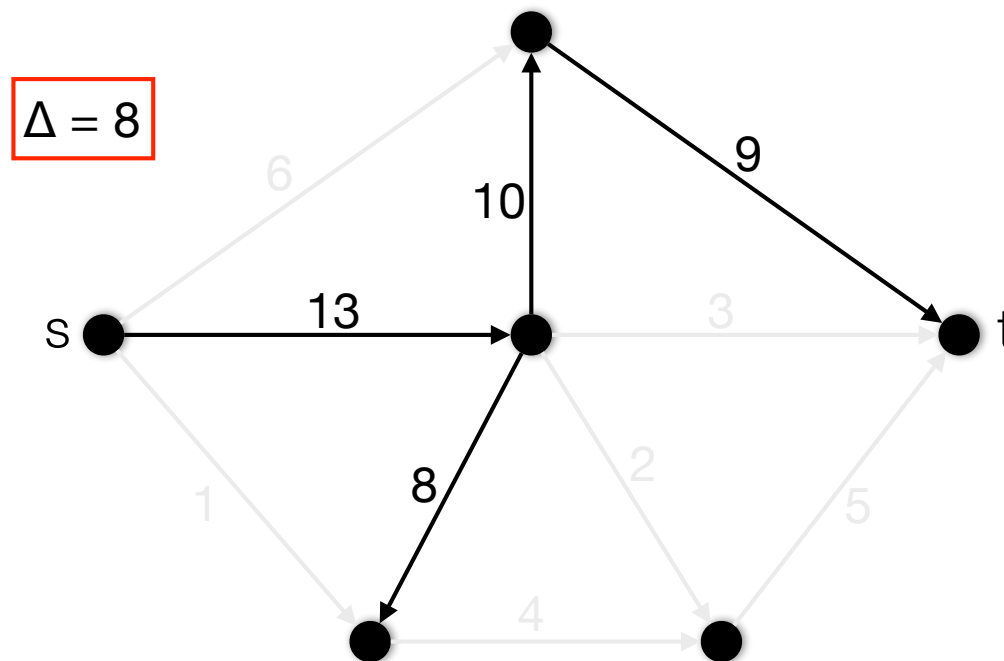
Scaling algorithm

- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_f(\Delta)$.
- Start with $\Delta = \text{“highest power of 2 } \leq \text{largest capacity out of } s\text{”}$



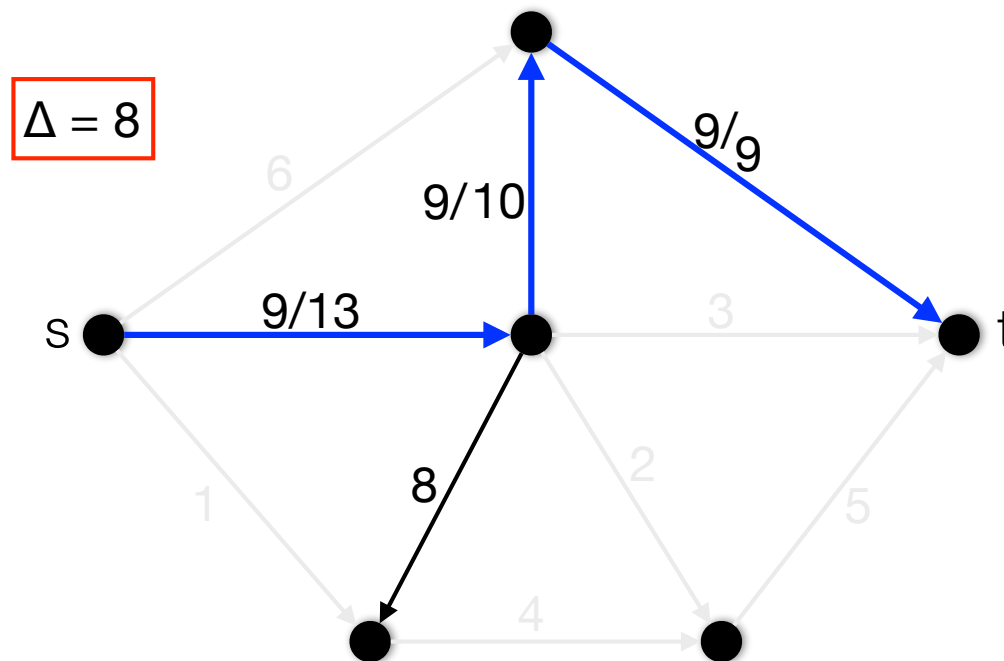
Scaling algorithm

- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_f(\Delta)$.
- Start with $\Delta = \text{"highest power of 2 } \leq \text{largest capacity out of } s\text{"}$



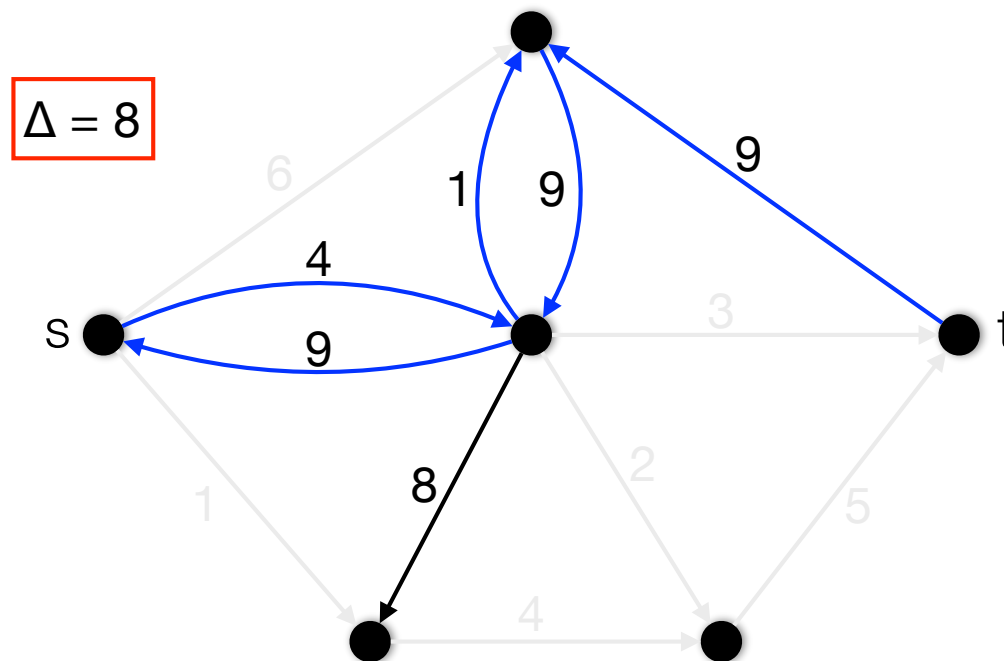
Scaling algorithm

- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_f(\Delta)$.
- Start with $\Delta = \text{"highest power of 2 } \leq \text{largest capacity out of } s\text{"}$



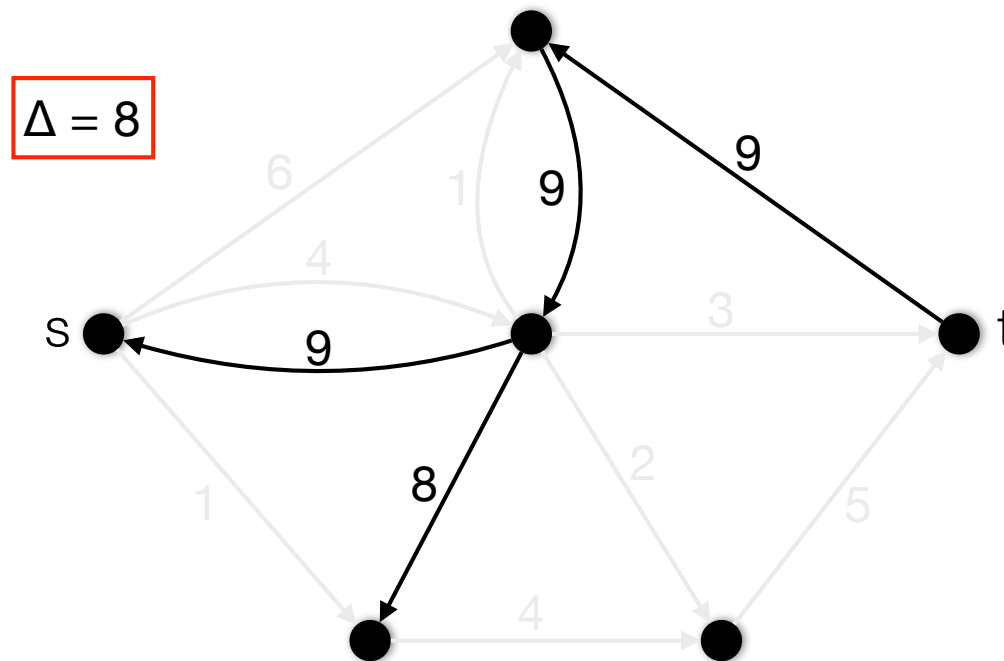
Scaling algorithm

- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_f(\Delta)$.
- Start with $\Delta = \text{"highest power of 2 } \leq \text{largest capacity out of } s\text{"}$



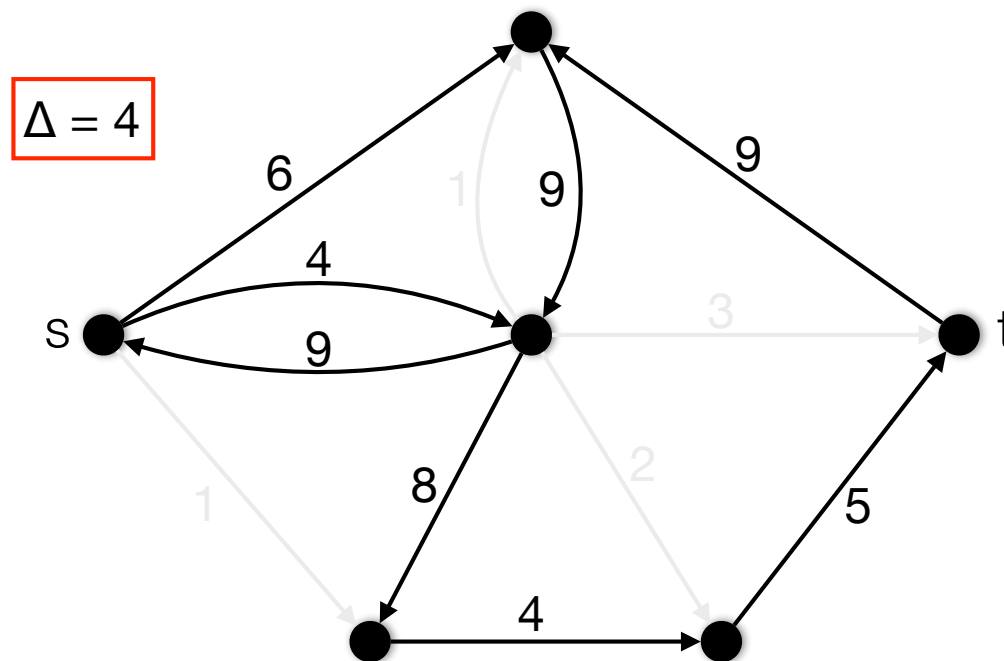
Scaling algorithm

- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_f(\Delta)$.
- Start with $\Delta = \text{“highest power of 2 } \leq \text{largest capacity out of } s\text{”}$
- When no more augmenting paths in $G_f(\Delta)$: $\Delta = \Delta/2$ (new phase).



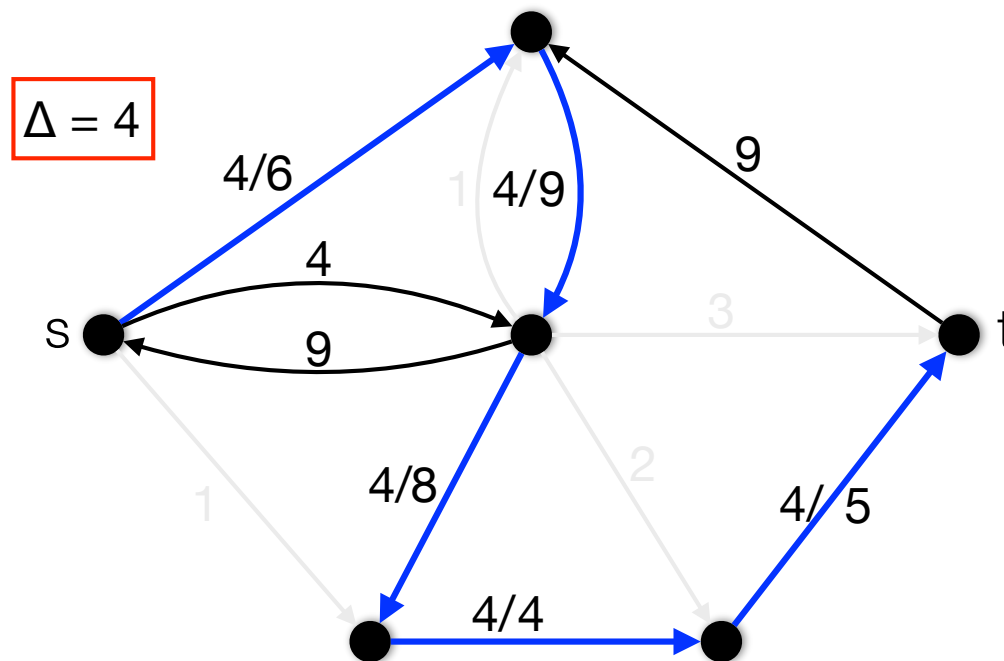
Scaling algorithm

- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_f(\Delta)$.
- Start with $\Delta = \text{"highest power of 2 } \leq \text{largest capacity out of } s\text{"}$
- When no more augmenting paths in $G_f(\Delta)$: $\Delta = \Delta/2$ (new phase).



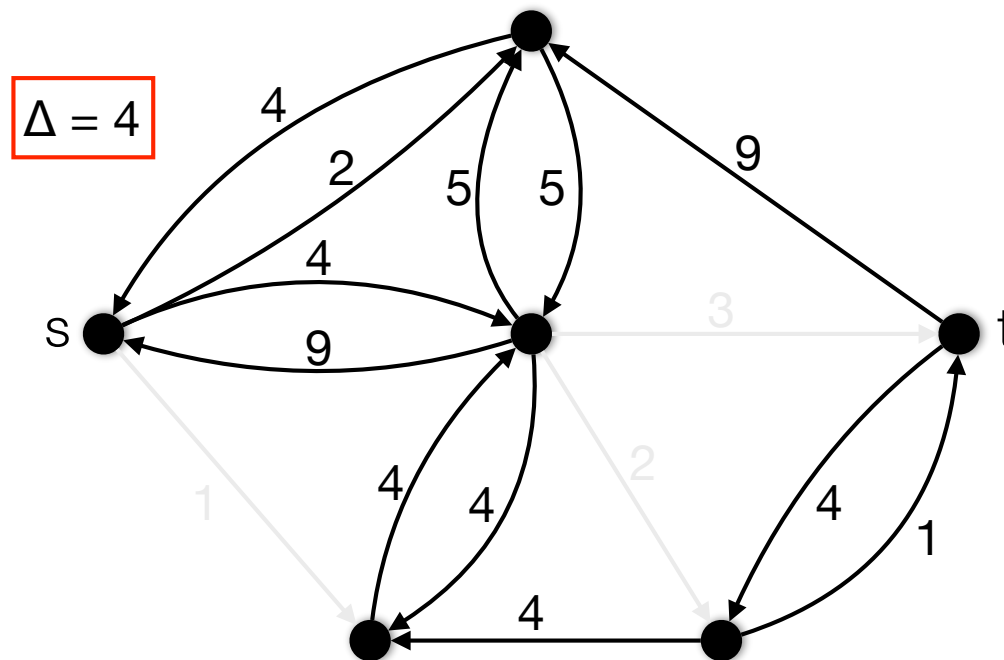
Scaling algorithm

- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_f(\Delta)$.
- Start with $\Delta =$ “highest power of 2 $\leq$ largest capacity out of s ”
- When no more augmenting paths in $G_f(\Delta)$: $\Delta = \Delta/2$ (new phase).



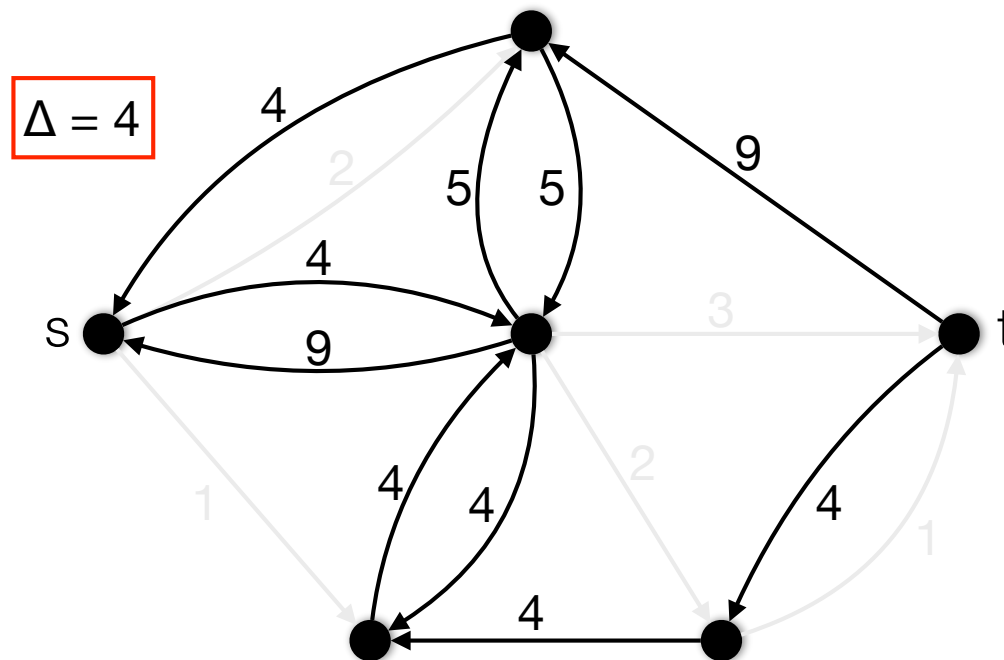
Scaling algorithm

- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_f(\Delta)$.
- Start with $\Delta = \text{"highest power of 2 } \leq \text{largest capacity out of } s\text{"}$
- When no more augmenting paths in $G_f(\Delta)$: $\Delta = \Delta/2$ (new phase).



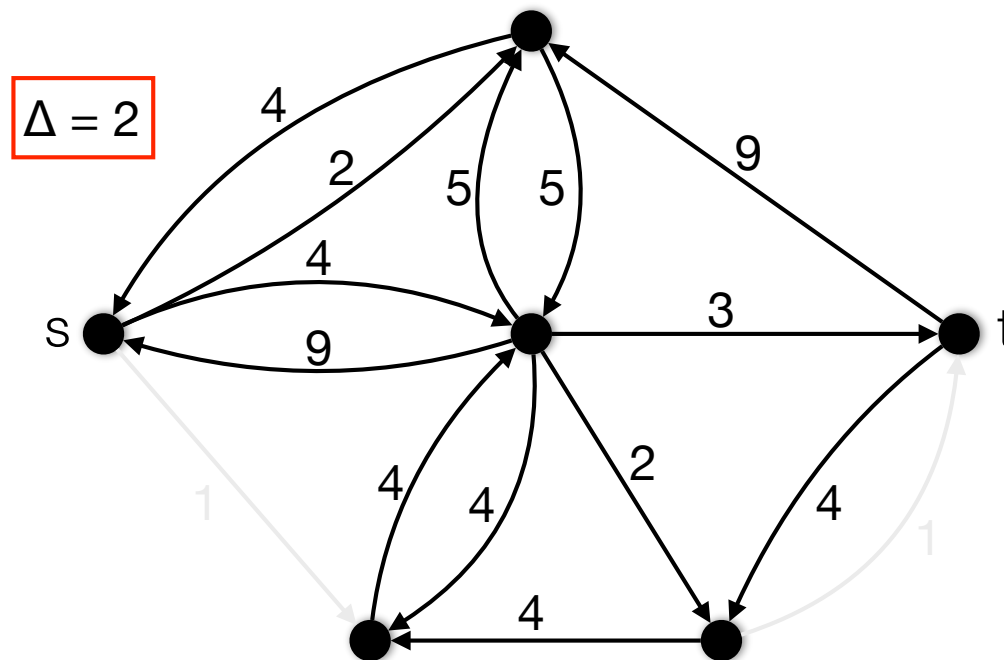
Scaling algorithm

- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_f(\Delta)$.
- Start with $\Delta = \text{“highest power of 2 } \leq \text{largest capacity out of } s\text{”}$
- When no more augmenting paths in $G_f(\Delta)$: $\Delta = \Delta/2$ (new phase).



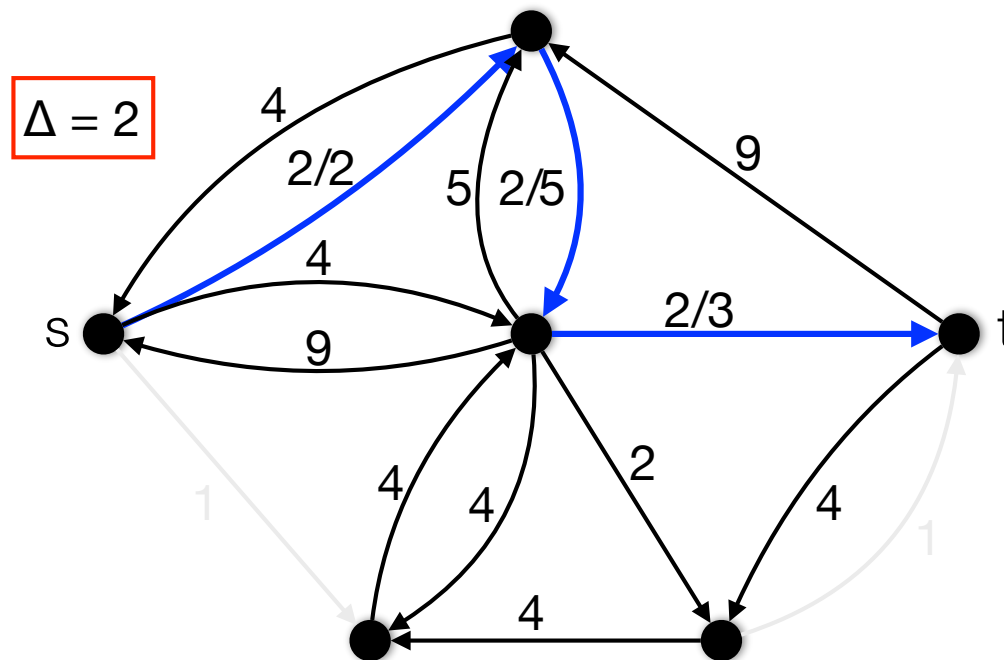
Scaling algorithm

- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_f(\Delta)$.
- Start with $\Delta = \text{“highest power of 2 } \leq \text{largest capacity out of } s\text{”}$
- When no more augmenting paths in $G_f(\Delta)$: $\Delta = \Delta/2$ (new phase).



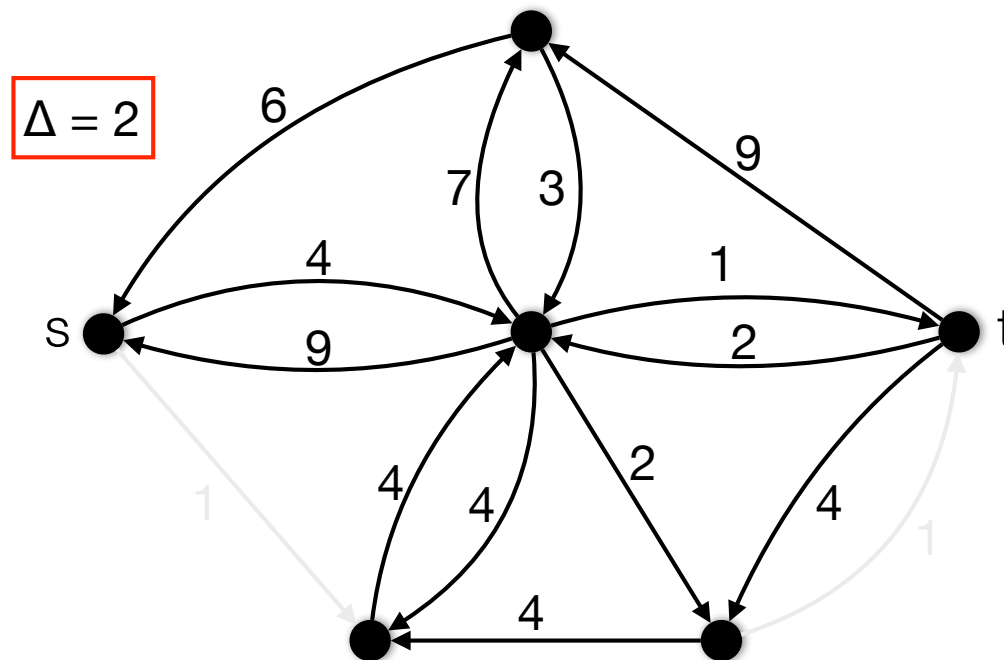
Scaling algorithm

- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_f(\Delta)$.
- Start with $\Delta = \text{"highest power of 2 } \leq \text{largest capacity out of } s\text{"}$
- When no more augmenting paths in $G_f(\Delta)$: $\Delta = \Delta/2$ (new phase).



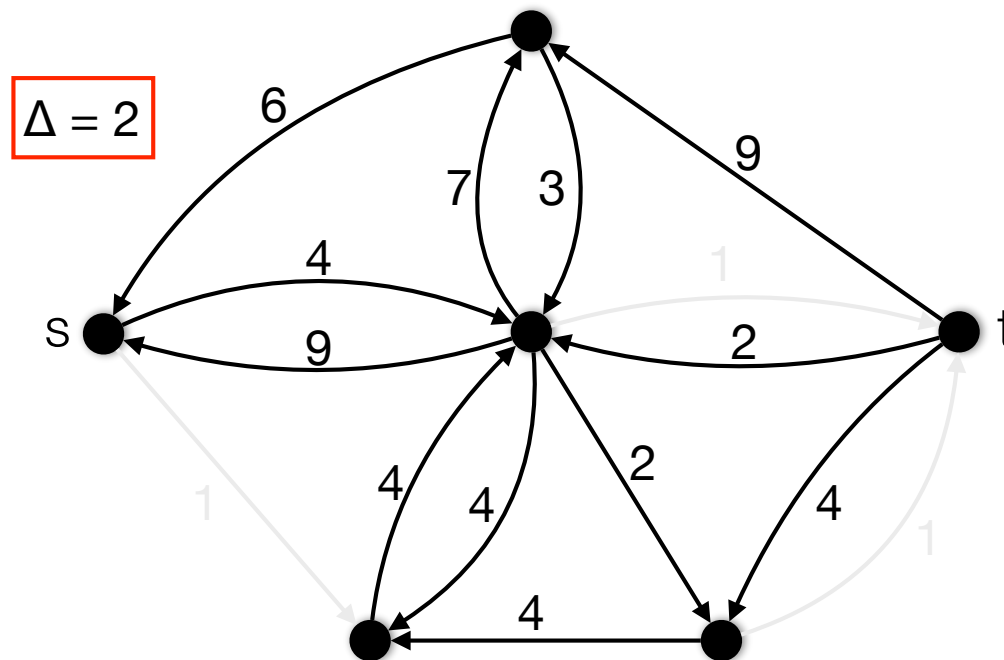
Scaling algorithm

- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_f(\Delta)$.
- Start with $\Delta = \text{“highest power of 2 } \leq \text{largest capacity out of } s\text{”}$
- When no more augmenting paths in $G_f(\Delta)$: $\Delta = \Delta/2$ (new phase).



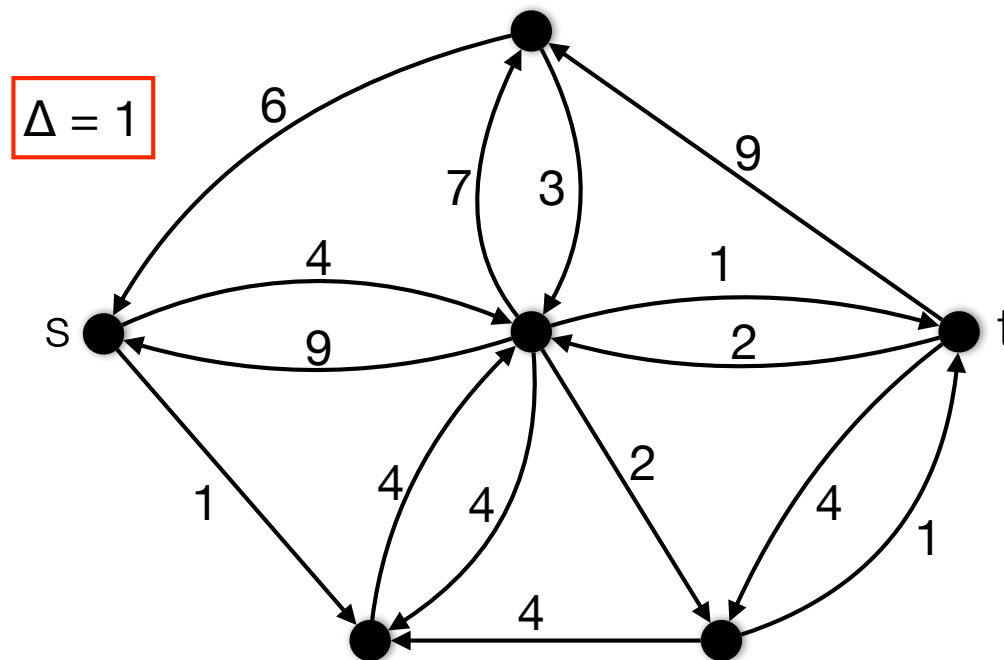
Scaling algorithm

- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_f(\Delta)$.
- Start with $\Delta = \text{“highest power of 2 } \leq \text{largest capacity out of } s\text{”}$
- When no more augmenting paths in $G_f(\Delta)$: $\Delta = \Delta/2$ (new phase).



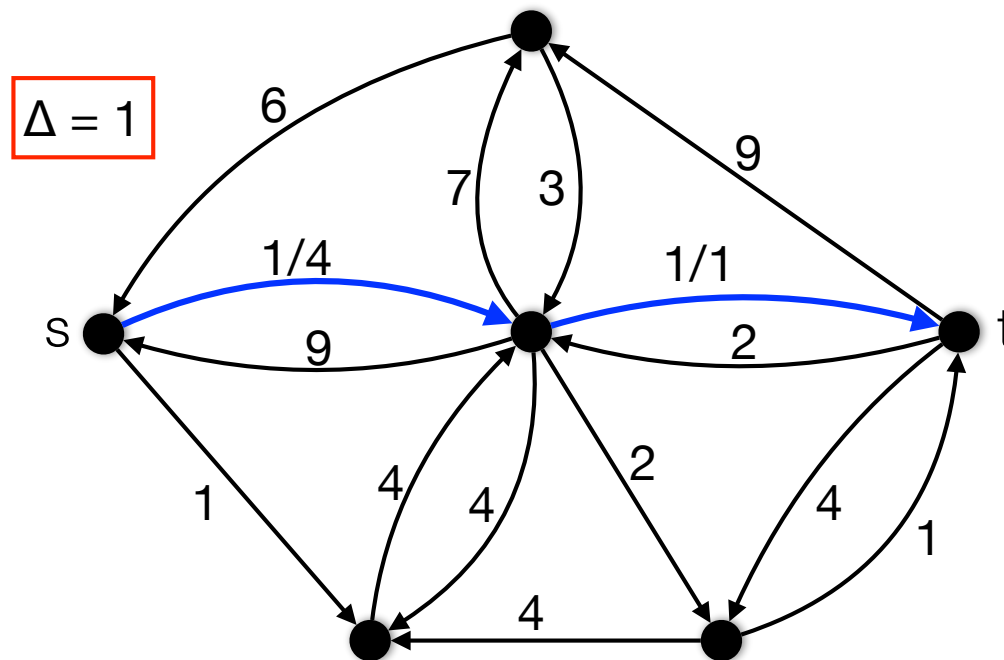
Scaling algorithm

- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_f(\Delta)$.
- Start with $\Delta = \text{“highest power of 2 } \leq \text{largest capacity out of } s\text{”}$
- When no more augmenting paths in $G_f(\Delta)$: $\Delta = \Delta/2$ (new phase).



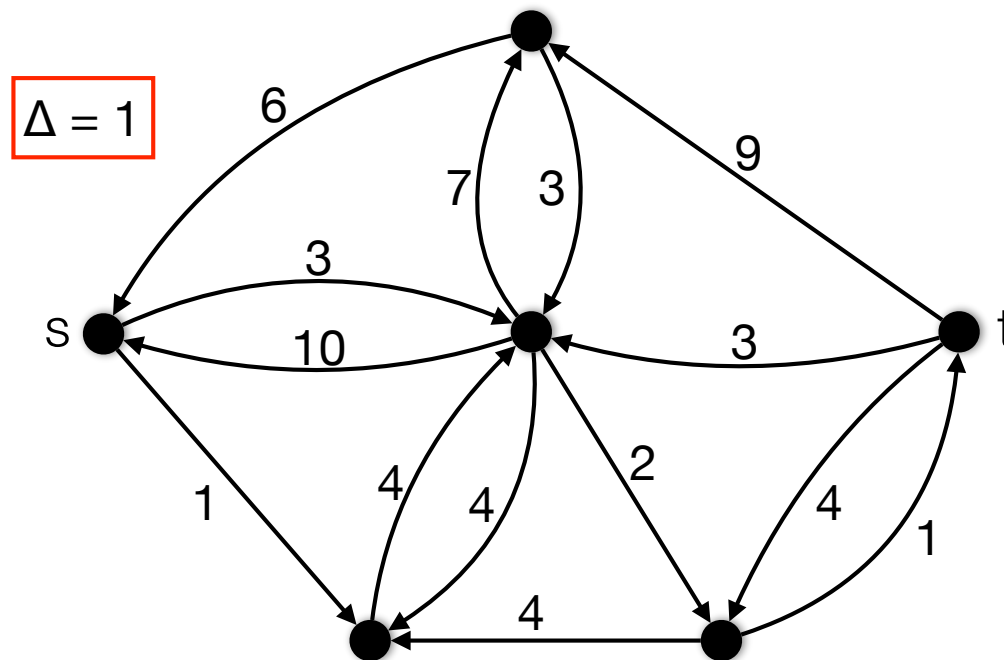
Scaling algorithm

- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_f(\Delta)$.
- Start with $\Delta = \text{“highest power of 2 } \leq \text{largest capacity out of } s\text{”}$
- When no more augmenting paths in $G_f(\Delta)$: $\Delta = \Delta/2$ (new phase).



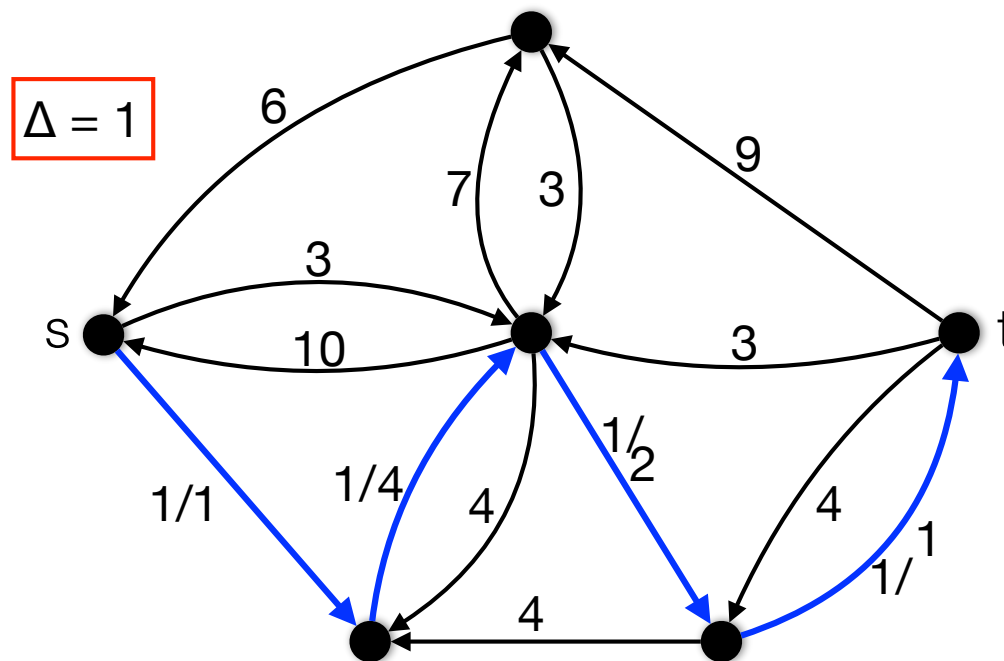
Scaling algorithm

- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_f(\Delta)$.
- Start with $\Delta = \text{"highest power of 2 } \leq \text{largest capacity out of } s\text{"}$
- When no more augmenting paths in $G_f(\Delta)$: $\Delta = \Delta/2$ (new phase).



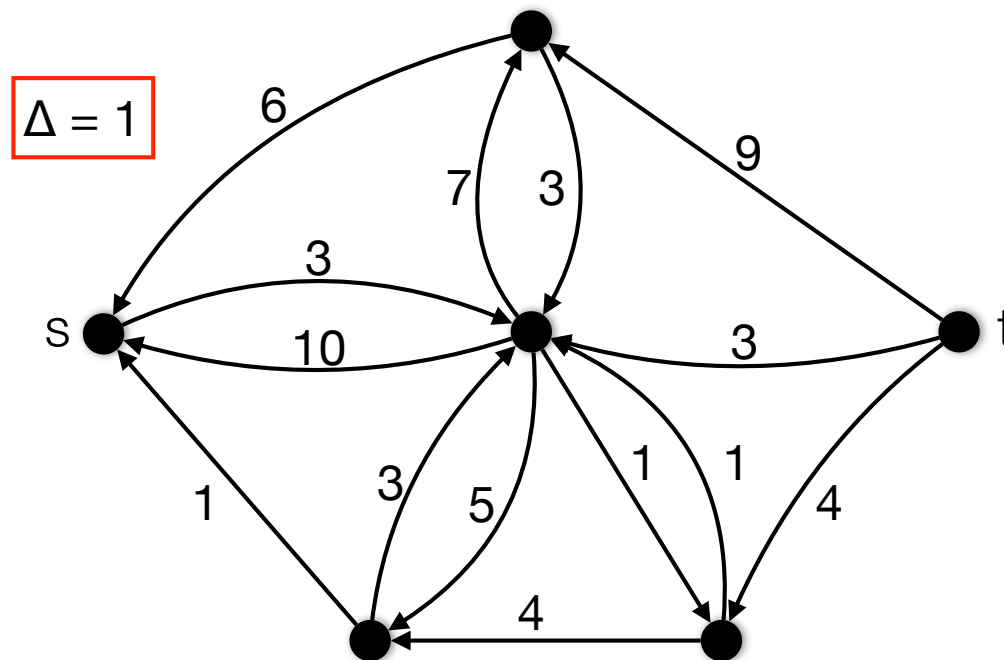
Scaling algorithm

- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_f(\Delta)$.
- Start with $\Delta = \text{“highest power of 2 } \leq \text{largest capacity out of } s\text{”}$
- When no more augmenting paths in $G_f(\Delta)$: $\Delta = \Delta/2$ (new phase).



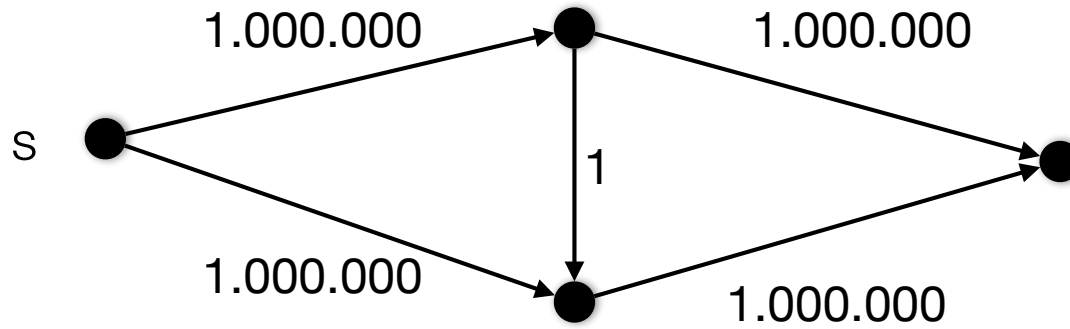
Scaling algorithm

- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_f(\Delta)$.
- Start with $\Delta = \text{“highest power of 2 } \leq \text{largest capacity out of } s\text{”}$
- When no more augmenting paths in $G_f(\Delta)$: $\Delta = \Delta/2$ (new phase).

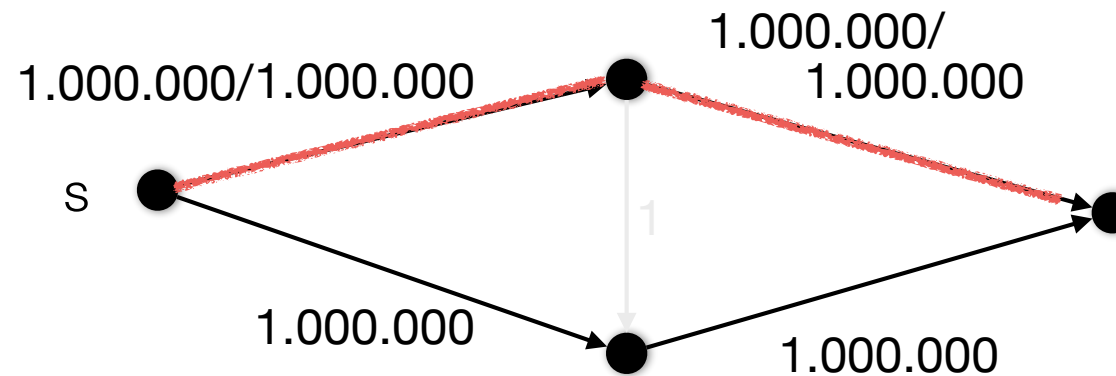
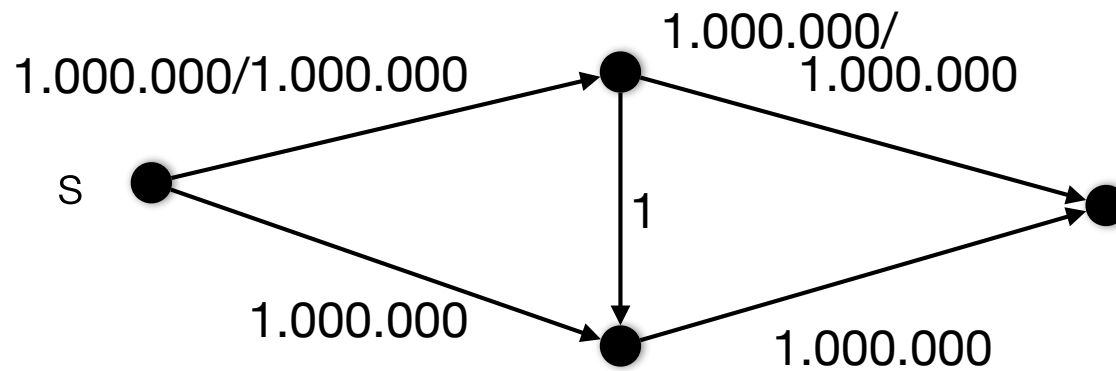


- Stop when no more augmenting paths in $G_f(1)$.

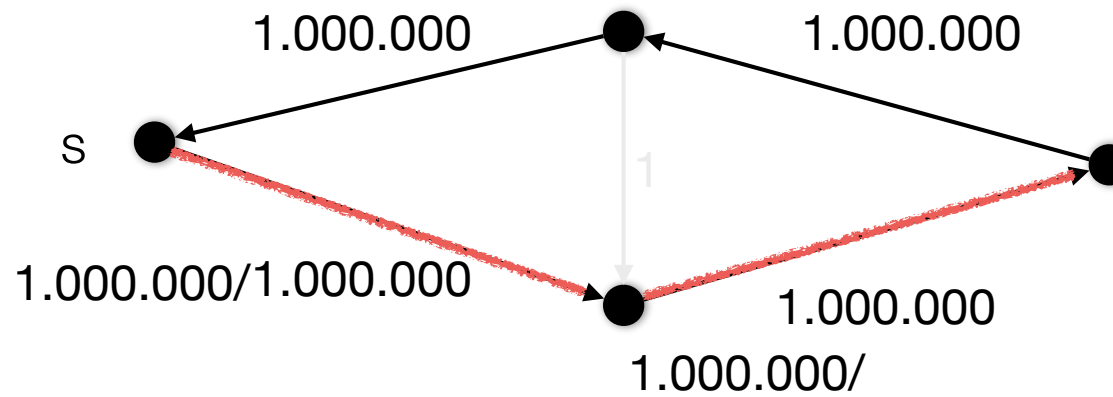
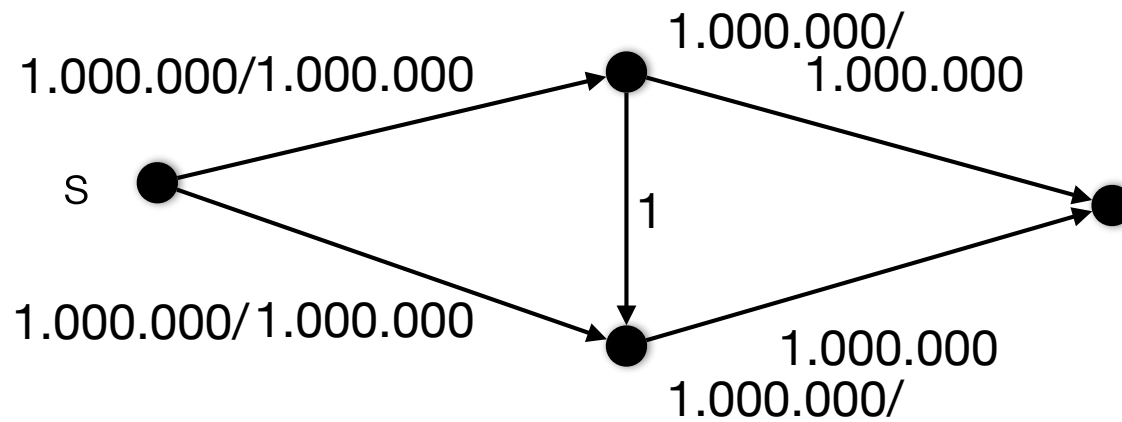
Scaling algorithm



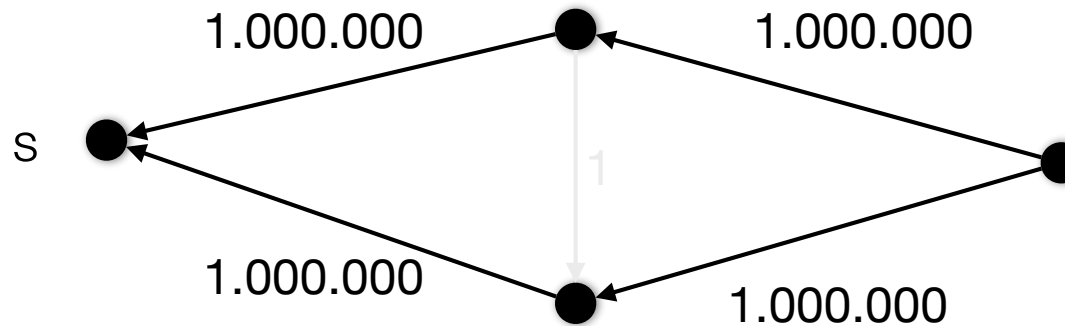
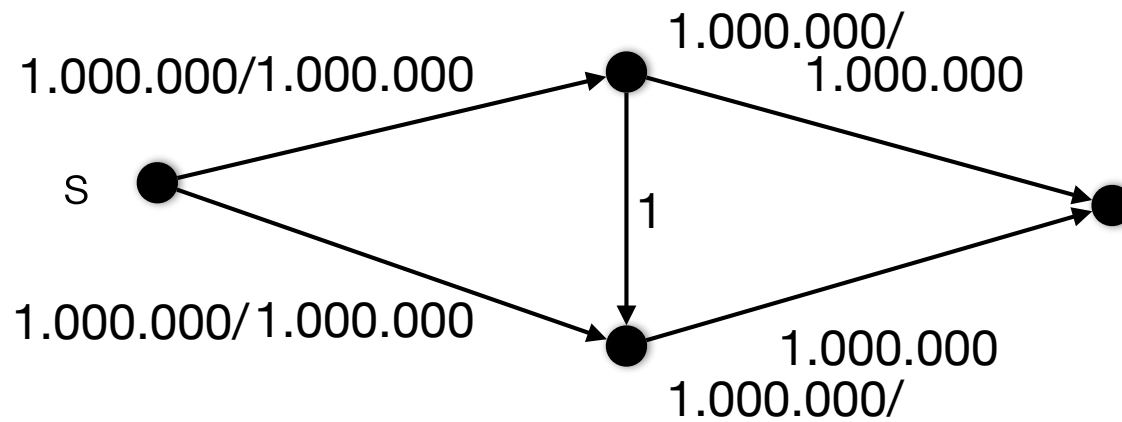
Scaling algorithm



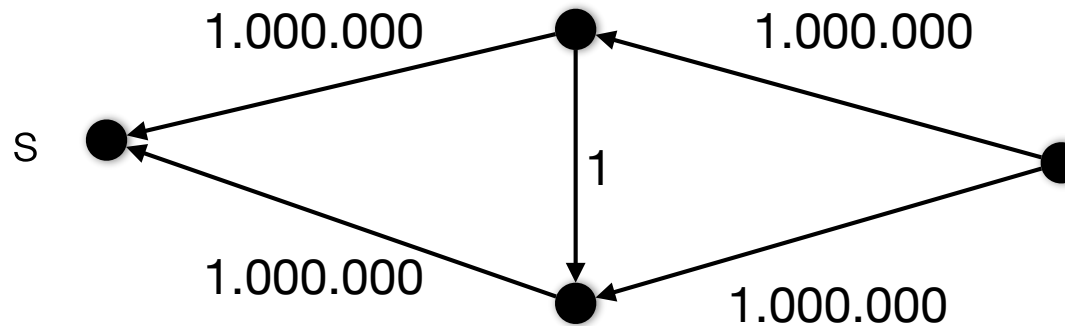
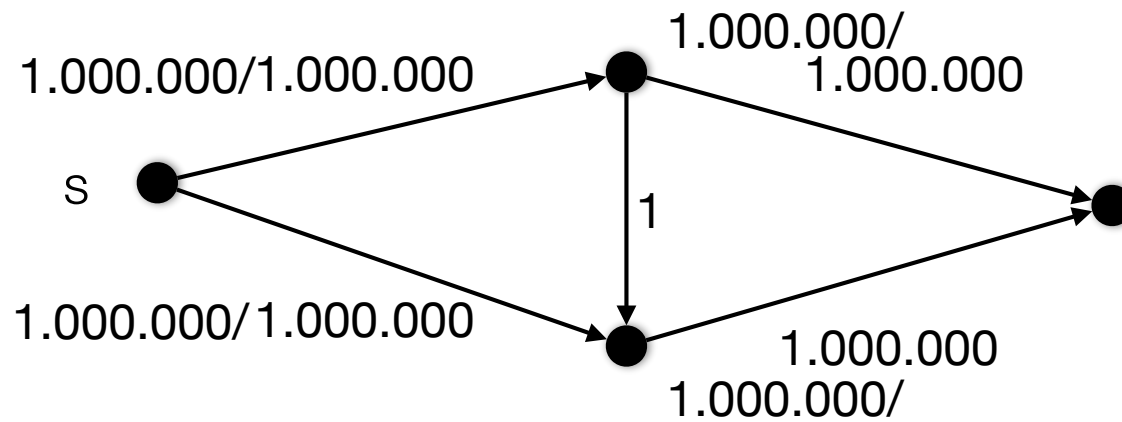
Scaling algorithm



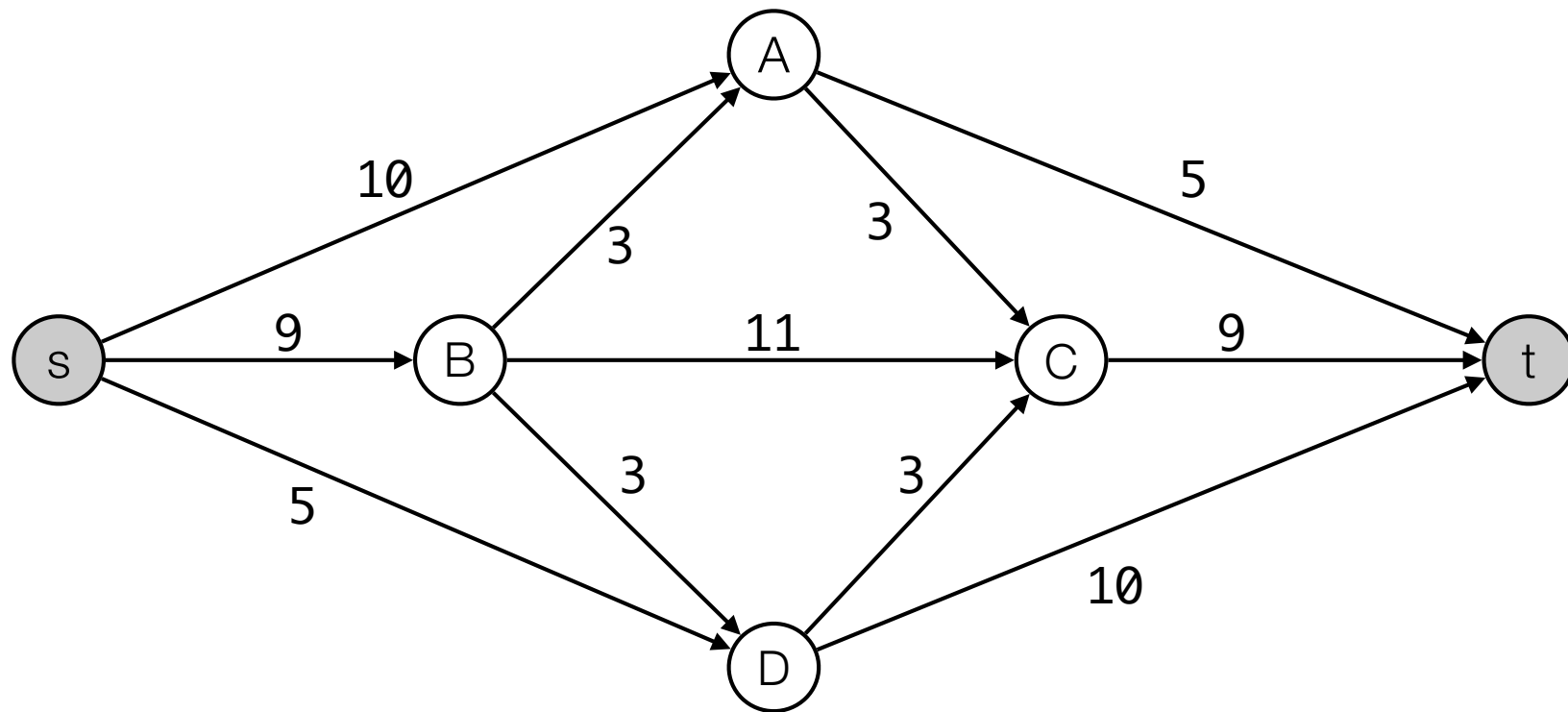
Scaling algorithm



Scaling algorithm



Exercise

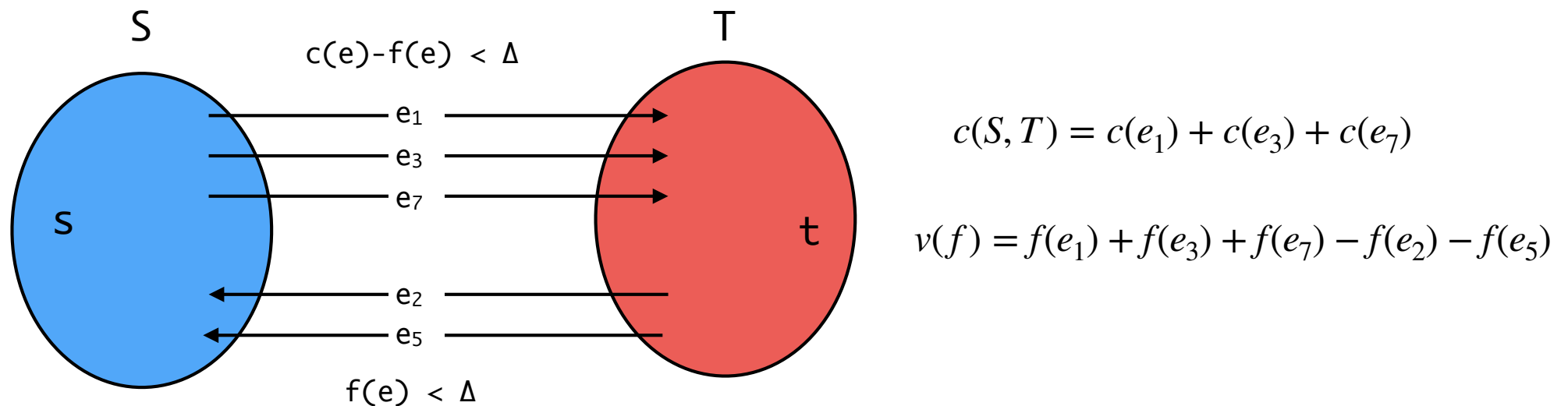


Scaling algorithm

- **Running time:** $O(m^2 \log C)$, where C is the largest capacity out of s .
- **Lemma 1.** *Number of scaling phases: $1 + \lceil \lg C \rceil$*
- Prove: number of augmentation paths found in a scaling phase is at most $2m$.
- **Lemma 2.** *Let f the flow when Δ -scaling phase ends, and let f^* be the maximum flow. Then $v(f^*) \leq v(f) + m\Delta$.*

Scaling algorithm

- **Lemma 2.** Let f the flow when Δ -scaling phase ends, and let f^* be the maximum flow. Then $v(f^*) \leq v(f) + m\Delta$.
- By the end of the phase there is a cut $c(S,T) \leq v(f) + m\Delta$.



$$\begin{aligned}
 c(S, T) - v(f) &= c(e_1) + c(e_3) + c(e_7) - f(e_1) - f(e_3) - f(e_7) + f(e_2) + f(e_5) \\
 &= c(e_1) - f(e_1) + c(e_3) - f(e_3) + c(e_7) - f(e_7) + f(e_2) + f(e_5) \\
 &< \Delta + \Delta + \Delta + \Delta + \Delta = 5\Delta
 \end{aligned}$$

Scaling algorithm

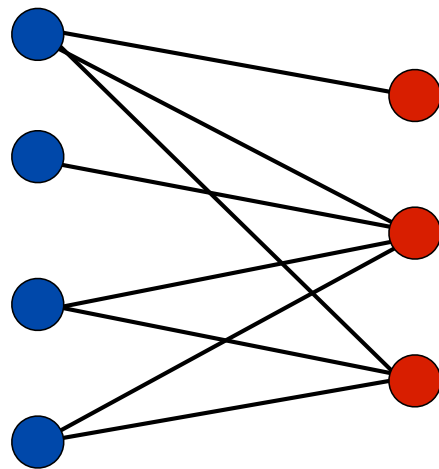
- **Running time:** $O(m^2 \log C)$, where C is the largest capacity out of s .
- **Lemma 1.** *Number of scaling phases: $1 + \lceil \lg C \rceil$*
- **Lemma 2.** *Let f the flow when Δ -scaling phase ends, and let f^* be the maximum flow. Then $v(f^*) \leq v(f) + m\Delta$.*
- **Lemma 3.** *The number of augmentations in a scaling phase is at most $2m$.*
 - First phase: can use each edge out of s in at most one augmenting path.
 - f flow at the end of previous phase.
 - Used $\Delta' = 2\Delta$ in last round.
 - Lemma 2: $v(f^*) \leq v(f) + m\Delta' = v(f) + 2m\Delta$.
 - “Leftover flow” to be found $\leq 2m\Delta$.
 - Each augmentation in a Δ -scaling phase augments flow with at least Δ .

Maximum flow algorithms

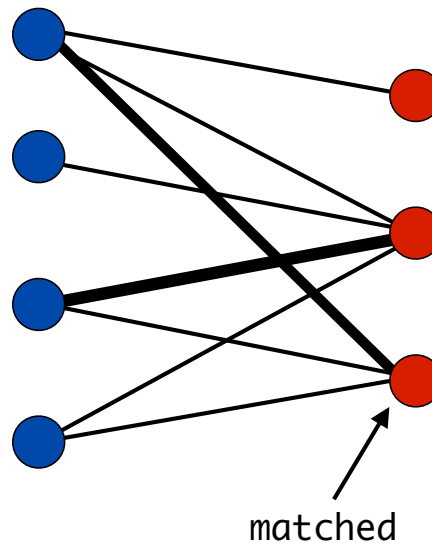
- **Edmonds-Karp**: $O(m^2n)$
- **Scaling**: $O(m^2 \log C)$
- **Ford-Fulkerson**: $O(m \cdot v(f))$.
- Preflow-push $O(n^3)$
- Other algorithms: $O(mn \log n)$ or $O(\min(n^{2/3}, m^{1/2})m \log n \log U)$.

Maximum Bipartite Matching

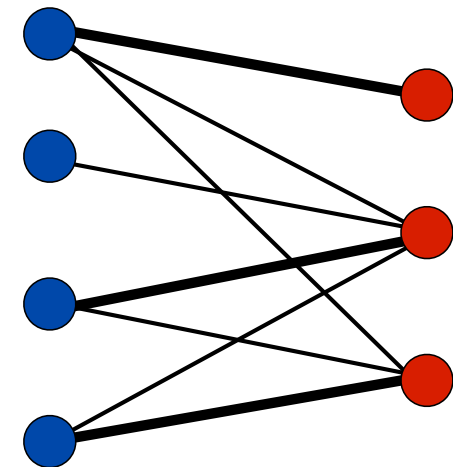
- **Bipartite graph:** Can color vertices red and blue such that all edges have a red and a blue endpoint.
- **Matching:** Subset of edges $M \subseteq E$ such that no edges in M share an endpoint.
- **Maximum matching:** matching of maximum cardinality.
- **Applications:**
 - planes to routes
 - jobs to workers/machines



Matching

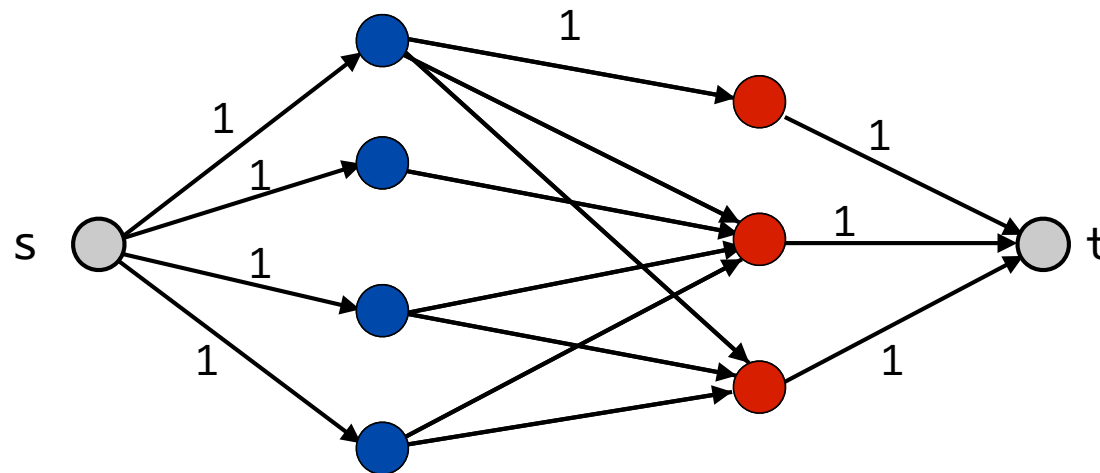


Maximum matching



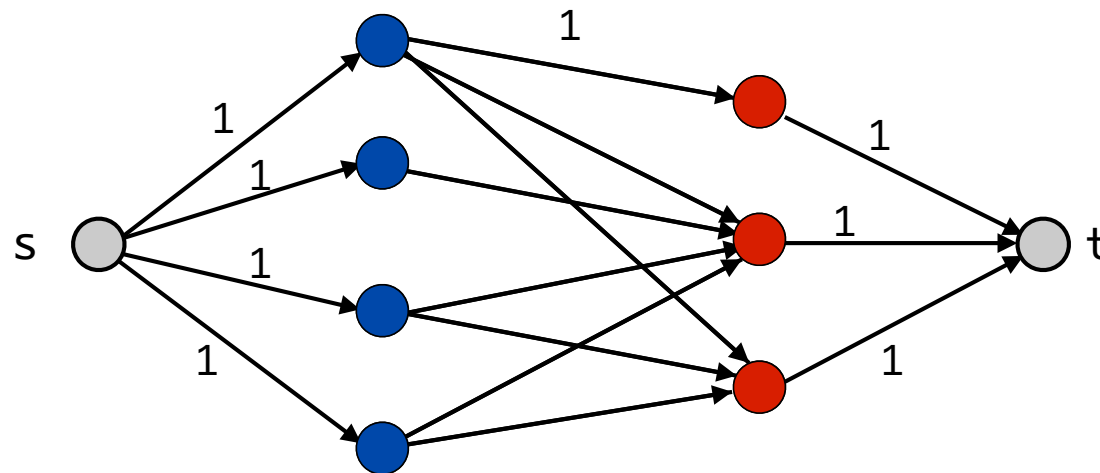
Maximum Bipartite Matching

- **Bipartite graph:** Can color vertices red and blue such that all edges have a red and a blue endpoint.
- **Matching:** Subset of edges $M \subseteq E$ such that no edges in M share an endpoint.
- **Maximum matching:** matching of maximum cardinality.
- Solve via flow:



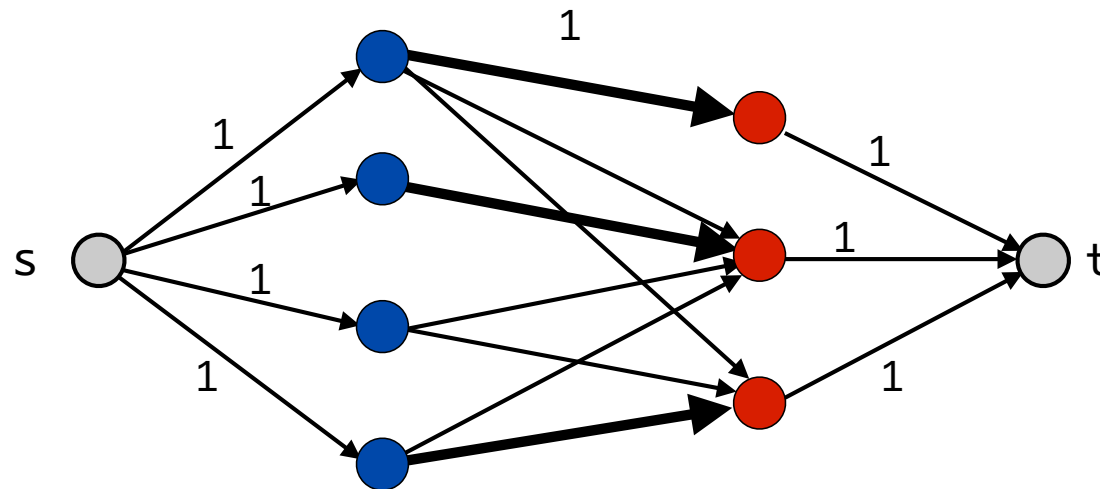
Maximum Bipartite Matching

- **Bipartite graph:** Can color vertices red and blue such that all edges have a red and a blue endpoint.
- **Matching:** Subset of edges $M \subseteq E$ such that no edges in M share an endpoint.
- **Maximum matching:** matching of maximum cardinality.
- Solve via flow:
 - Matching $M \Rightarrow$ flow of value $|M|$



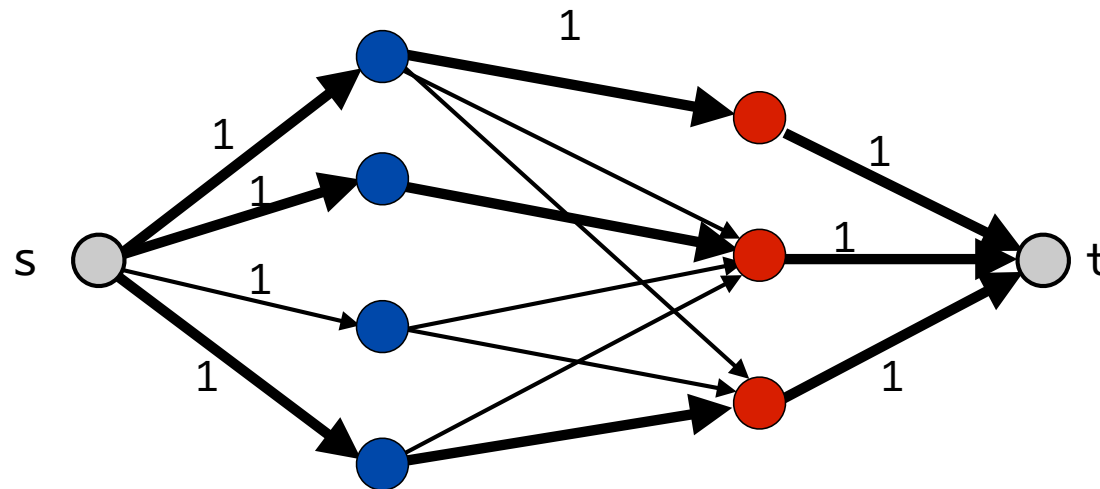
Maximum Bipartite Matching

- **Bipartite graph:** Can color vertices red and blue such that all edges have a red and a blue endpoint.
- **Matching:** Subset of edges $M \subseteq E$ such that no edges in M share an endpoint.
- **Maximum matching:** matching of maximum cardinality.
- Solve via flow:
 - Matching $M \Rightarrow$ flow of value $|M|$



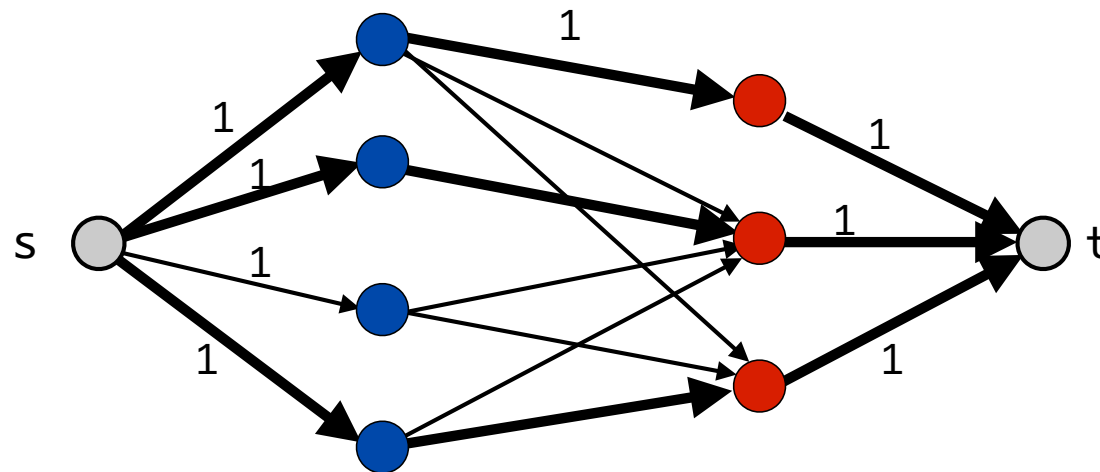
Maximum Bipartite Matching

- **Bipartite graph:** Can color vertices red and blue such that all edges have a red and a blue endpoint.
- **Matching:** Subset of edges $M \subseteq E$ such that no edges in M share an endpoint.
- **Maximum matching:** matching of maximum cardinality.
- Solve via flow:
 - Matching $M \Rightarrow$ flow of value $|M|$



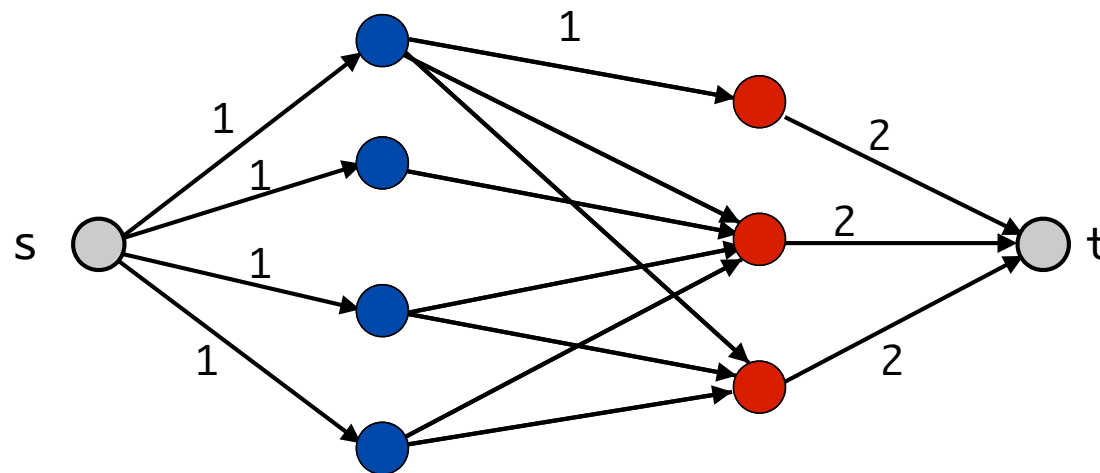
Maximum Bipartite Matching

- **Bipartite graph:** Can color vertices red and blue such that all edges have a red and a blue endpoint.
- **Matching:** Subset of edges $M \subseteq E$ such that no edges in M share an endpoint.
- **Maximum matching:** matching of maximum cardinality.
- Solve via flow:
 - Matching $M \Rightarrow$ flow of value $|M|$
 - Flow of value $v(f) \Rightarrow$ matching of size $v(f)$



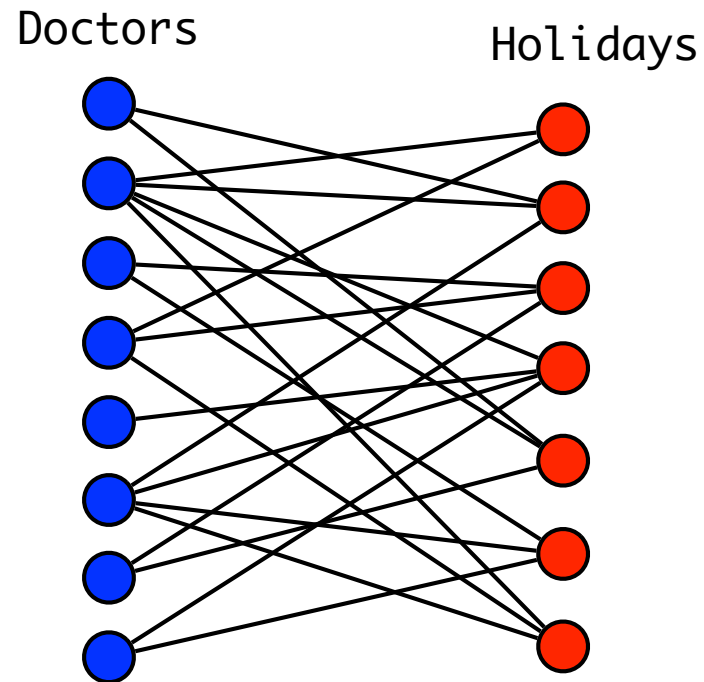
Maximum Bipartite Matching

- **Bipartite graph:** Can color vertices red and blue such that all edges have a red and a blue endpoint.
- **Matching:** Subset of edges $M \subseteq E$ such that no edges in M share an endpoint.
- **Maximum matching:** matching of maximum cardinality.
- Solve via flow:
- Can generalize to general matchings



Scheduling of doctors

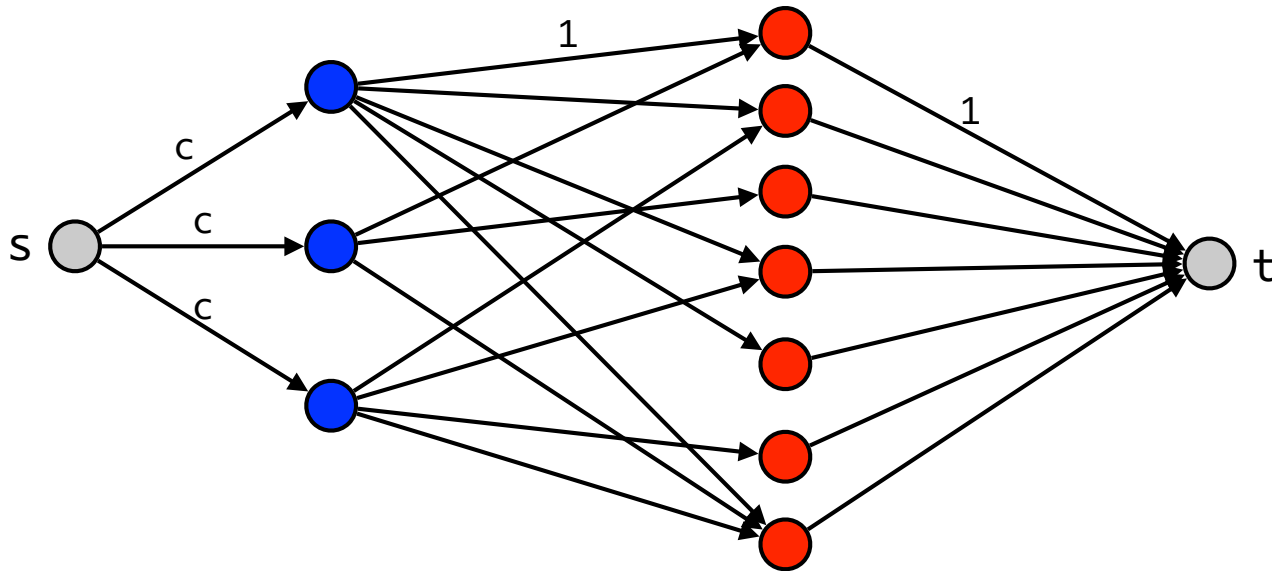
- X doctors, Y holidays, each doctor should work at at most 1 holiday, each doctor is available at some of the holidays.



- Same problem, but each doctor should work at most c holidays?

Scheduling of doctors

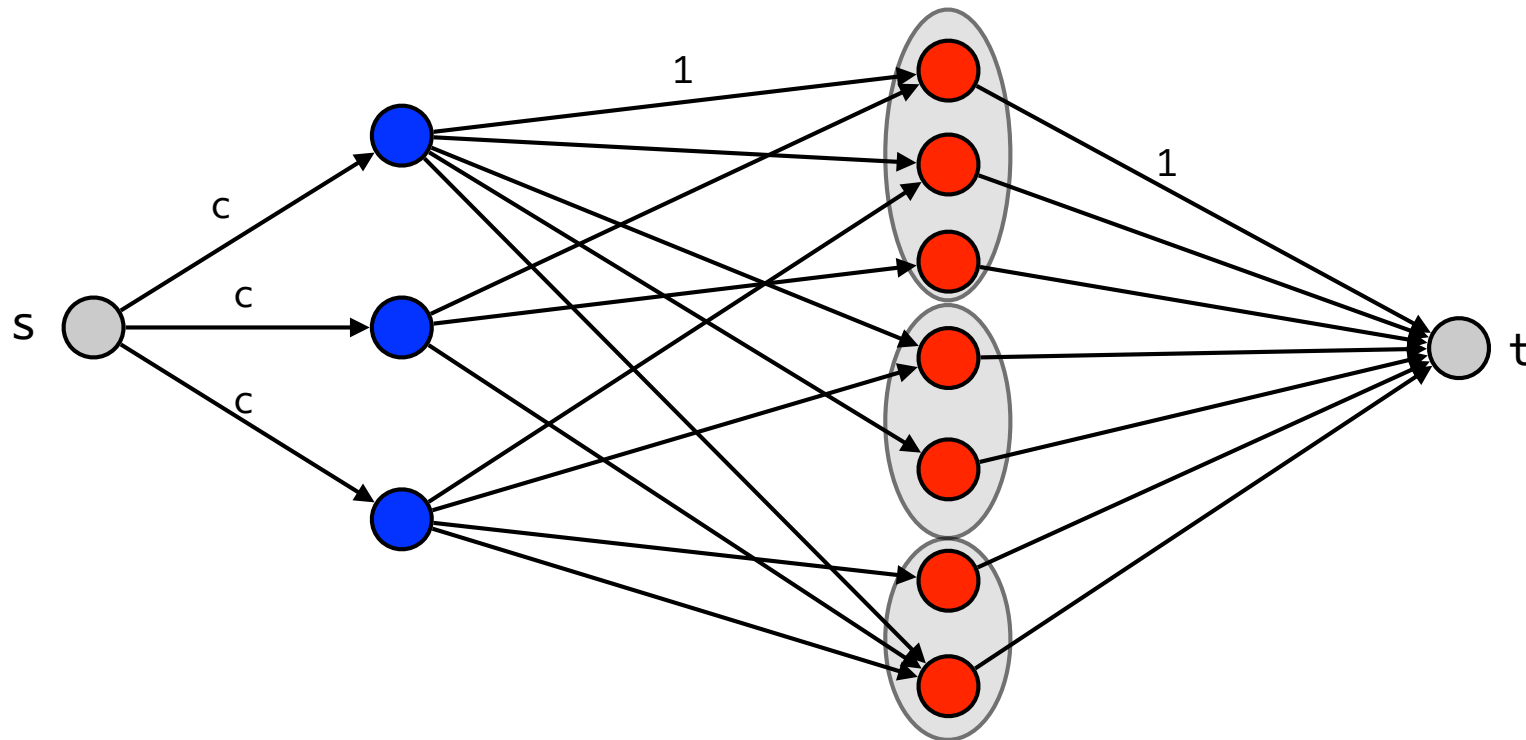
- X doctors, Y holidays, each doctor should work at at most c holidays, each doctor is available at some of the holidays.



- Same problem, but each doctor should work at most one day in each vacation period?

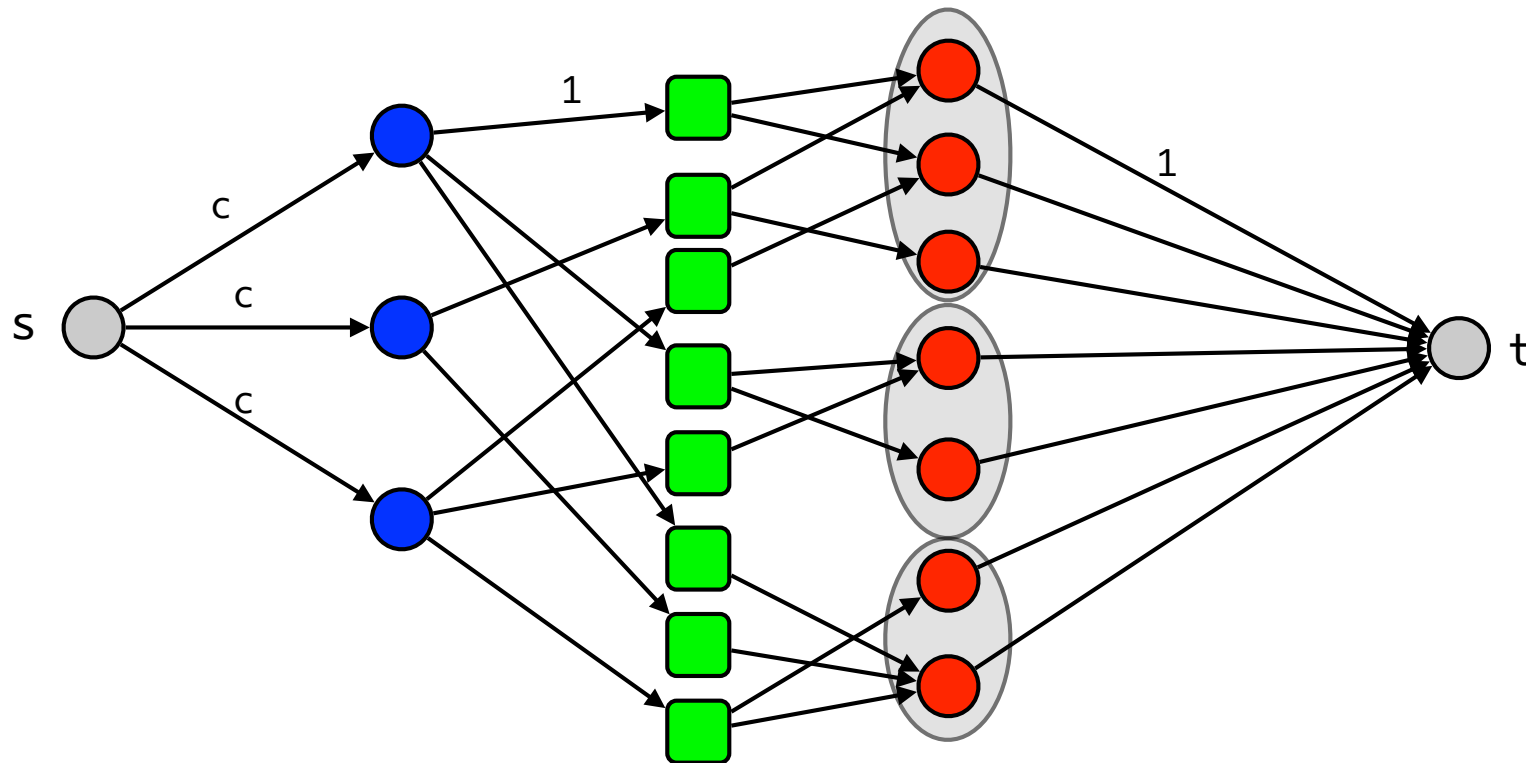
Scheduling of doctors

- X doctors, Y holidays, each doctor should work at at most c holidays, each doctor is available at some of the holidays.
- Same problem, but each doctor should work at most one day in each vacation period?



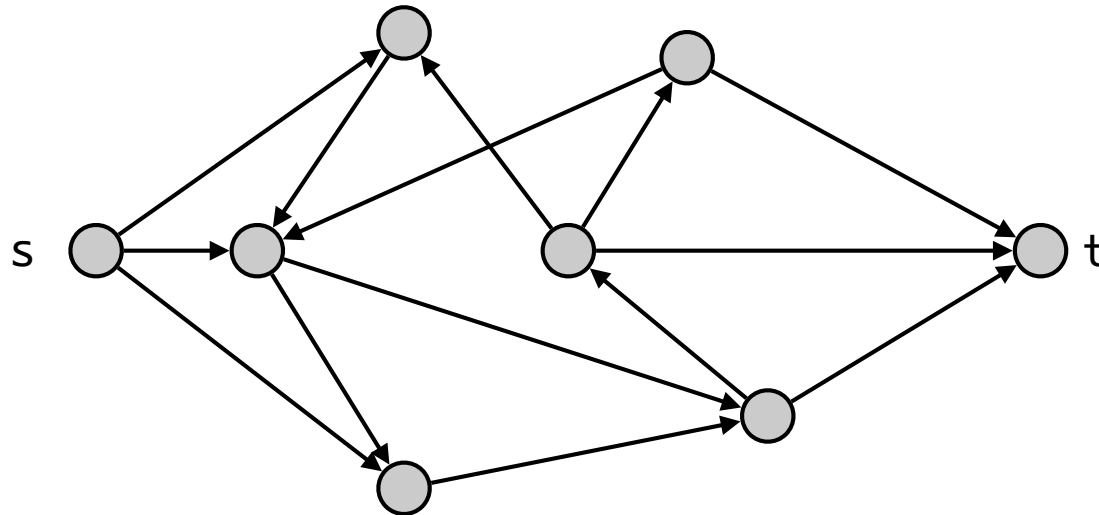
Scheduling of doctors

- X doctors, Y holidays, each doctor should work at at most c holidays, each doctor is available at some of the holidays.
- Same problem, but each doctor should work at most one day in each vacation period?



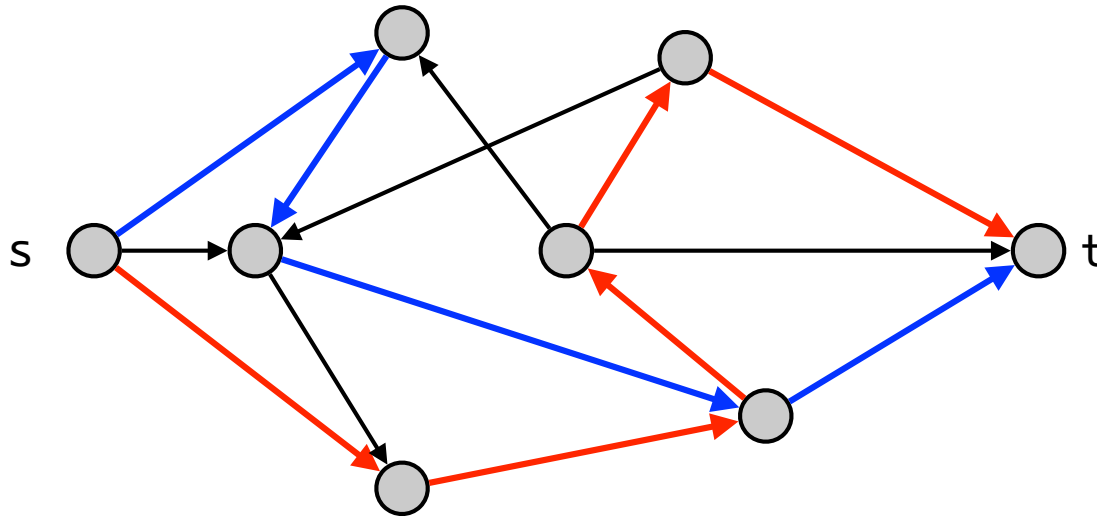
Edge Disjoint paths

- Problem: Find maximum number of edge-disjoint paths from s to t .
- Two paths are edge-disjoint if they have no edge in common.



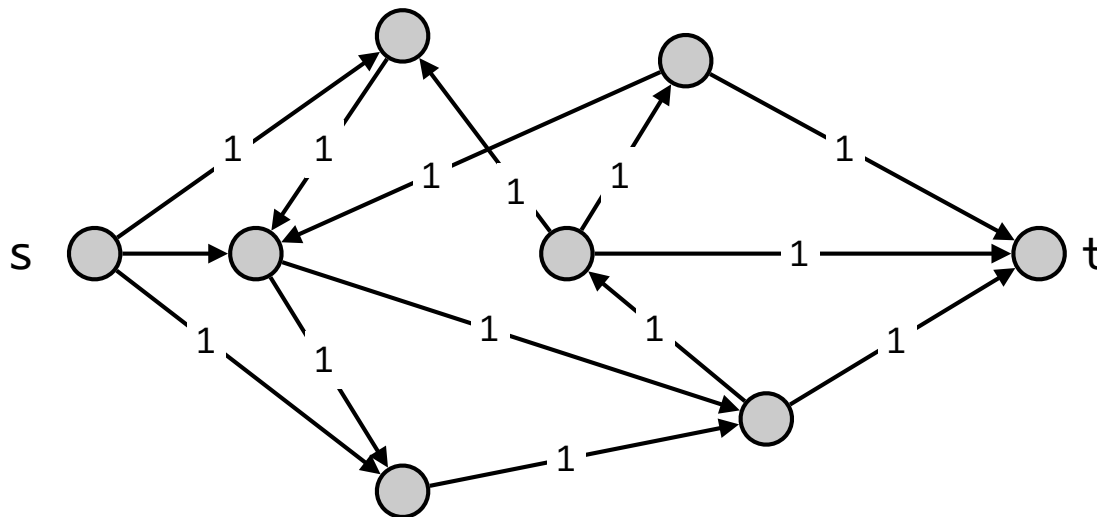
Edge Disjoint paths

- **Edge-disjoint path problem.** Find the maximum number of edge-disjoint paths from s to t .
- Two paths are edge-disjoint if they have no edge in common.



Edge Disjoint Paths

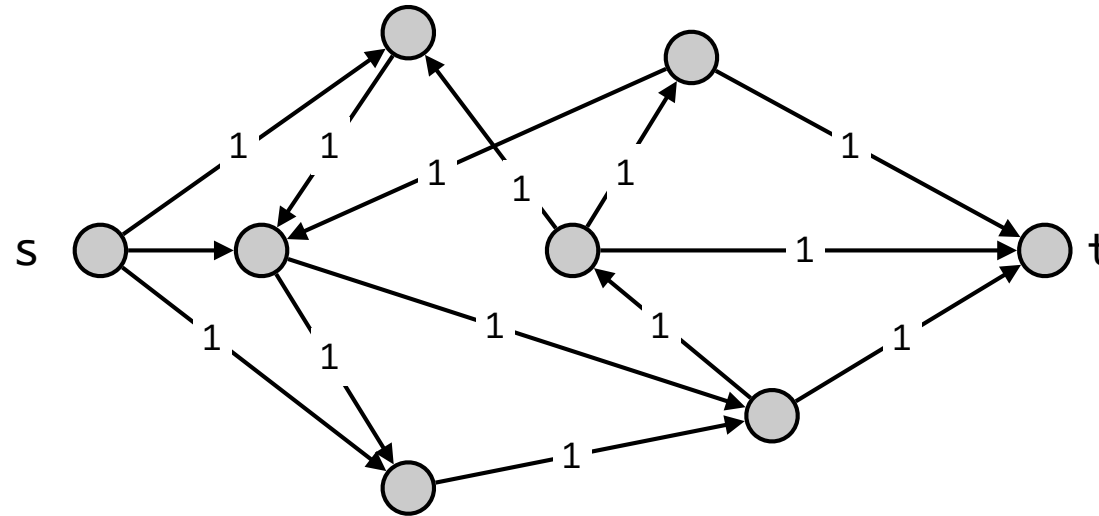
- Reduction to max flow: assign capacity 1 to each edge.



- **Thm.** Max number of edge-disjoint s-t paths is equal to the value of a maximum flow.
 - Suppose there are k edge-disjoint paths: then there is a flow of k (let all edges on the paths have flow 1).
 - Other way (graph theory course).
- Ford-Fulkerson: $v(f) \leq n$ (no multiple edges and therefore at most n edges out of s) $\Rightarrow$ running time $O(nm)$.

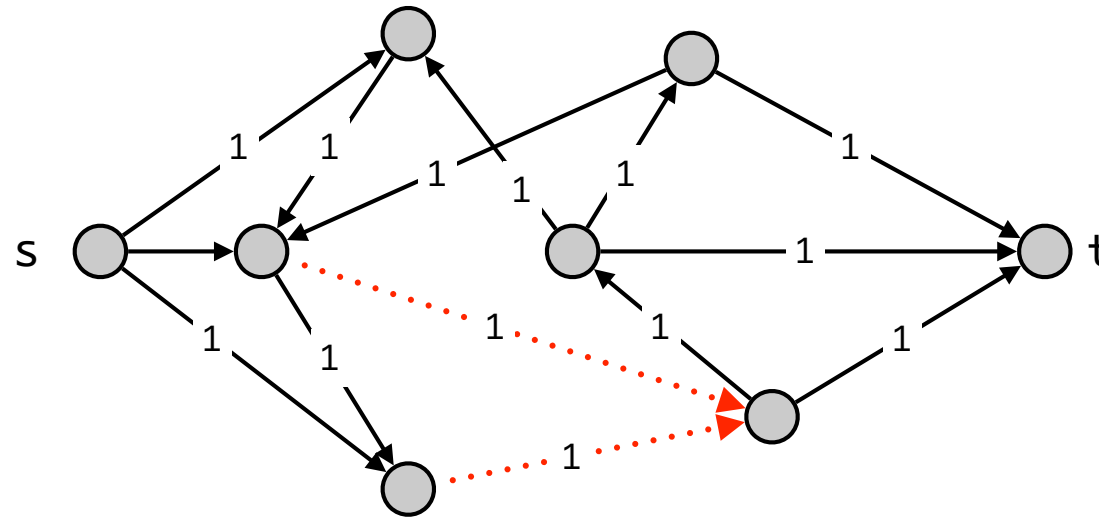
Network Connectivity

- **Network connectivity.** Find minimum number of edges whose removal disconnects t from s (destroys all s - t paths).



Network Connectivity

- **Network connectivity.** Find minimum number of edges whose removal disconnects t from s (destroys all s - t paths).



- Set all capacities to 1 and find minimum cut.
- Thm. (Menger) The maximum number of edge-disjoint s - t paths is equal to the minimum number of edges whose removal disconnects t from s .

Baseball elimination

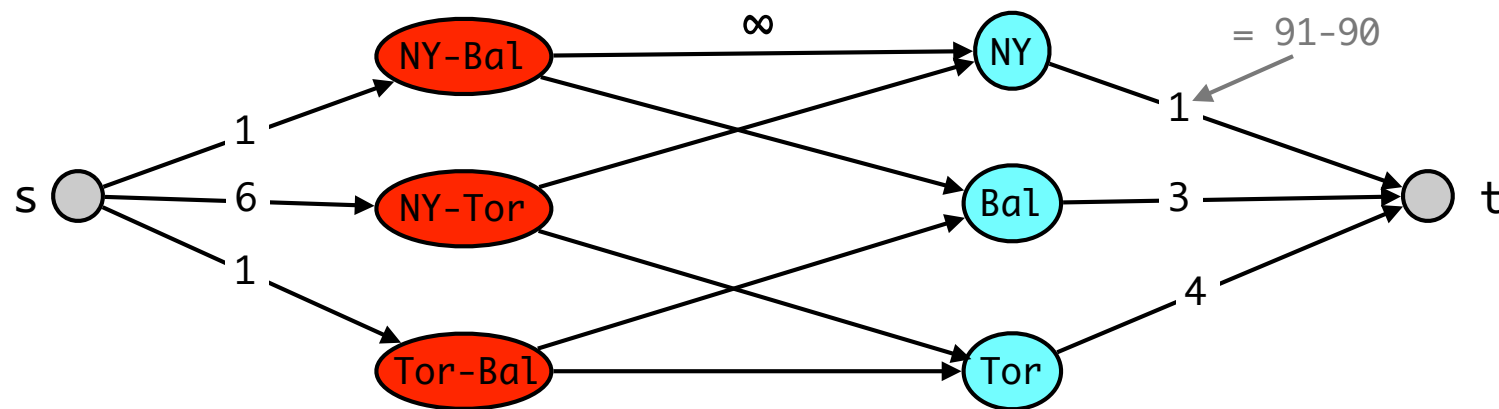
Team	Wins	Games left	Against			
			NY	Bal	Tor	Bos
New York	92	2	-	1	1	0
Baltimore	91	3	1	-	1	1
Toronto	91	3	1	1	-	1
Boston	90	2	0	1	1	-

- Question: Can Boston finish in first place (or in tie of first place)?
- No: Boston must win both its remaining 2 and NY must loose. But then Baltimore and Toronto both beat NY so winner of Baltimore-Toronto will get 93 points.
- Other argument: Boston can finish with at most 92. Cumulatively the other three teams have 274 wins currently and their 3 games against each other will give another 3 points => 277. $277/3 = 92,33333$ => one of them must win > 92 .

Baseball elimination

Team	Wins	Games left	Against			
			NY	Bal	Tor	Bos
New York	90	11	-	1	6	4
Baltimore	88	6	1	-	1	4
Toronto	87	11	6	1	-	4
Boston	79	12	4	4	4	-

- Question: Can Boston finish in first place (or in tie of first place)?



Boston can get at most $79 + 12 = 91$ points

- Boston cannot win (or tie) $\Leftrightarrow \text{max s-t flow} < 8$.

Node capacities

- Capacities on nodes.

