Lecture 13

More problems on Maxflow, Maximum Matching, Baseball elimination, Vertex cover

CS 161 Design and Analysis of Algorithms

Ioannis Panageas

Flow of a Network (Recap)

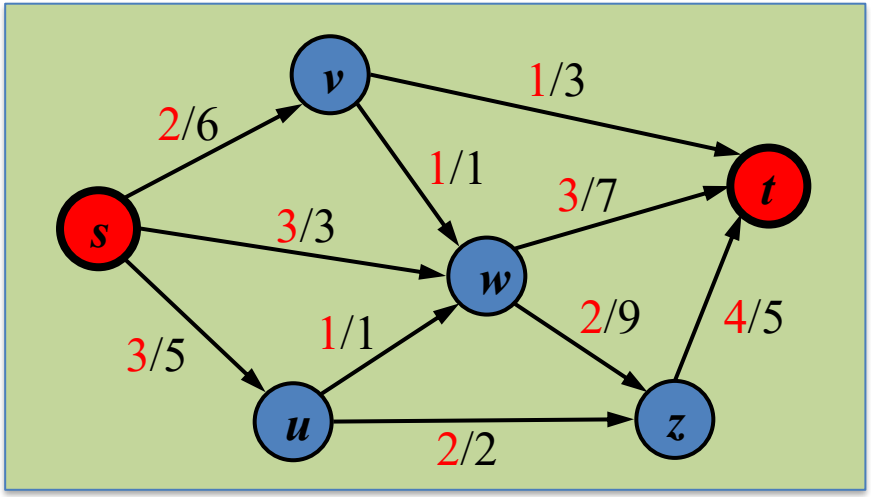
Definition: Function $f : E \to \mathbb{N}$ from edges to non-negative integers so that for each edge e it holds

$$0 \leq f(e) \leq c(e) \quad \text{Capacity constraint}$$

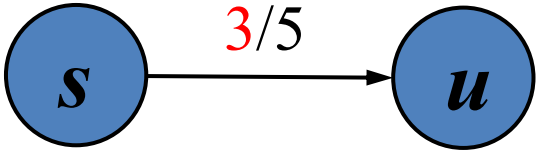
$$\sum_{e \in \mathbf{outgoing}(u)} f(e) = \sum_{e \in \mathbf{incoming}(u)} f(e) \quad \text{Conservation rule}$$

for all $u \neq s, t$

Example:



Capacity constraint:



$$0 \leq 3 \leq 5$$

Flow of a Network (Recap)

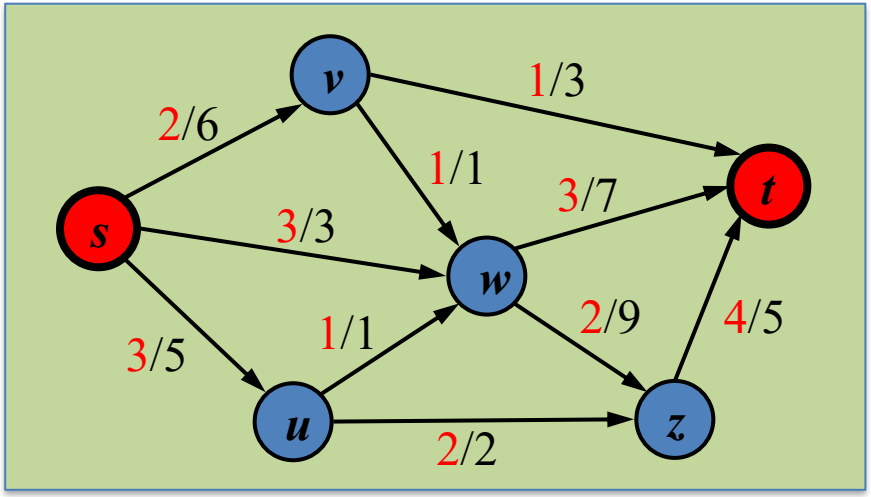
Definition: Function $f : E \to \mathbb{N}$ from edges to non-negative integers so that for each edge e it holds

$$0 \leq f(e) \leq c(e) \quad \text{Capacity constraint}$$

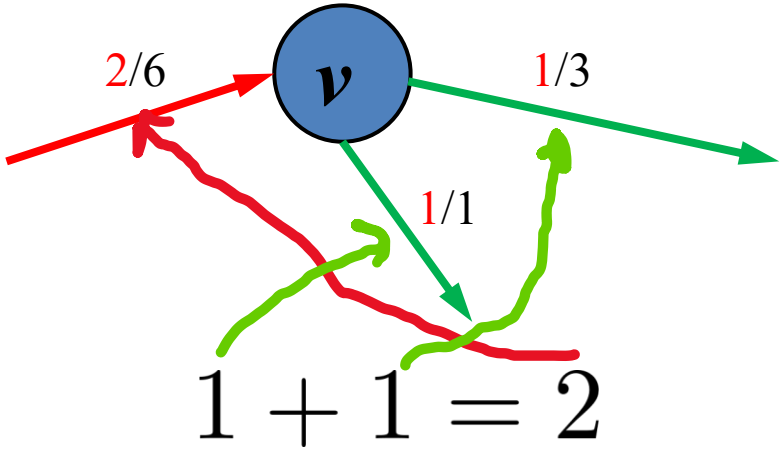
$$\sum_{e \in \mathbf{outgoing}(u)} f(e) = \sum_{e \in \mathbf{incoming}(u)} f(e) \quad \text{Conservation rule}$$

for all $u \neq s, t$

Example:



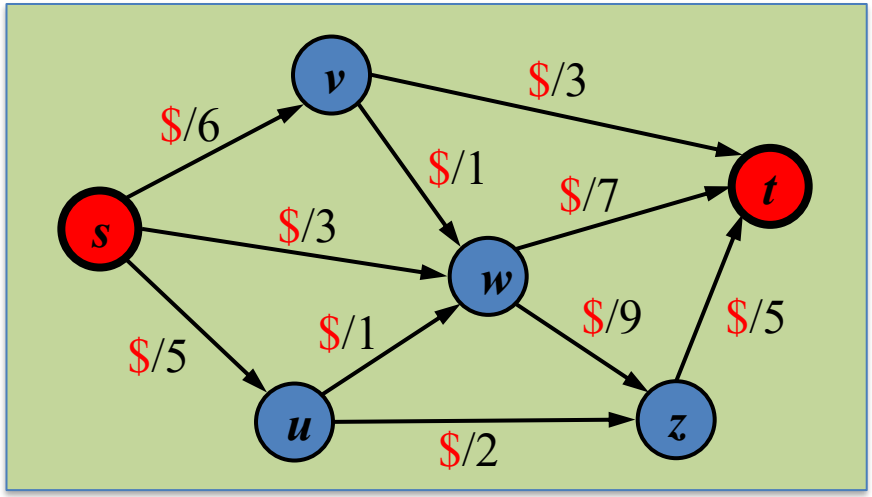
Conservation rule:



Maxflow Problem (Recap)

Problem: Given a network G, a source s and a sink t, and capacities on the edges, compute the maximum possible flow value $|f^*|$.

Example:



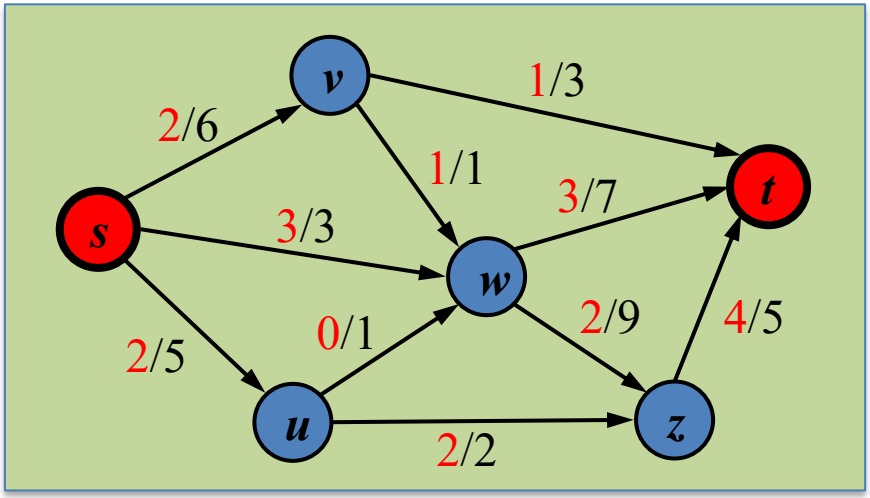
Find the $ to get maxflow $|f^*|$

Augmenting paths (Recap)

We are given a network G with edge capacities c and a flow f.
Let (u, v) be an edge from u to v.

Residual capacity from u to v is $\Delta_f(u, v) = c(u, v) - f(u, v)$.

Residual capacity from v to u is $\Delta_f(v, u) = f(u, v)$.
Imply the residual graph.



Augmenting path: Path from s to t with positive residual capacities.

$s \to v \to t$ augmenting path
$s \to u \to w \to v \to t$ augmenting path

$s \to u \to z \to t$ is **not**

The Ford-Fulkerson Algorithm

Main idea: Repeatedly search for an augmenting path π:

- If there is an augmenting path, augment flow with $\Delta_f(\pi)$ (minimum residual capacity among the edges of π) along the edges of π.
- If there is no augmenting path, terminate.

Remark: You can use DFS (or BFS) to search for an augmenting path.

Running time:

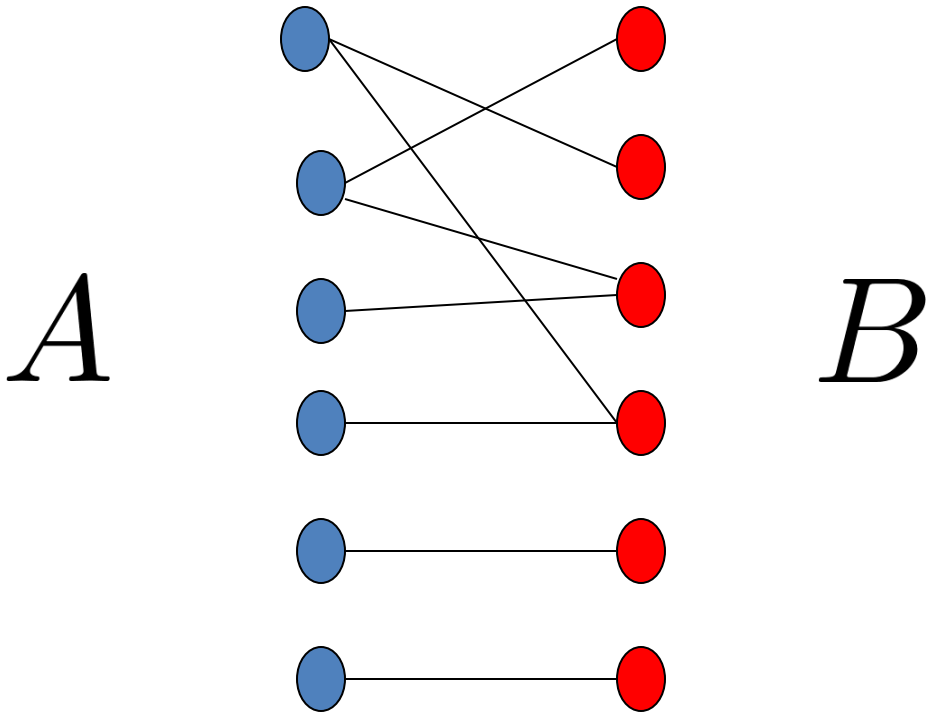
Time to search for an augmenting path $\times$ number of updates.

$$\Theta(|V| + |E|) \cdot |f^*|$$

Running time of DFS or BFS

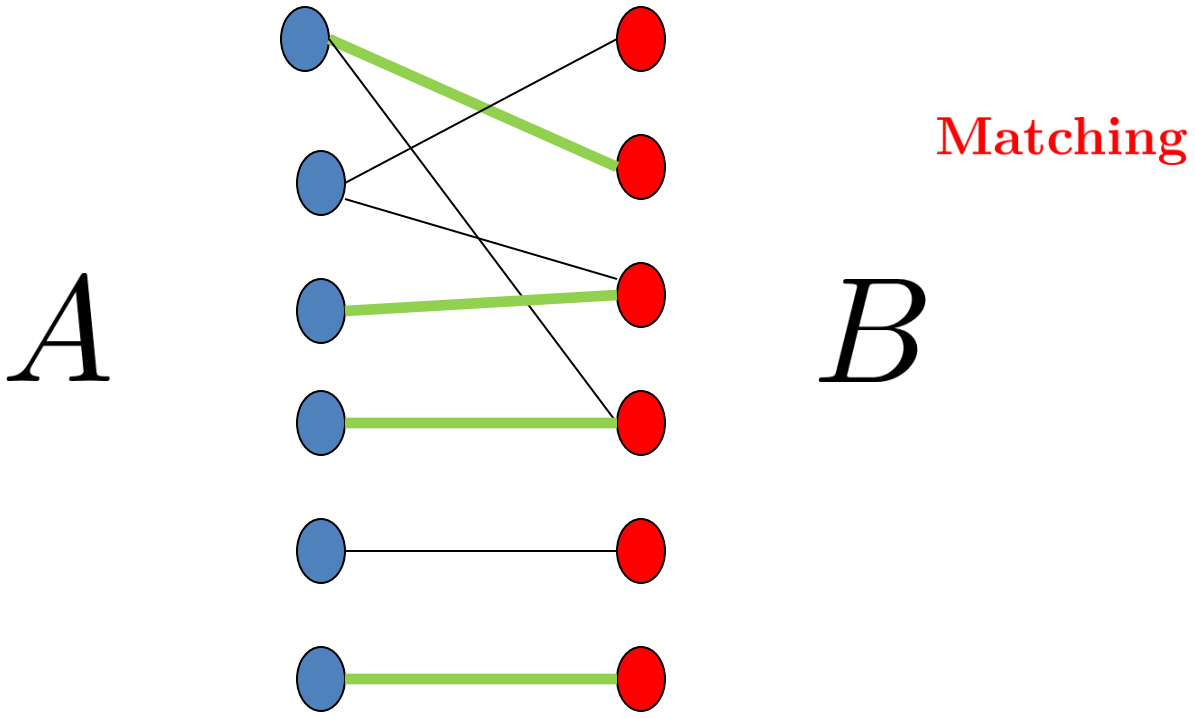
Updates increase flow by 1 unit only

Case study 0: Maximum Matching



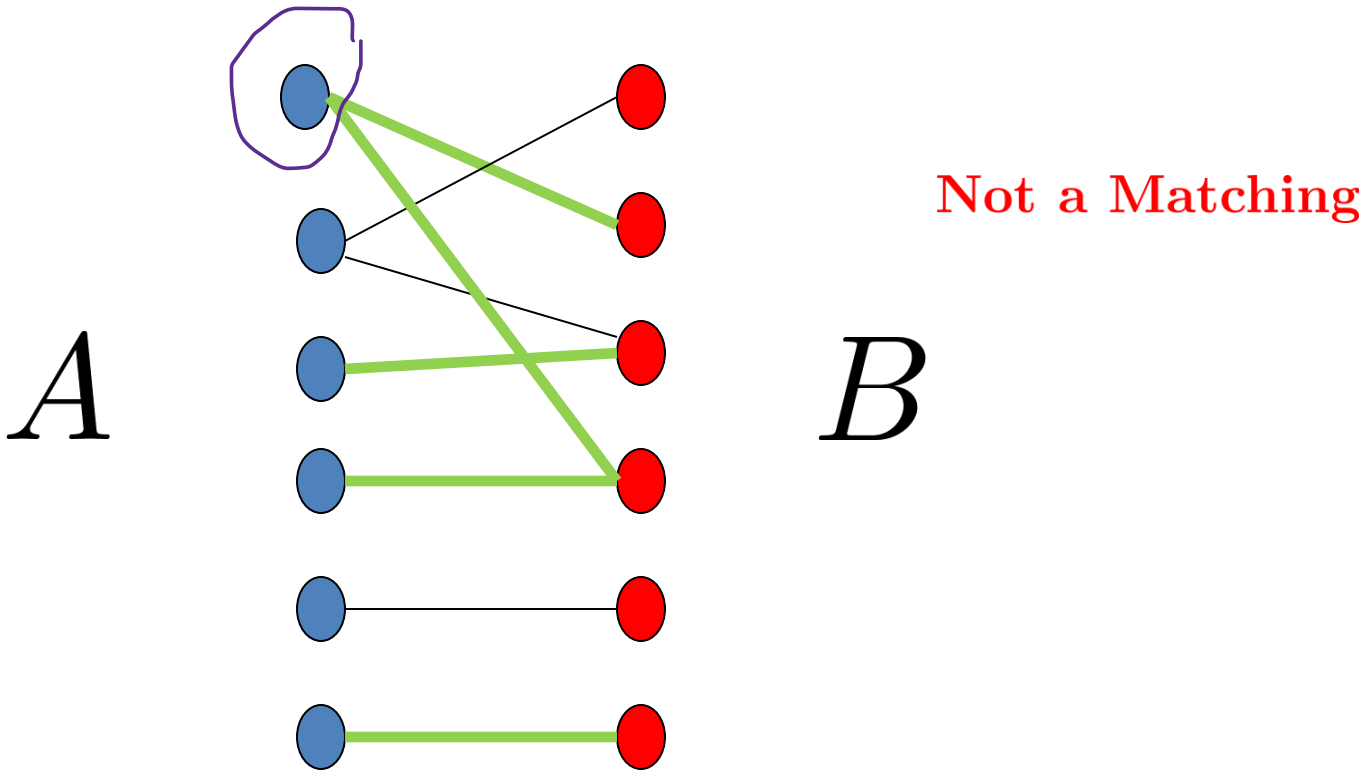
Definition: Given a **bipartite** graph, a **matching** is just a collection of edges that do not share a vertex.

Case study 0: Maximum Matching



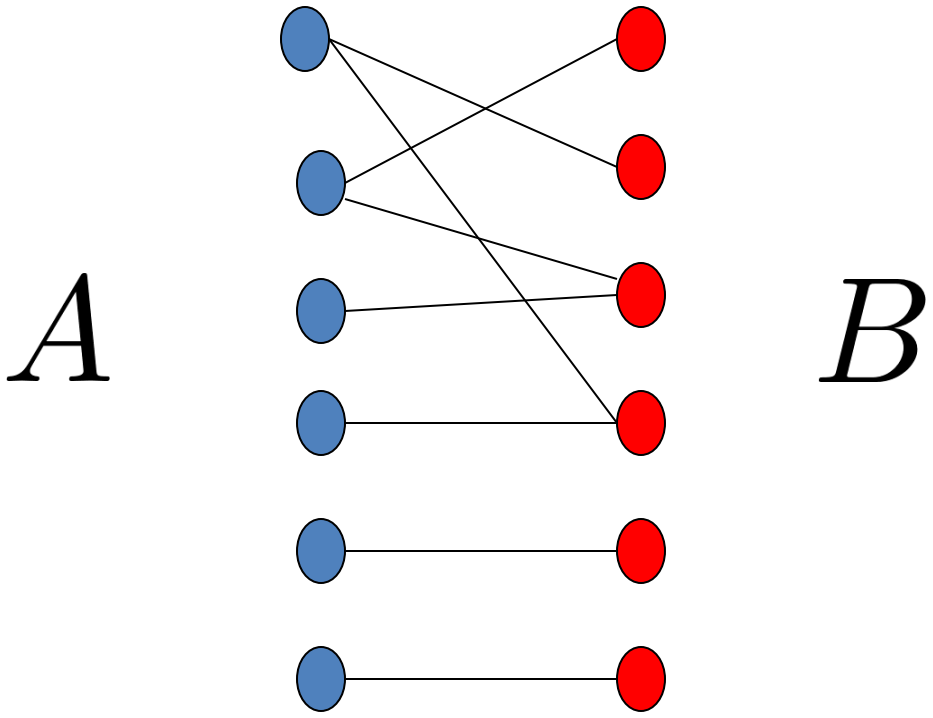
Definition: Given a **bipartite** graph, a **matching** is just a collection of edges that do not share a vertex.

Case study 0: Maximum Matching



Definition: Given a **bipartite** graph, a **matching** is just a collection of edges that do not share a vertex.

Case study 0: Maximum Matching

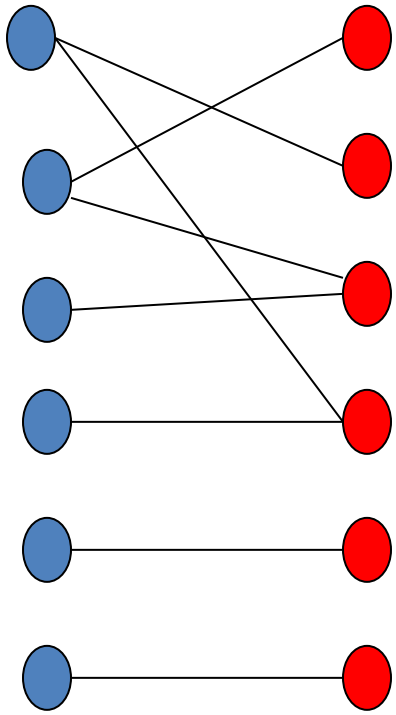


Problem: Given a **bipartite** graph, compute/find a maximum matching.

Case study 0: Maximum Matching

Problem: Given a **bipartite** graph, compute/find a maximum matching.

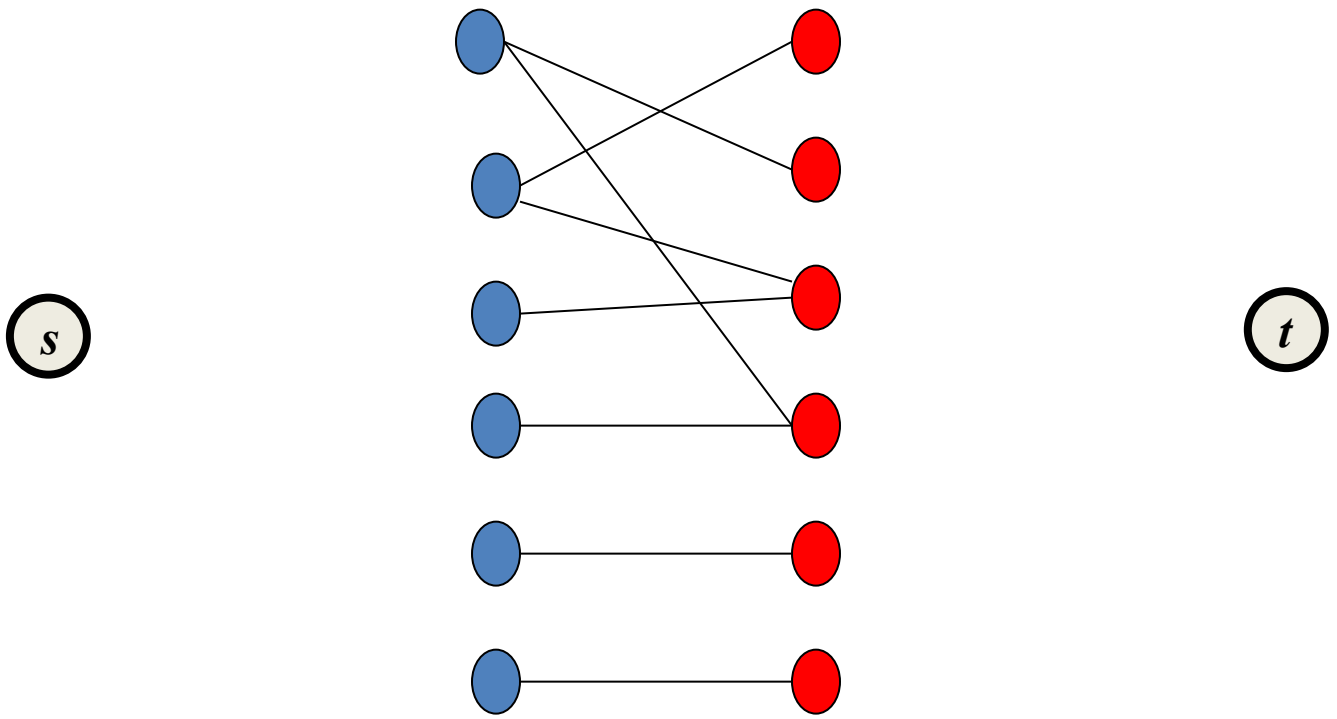
Idea: **Reduce** it to Maxflow problem. To do that, we need a network flow and s, t.



Case study 0: Maximum Matching

Problem: Given a **bipartite** graph, compute/find a maximum matching.

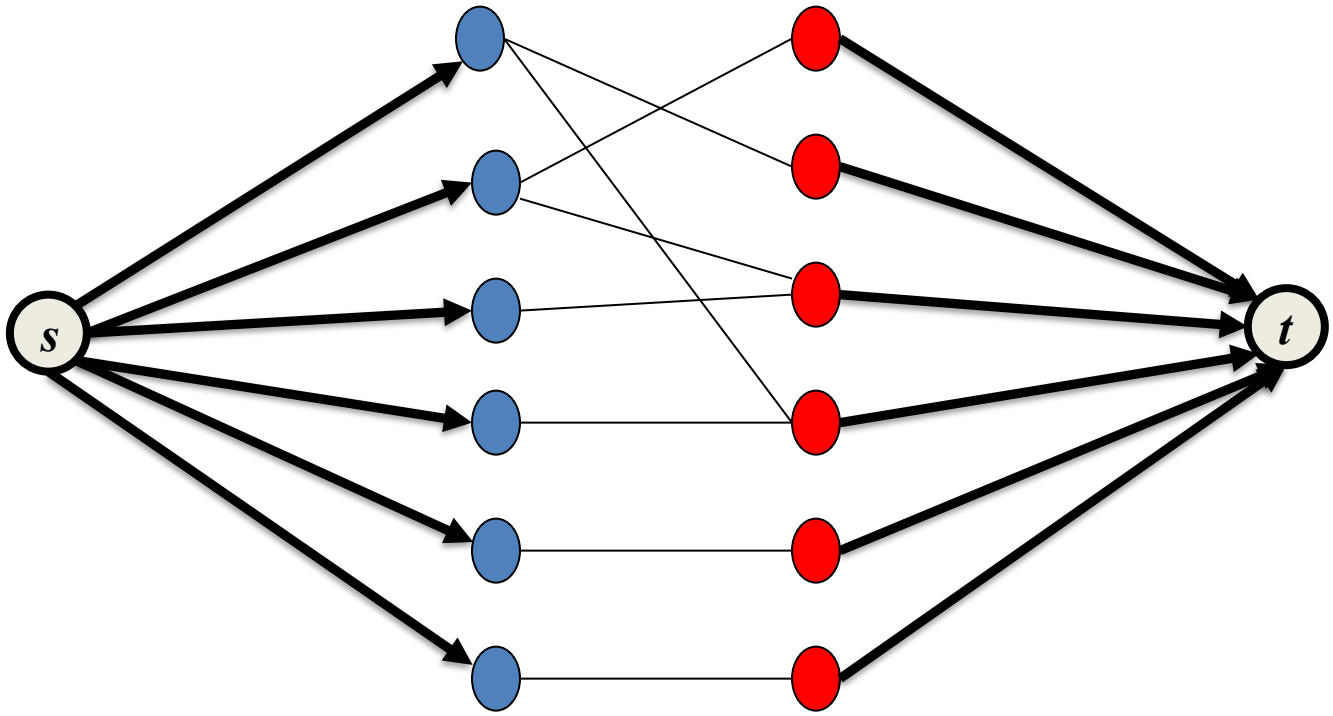
Idea: **Reduce** it to Maxflow problem. To do that, we need a network flow and s, t.



Case study 0: Maximum Matching

Problem: Given a **bipartite** graph, compute/find a maximum matching.

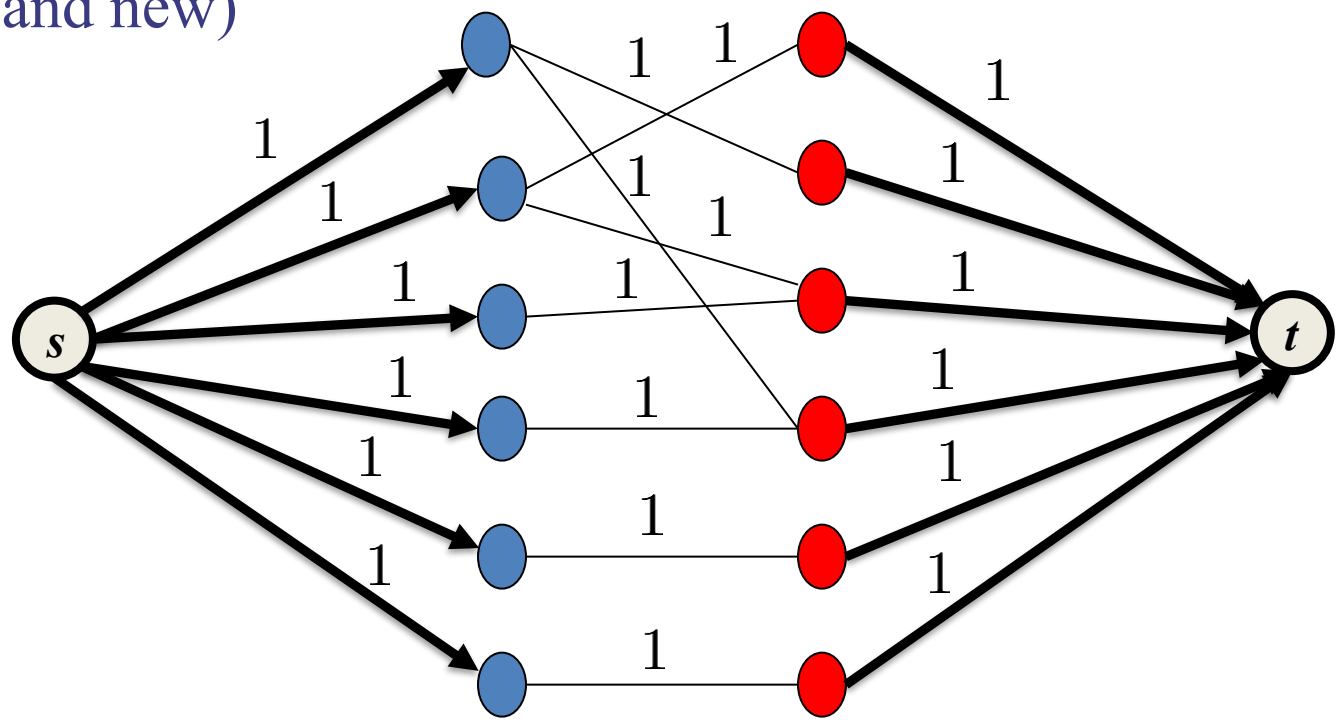
Idea: **Reduce** it to Maxflow problem. To do that, we need a network flow and s, t.



Case study 0: Maximum Matching

Problem: Given a **bipartite** graph, compute/find a maximum matching.

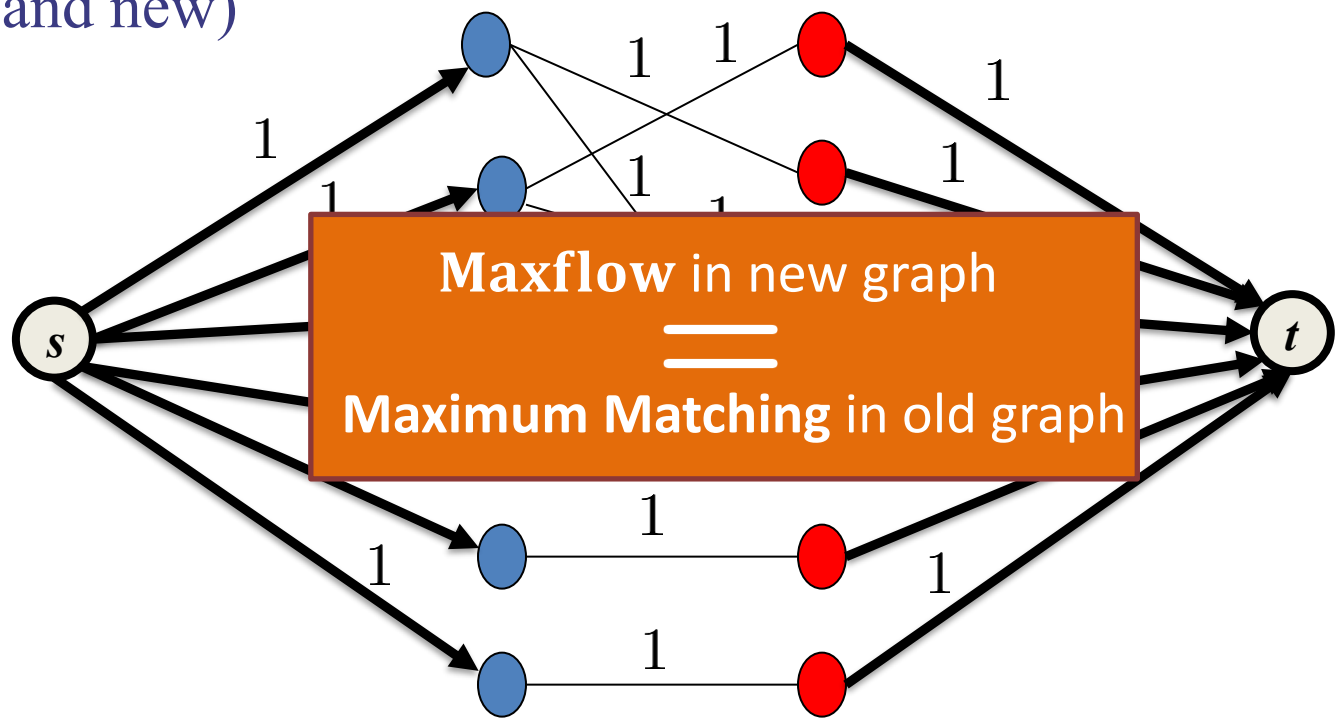
Idea: **Reduce** it to Maxflow problem. To do that, we need a network flow and s, t. Put capacity one for all edges (old and new)



Case study 0: Maximum Matching

Problem: Given a **bipartite** graph, compute/find a maximum matching.

Idea: **Reduce** it to Maxflow problem. To do that, we need a network flow and s, t. Put capacity one for all edges (old and new)

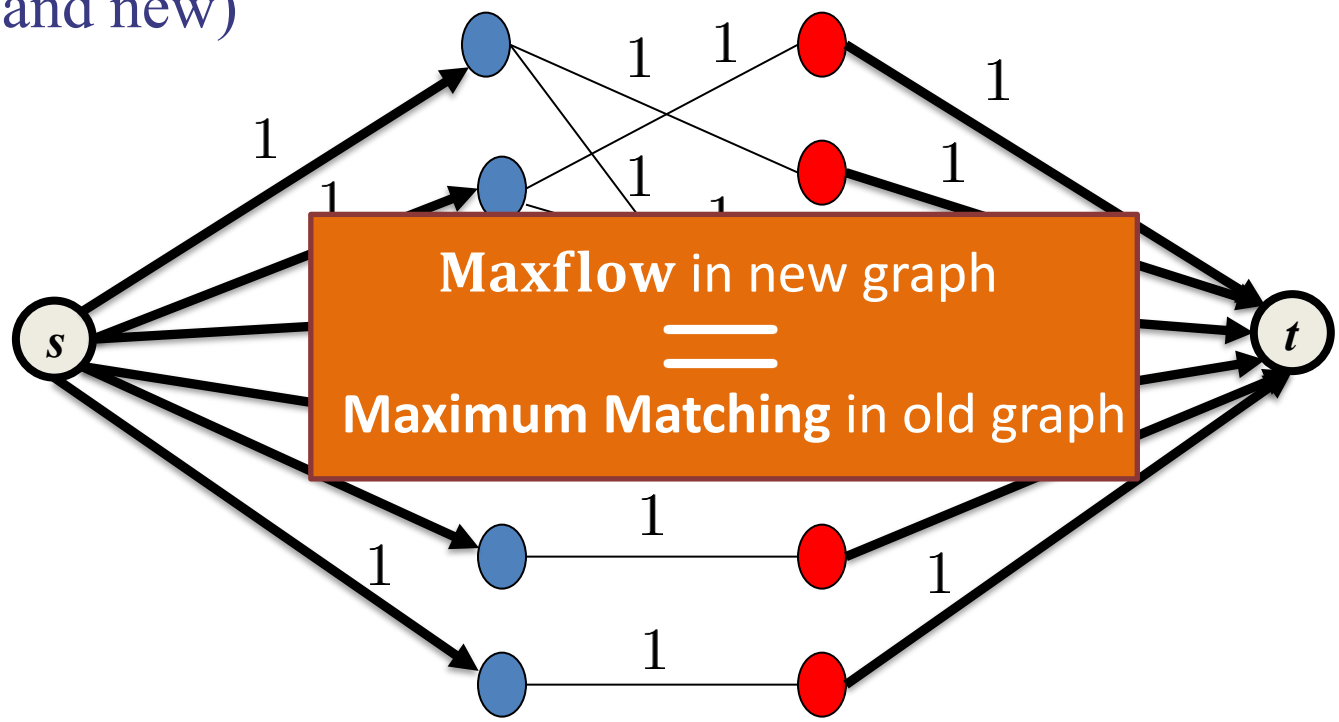


Case study 0: Maximum Matching

Problem: Given a **bipartite** graph, compute/find a maximum matching.

Idea: **Reduce** it to Maxflow problem. To do that, we need a network flow and ges (old and new)

Running time $O((V + E) \cdot V)$

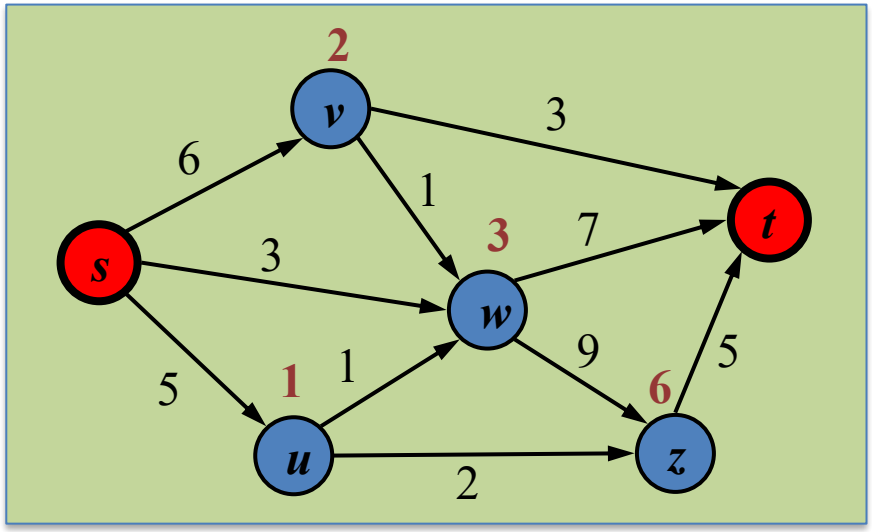


Case study I: Capacities on the vertices

Problem: Given a network G, a source s and a sink t, and capacities on the edges **and vertices**, compute the maximum possible flow value $|f^*|$.

$$\sum_{e \in \textbf{outgoing}(u)} f(e) = \sum_{e \in \textbf{incoming}(u)} f(e) \leq c(u)$$

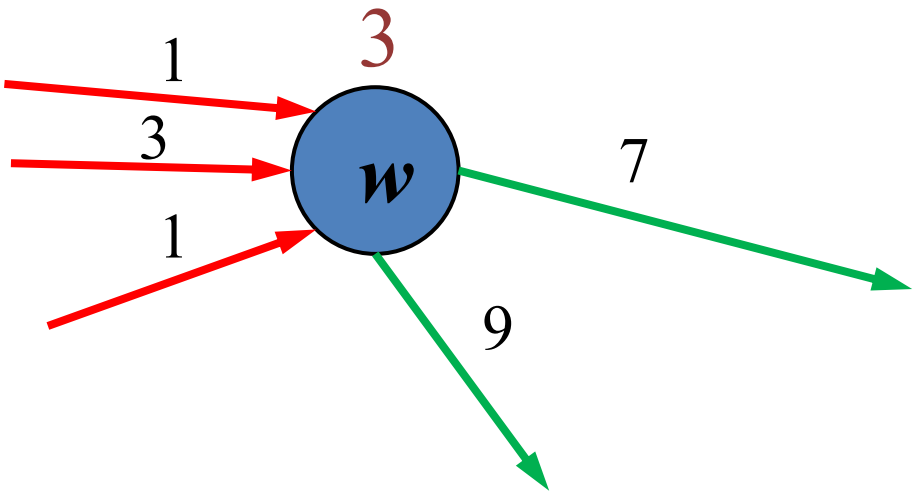
Example:



$$c(u) = 1$$
$$c(v) = 2$$
$$c(w) = 3$$
$$c(z) = 6$$

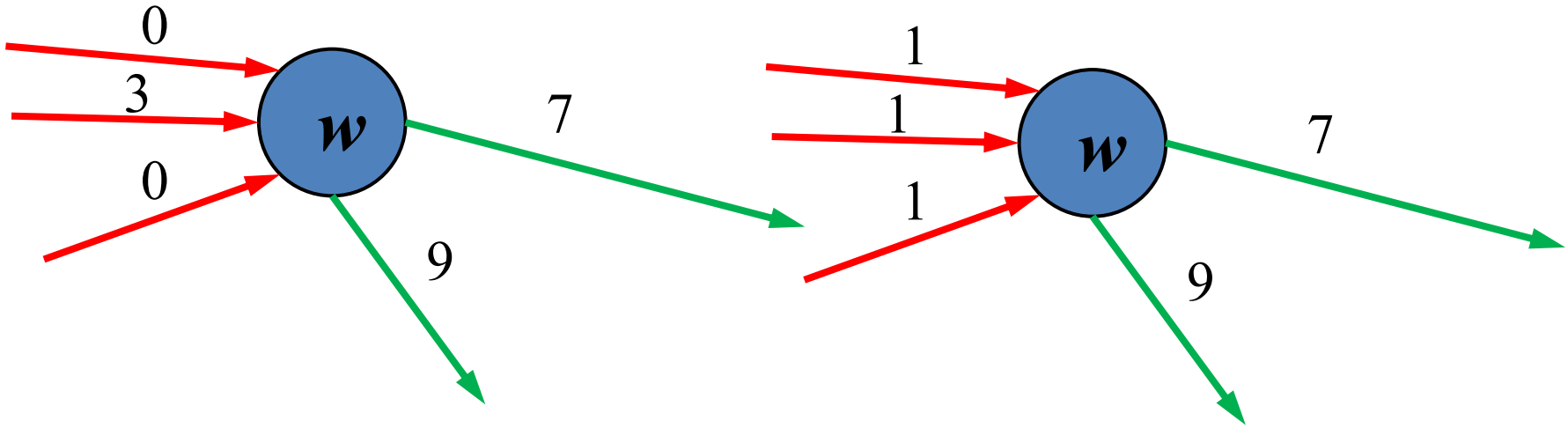
Case study I: Capacities on the vertices

Main idea: **Reduce** it to classic Maxflow problem. To do that, we need to remove the capacity constraints on vertices. How?



Case study I: Capacities on the vertices

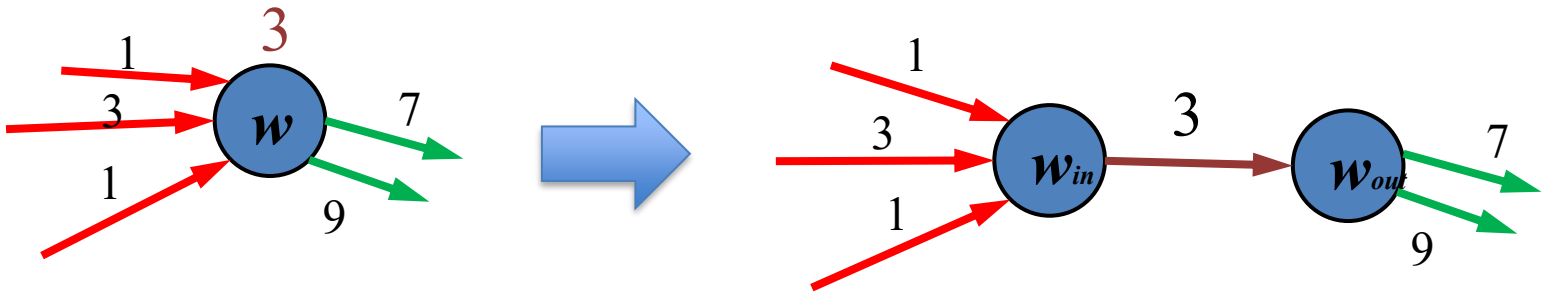
Main idea: **Reduce** it to classic Maxflow problem. To do that, we need to remove the capacity constraints on vertices. How?



Slow approach: Consider all possible capacities on incoming edges. Run Maxflow in all possible flow networks. Exponentially many.

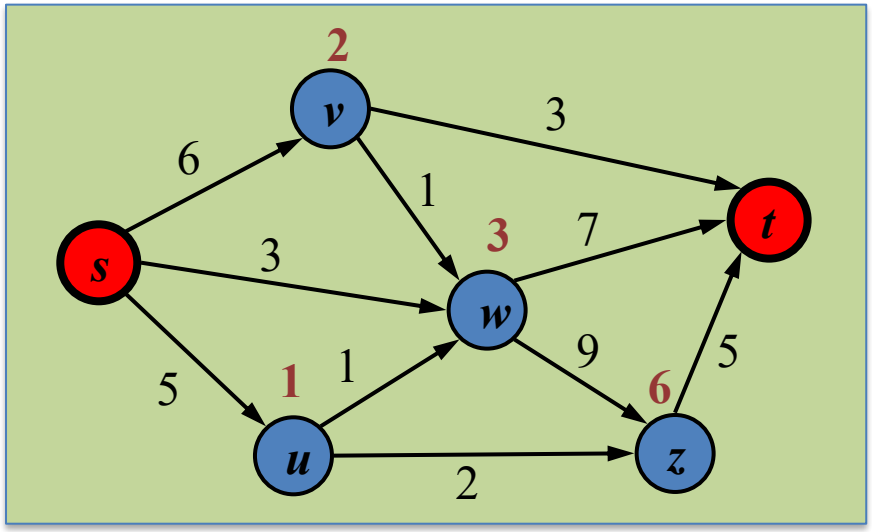
Case study I: Capacities on the vertices

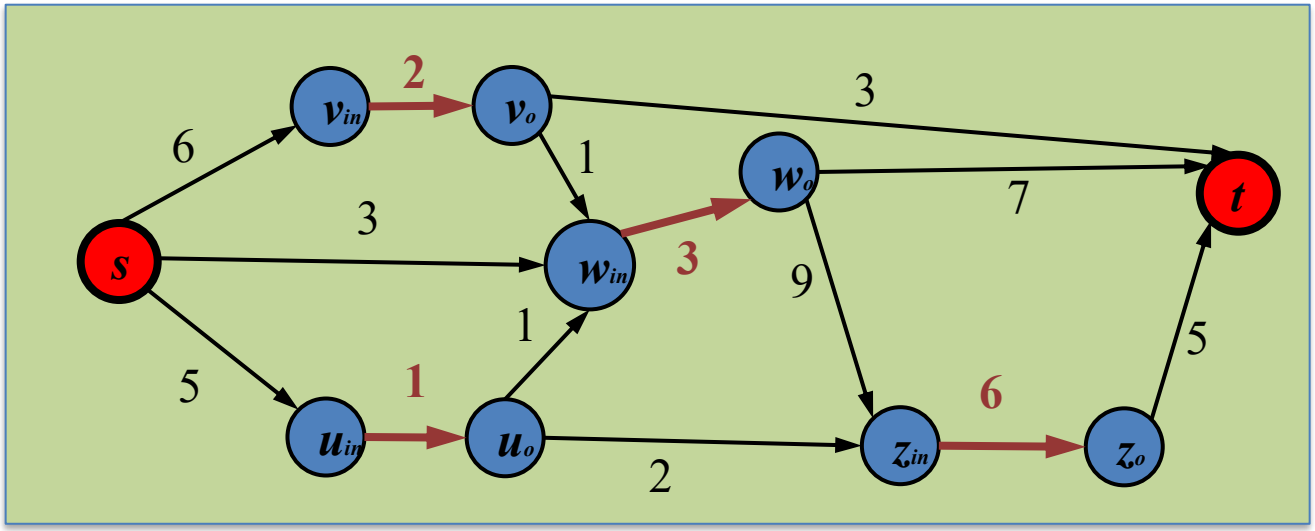
Main idea: **Reduce** it to classic Maxflow problem. To do that, we need to remove the capacity constraints on vertices. How?



Fast approach: Transform the capacity on vertices to capacities on edges. For every vertex w, create two new vertices w_{in}, w_{out} and connect all incoming edges of w to w_{in} and all outgoing edges to w_{out}.

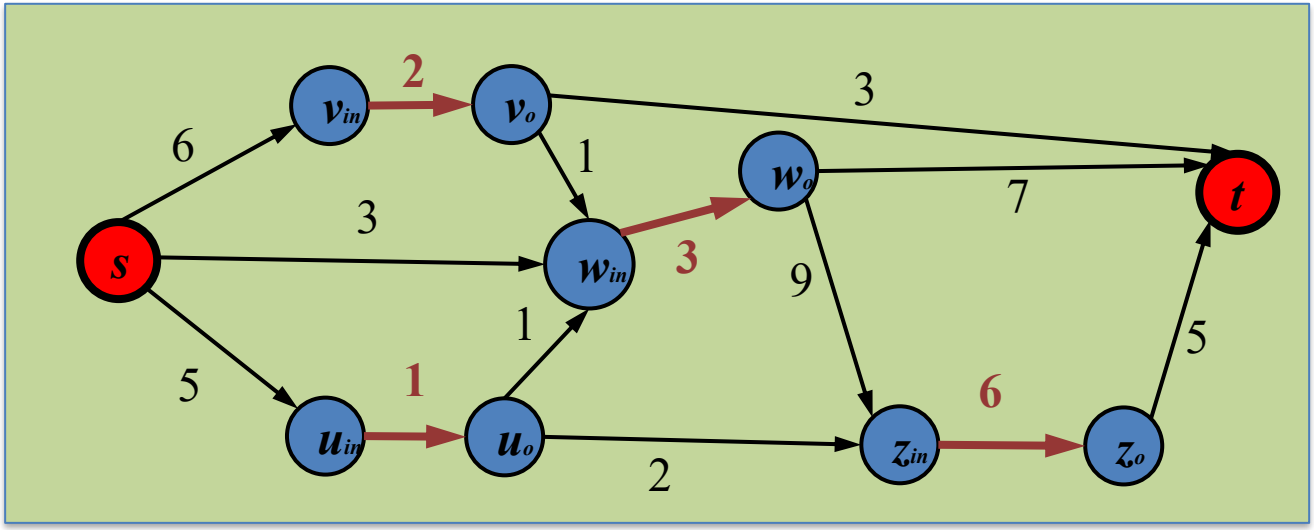
Case study I: Capacities on the vertices





Case study I: Capacities on the vertices

Running time: $\Theta(V + E)$ to create the new graph
+ Ford-Fulkerson (or Edmonds-Karp).



Another Question: What if you have multiple sources and/or sinks?

Case study II: Baseball Elimination

Problem: Let T be a set of teams in baseball league. At any point during the season, each team, i will have w_i wins, and will have g_i games left to play. The task is to determine whether it is possible for team i to finish the season in first place, given the games it has already won and the games it has left to play. Note that this depends on the games left for team i but also depends on the respective schedules of the other teams.

Example:

Team i	Wins w_i	Games Left g_i	Schedule $(g_{i,j})$			
			LA	Oak	Sea	Tex
Los Angeles	81	8	-	1	6	1
Oakland	77	4	1	-	0	3
Seattle	76	7	6	0	-	1
Texas	74	5	1	3	1	-

Case study II: Baseball Elimination

Problem: Let T be a set of teams in baseball league. At any point during the season, each team, i will have w_i wins, and will have g_i games left to play. The task is to determine whether it is possible for team i to finish the season in first place, given the games it has already won and the games it has left to play. Note that this depends on the games left for team i but also depends on the respective schedules of the other teams.

Example:

Team i	Wins w_i	Games Left g_i	Schedule $(g_{i,j})$			
			LA	Oak	Sea	Tex
Los Angeles	81	8	-	1	6	1
Oakland	77	4	1	-	0	3
Seattle	76	7	6	0	-	1
Texas	74	5	1	3	1	-

Question: Can Texas finish first? What about Oakland?

Case study II: Baseball Elimination

Example:

Team i	Wins w_i	Games Left g_i	Schedule $(g_{i,j})$			
			LA	Oak	Sea	Tex
Los Angeles	81	8	-	1	6	1
Oakland	77	4	1	-	0	3
Seattle	76	7	6	0	-	1
Texas	74	5	1	3	1	-

Question: Can Texas finish first? What about Oakland?

Answer: For Texas the answer is NO. It can win at most 79 wins.
For Oakland is also NO. Either LA or Seattle will definitely reach 82 wins.

Case study II: Baseball Elimination

Example:

Team i	Wins w_i	Games Left g_i	Schedule $(g_{i,j})$			
			LA	Oak	Sea	Tex
Los Angeles	81	8	-	1	6	1
Oakland	77	4	1	-	0	3
Seattle	76	7	6	0	-	1
Texas	74	5	1	3	1	-

Focus on team k.

Maximum possible number of wins is $W = w_k + g_k$.

Case study II: Baseball Elimination

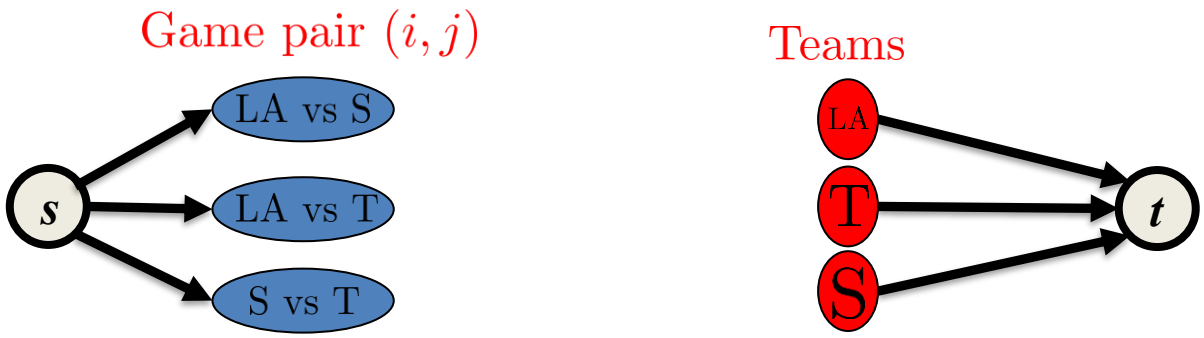
Example:

Team i	Wins w_i	Games Left g_i	Schedule $(g_{i,j})$			
			LA	Oak	Sea	Tex
Los Angeles	81	8	-	1	6	1
Oakland	77	4	1	-	0	3
Seattle	76	7	6	0	-	1
Texas	74	5	1	3	1	-

Focus on team k.

Maximum possible number of wins is $W = w_k + g_k$.

If there exists a team i so that $w_i > W$ then k is eliminated.



Case study II: Baseball Elimination

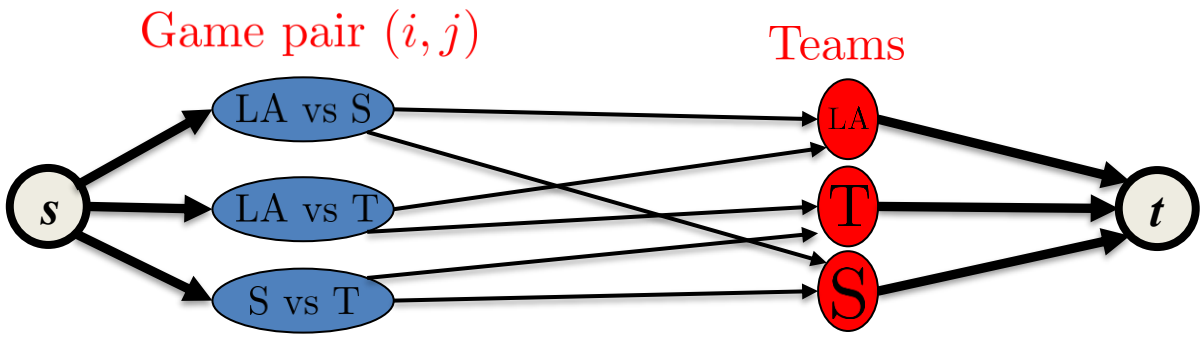
Example:

Team i	Wins w_i	Games Left g_i	Schedule $(g_{i,j})$			
			LA	Oak	Sea	Tex
Los Angeles	81	8	-	1	6	1
Oakland	77	4	1	-	0	3
Seattle	76	7	6	0	-	1
Texas	74	5	1	3	1	-

Focus on team k.

Maximum possible number of wins is $W = w_k + g_k$.

If there exists a team i so that $w_i > W$ then k is eliminated.



Case study II: Baseball Elimination

Example:

Team i	Wins w_i	Games Left g_i	Schedule $(g_{i,j})$			
			LA	Oak	Sea	Tex
Los Angeles	81	8	-	1	6	1
Oakland	77	4	1	-	0	3
Seattle	76	7	6	0	-	1
Texas	74	5	1	3	1	-

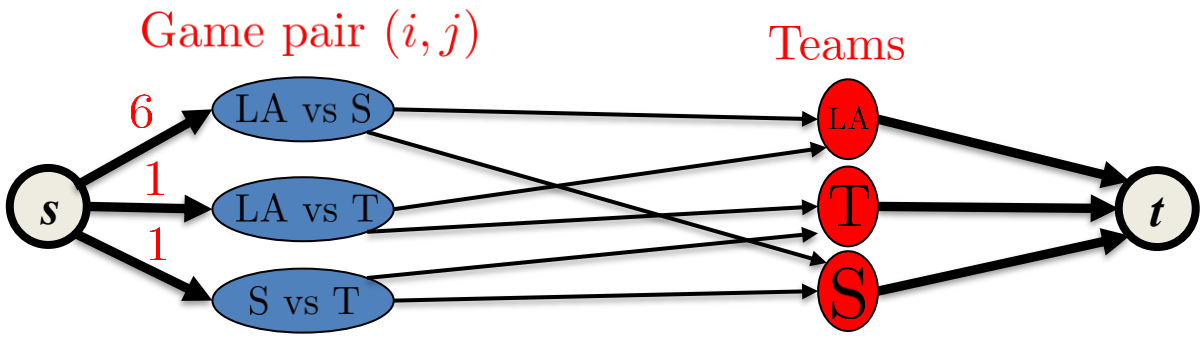
Focus on team k.

Maximum possible number of wins is $W = w_k + g_k$.

If there exists a team i so that $w_i > W$ then k is eliminated.

$k =$ Oakland.

$W = 81$.



Case study II: Baseball Elimination

Example:

Team i	Wins w_i	Games Left g_i	Schedule $(g_{i,j})$			
			LA	Oak	Sea	Tex
Los Angeles	81	8	-	1	6	1
Oakland	77	4	1	-	0	3
Seattle	76	7	6	0	-	1
Texas	74	5	1	3	1	-

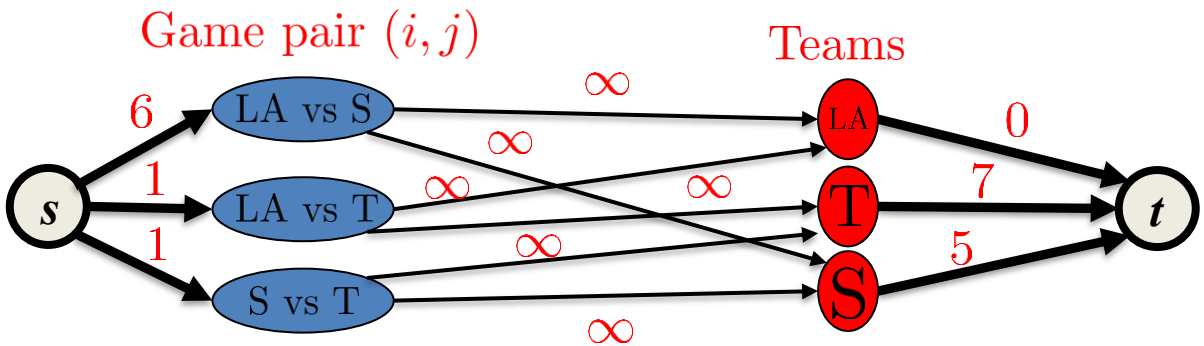
Focus on team k.

Maximum possible number of wins is $W = w_k + g_k$.

If there exists a team i so that $w_i > W$ then k is eliminated.

$k =$ Oakland.

$W = 81$.



Case study II: Baseball Elimination

Example:

Team i	Wins w_i	Games Left g_i	Schedule $(g_{i,j})$			
			LA	Oak	Sea	Tex
Los Angeles	81	8	-	1	6	1
Oakland	77	4	1	-	0	3
Seattle	76	7	6	0	-	1
Texas	74	5	1	3	1	-

For team i, capacity is $W - w_i$

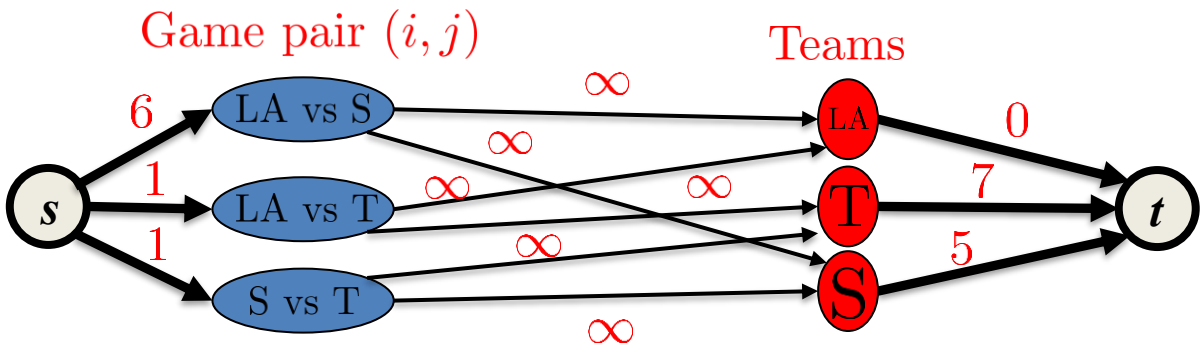
Focus on team k.

Maximum possible number of wins is $W = w_k + g_k$.

If there exists a team i so that $w_i > W$ then k is eliminated.

$k =$ Oakland.

$W = 81$.



Case study II: Baseball Elimination

Intuition: Suppose k wins all its games (W wins in total). All possible flows show how the rest of the games will proceed. Need to check if all games can be played without any other team exceeding W wins.

Case study II: Baseball Elimination

Intuition: Suppose k wins all its games (W wins in total). All possible flows show how the rest of the games will proceed. Need to check if all games can be played without any other team exceeding W wins.

Maxflow in graph
= # of remaining games (without k)
k can finish first.

Case study III: Vertex Cover on Bipartite Graphs

Problem (Practice): We use the term line of a matrix to mean either a row or a column. Given an $n \times n$ matrix of **0**'s and **1**'s, your task is to find a smallest number of lines such that every **1** entry of the matrix is contained in one of the selected lines.

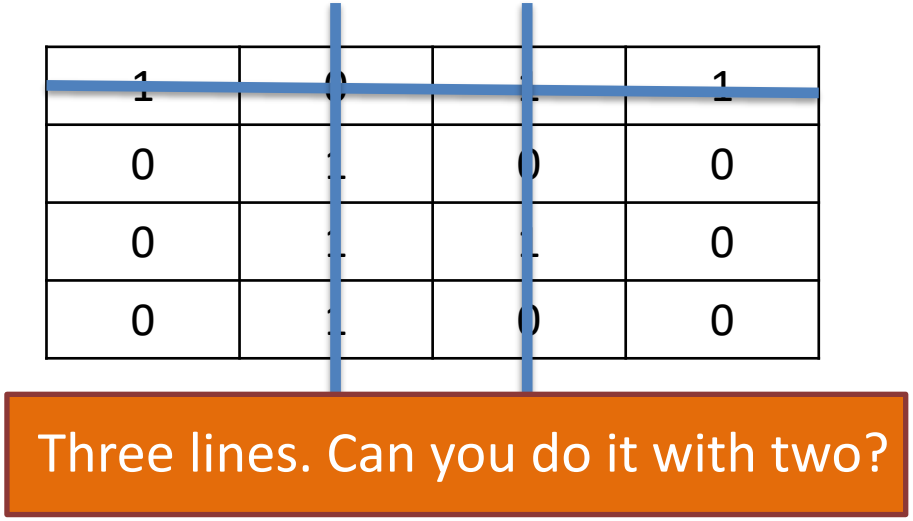
Example:

1	0	1	1
0	1	0	0
0	1	1	0
0	1	0	0

Case study III: Vertex Cover on Bipartite Graphs

Problem (Practice): We use the term line of a matrix to mean either a row or a column. Given an $n \times n$ matrix of **0**'s and **1**'s, your task is to find a smallest number of lines such that every **1** entry of the matrix is contained in one of the selected lines.

Example:



Case study III: Vertex Cover on Bipartite Graphs

Idea: Create a bipartite graph.

1	0	1	1
0	1	0	0
0	1	1	0
0	1	0	0

Case study III: Vertex Cover on Bipartite Graphs

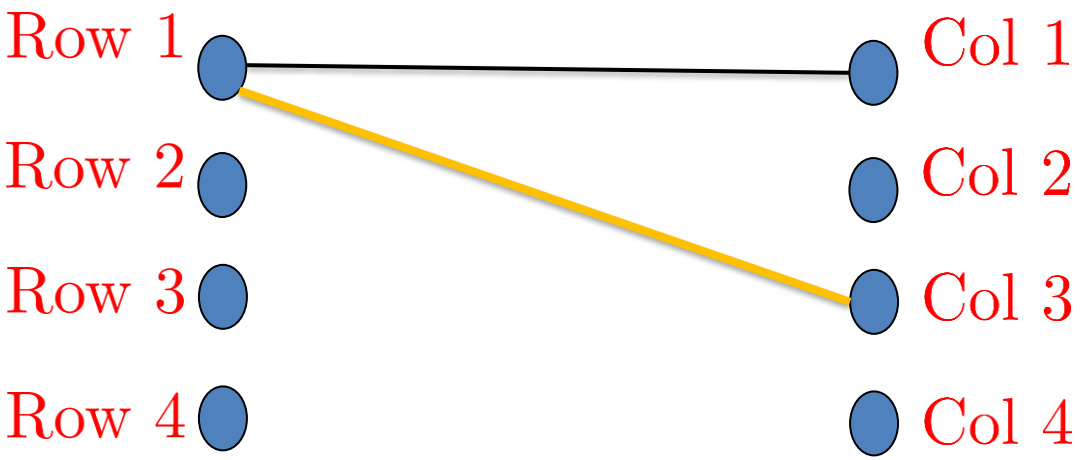
Idea: Create a bipartite graph.

1	0	1	1
0	1	0	0
0	1	1	0
0	1	0	0

Case study III: Vertex Cover on Bipartite Graphs

Idea: Create a bipartite graph.

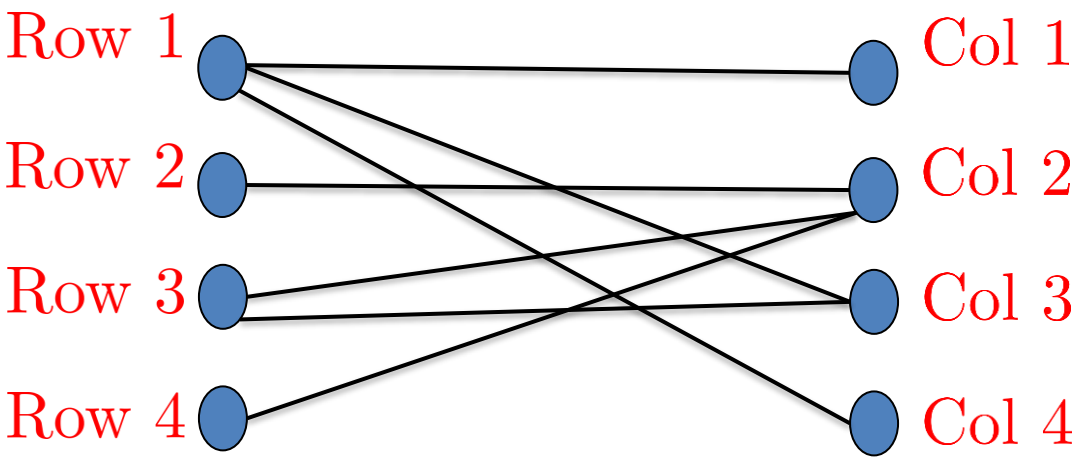
1	0	1	1
0	1	0	0
0	1	1	0
0	1	0	0



Case study III: Vertex Cover on Bipartite Graphs

Idea: Create a bipartite graph.

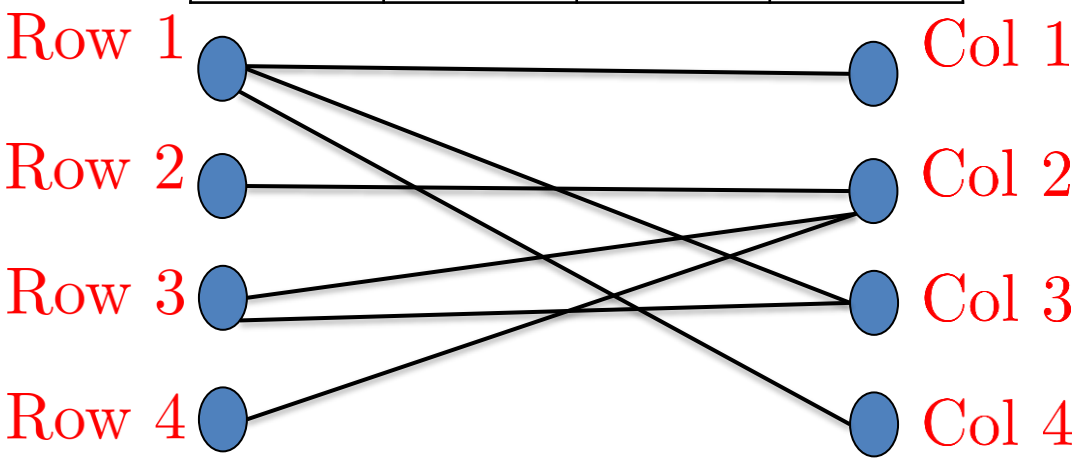
1	0	1	1
0	1	0	0
0	1	1	0
0	1	0	0



Case study III: Vertex Cover on Bipartite Graphs

Idea 2: Choose min number of vertices to cover all the edges (minimum vertex cover)

1	0	1	1
0	1	0	0
0	1	1	0
0	1	0	0

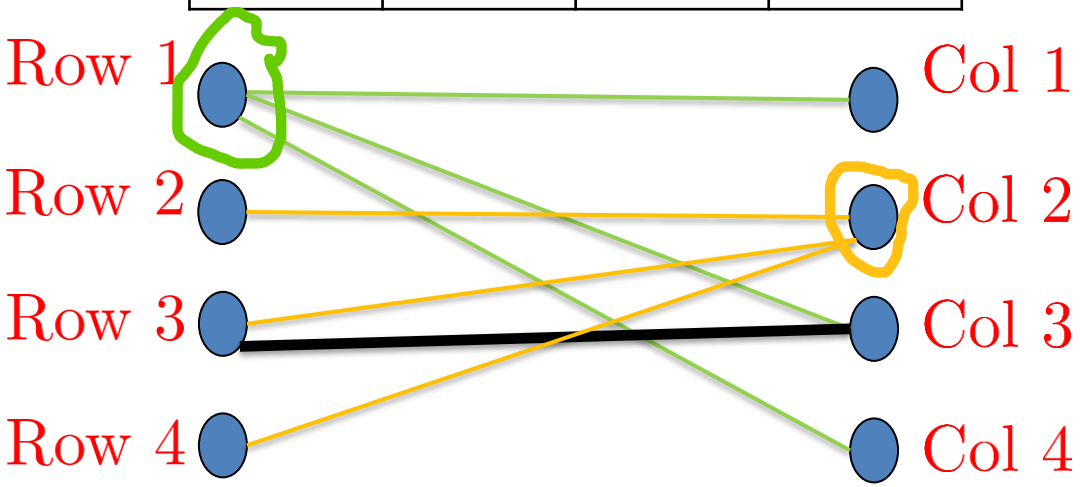


Case study III: Vertex Cover on Bipartite Graphs

Idea 2: Choose min number of vertices to cover all the edges (minimum vertex cover)

Not Vertex Cover

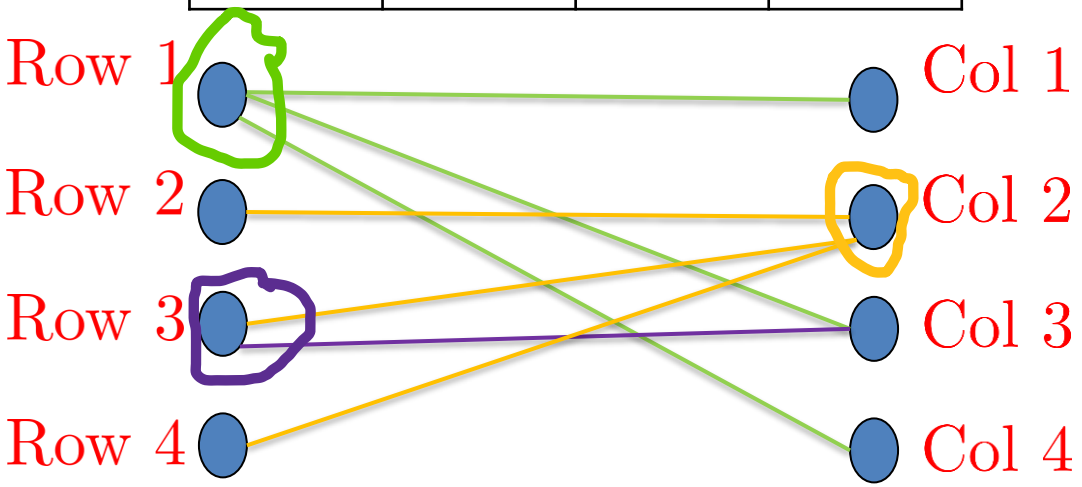
1	0	1	1
0	1	0	0
0	1	1	0
0	1	0	0



Case study III: Vertex Cover on Bipartite Graphs

Idea 2: Choose min number of vertices to cover all the edges (minimum vertex cover)

1	0	1	1
0	1	0	0
0	1	1	0
0	1	0	0



Case study III: Vertex Cover on Bipartite Graphs

Idea 3: Find a maximum Matching. Theorem (Konig): This is equal to min VC in bipartite graphs.

1	0	1	1
0	1	0	0
0	1	1	0
0	1	0	0