Maximum Matching in an Unweighted Bipartite Graph

Joy Mukherjee

Matching

- A matching in an undirected graph G = (V, E) is a subset of edges M of E such that the edges in M are pairwise disjoint, i.e., any two edges in M do not share any end vertices.

- A bipartite graph G = (V, E) is a graph, whose vertices can be divided into two partites X and Y such that and $X \cup Y = V, X \cap Y = \phi \; and \; \forall (u, v) \in E \; u \in X \leftrightarrow v \in Y$.

- Objective: Find a matching of maximum size in the bipartite graph

An Application

- J: Set of jobs
- C: Set of candidates
- L_j : List of jobs a candidate can do
- **Objective**: Maximize the number of job assignments
- **Constraints**:
  i. Each candidate can get at most one job
  ii. Each job is assigned to at most one candidate

Input / Output

- Input:

  A Bipartite Graph G=(X, Y, E), where X and Y are vertex sets and E is the set of edges from X to Y.

- Output:

  Matching of maximum possible size

- Matching:

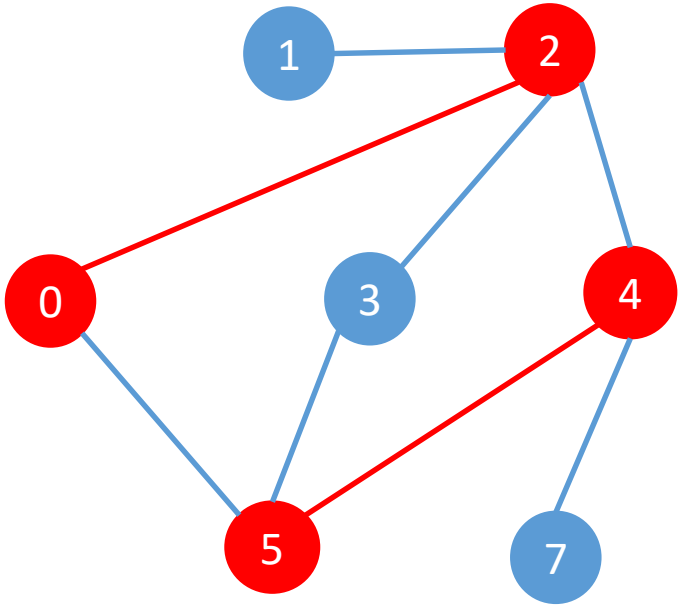
  A subset M of E edges in G is said to be a matching if at most one edge from M is incident on any vertex in X or Y.

Key Definitions

- A free (or unsaturated) vertex for a matching M is a vertex that is not an endpoint of any edge in M.
- An unsaturated edge for a matching M is an edge that is in E – M.
- An alternating path alternates between the edges in E-M and M.
- An augmenting path is an alternating path of odd length that starts and ends at a free vertex.

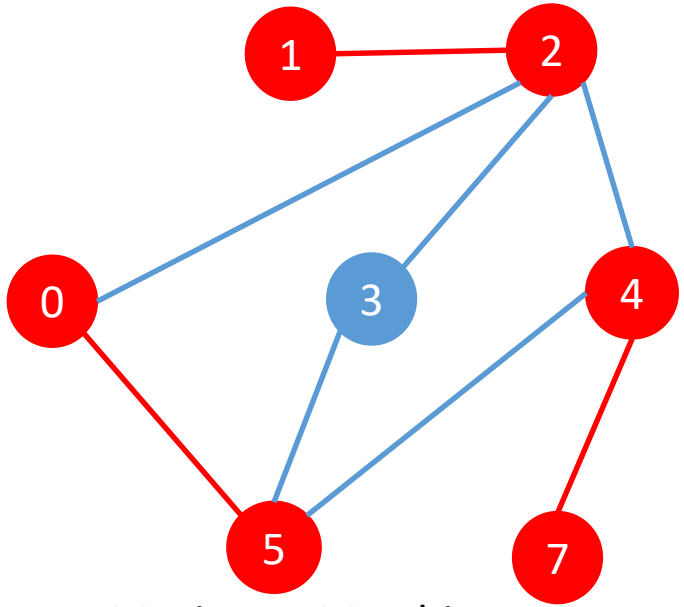
Edmond's Algorithm

1. M = {}
2. While(there is an augmenting path P for M)
   i. M = M XOR P
3. Report M



Maximal Matching

M = {(0, 2), (4, 5)}
P = {(7, 4), (4, 5), (5, 0), (0, 2), (2, 1)}
M = {(7, 4), (5, 0), (2, 1)}



Maximum Matching

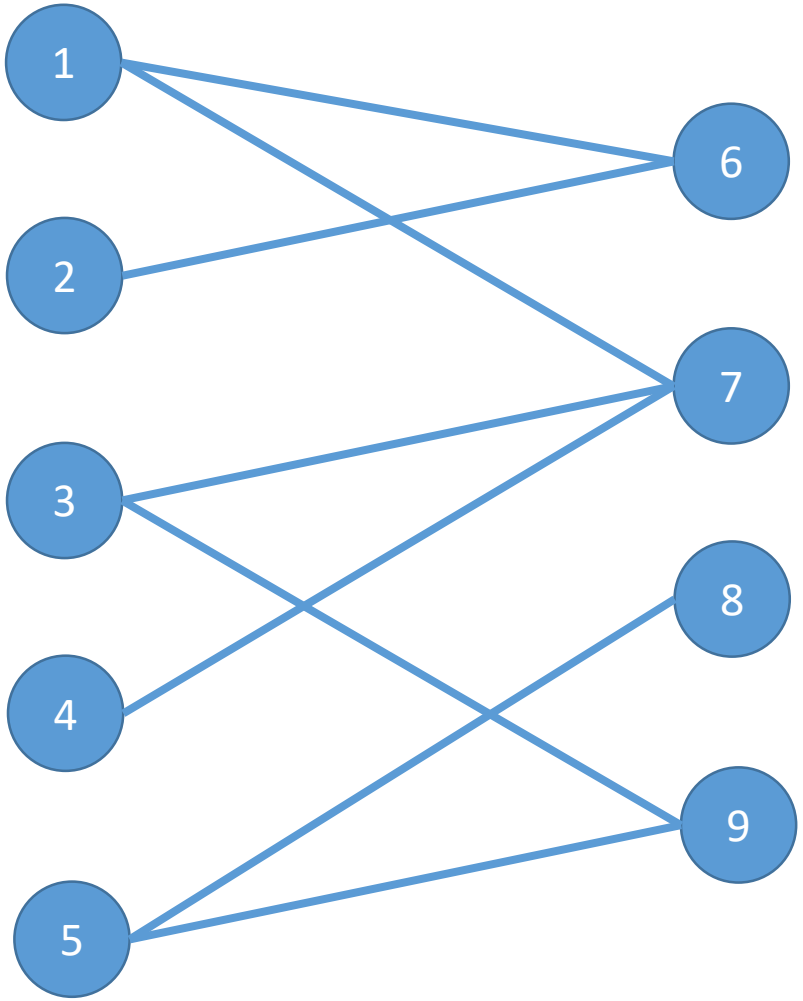
Edmond's Algorithm : Key Idea

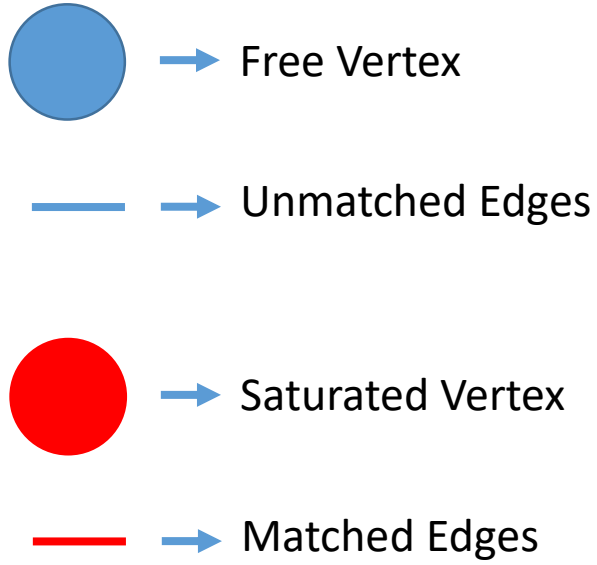
- Grow paths from all free vertices in X
- An augmenting path grows forward using edges in E-M (unsaturated edges X->Y), and grows backward using edges in M-E (saturated edges Y->X)
- If such a path ends at Y at a free vertex, an augmenting path is found.

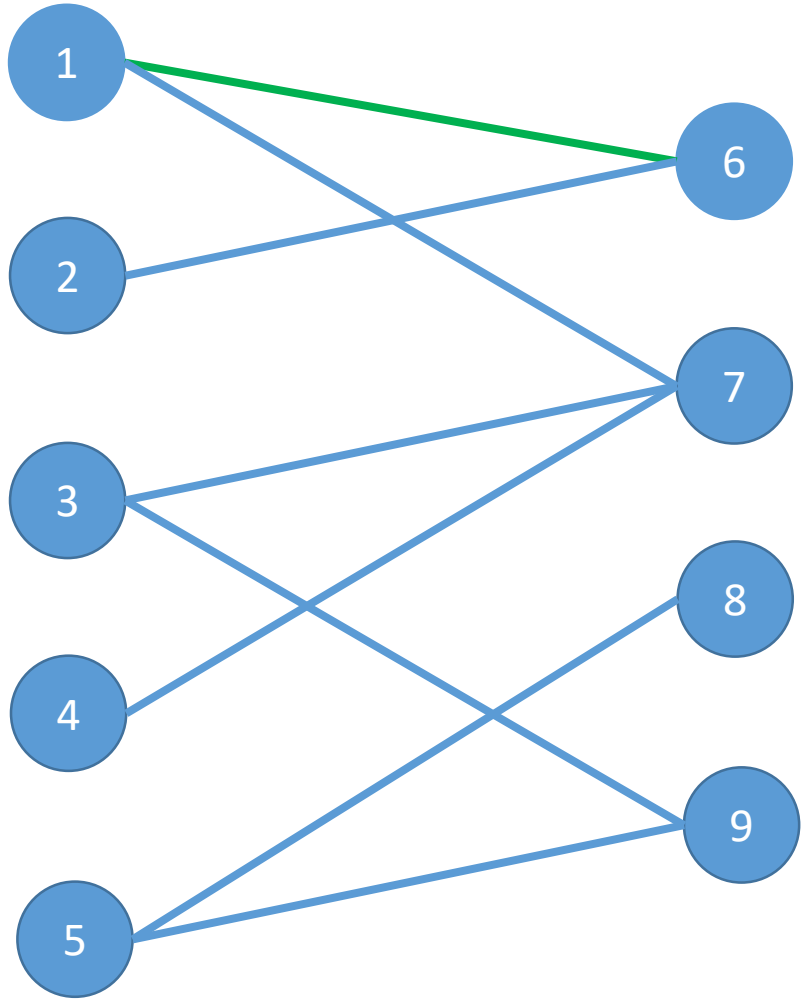
Edmond's Algorithm : Implementation

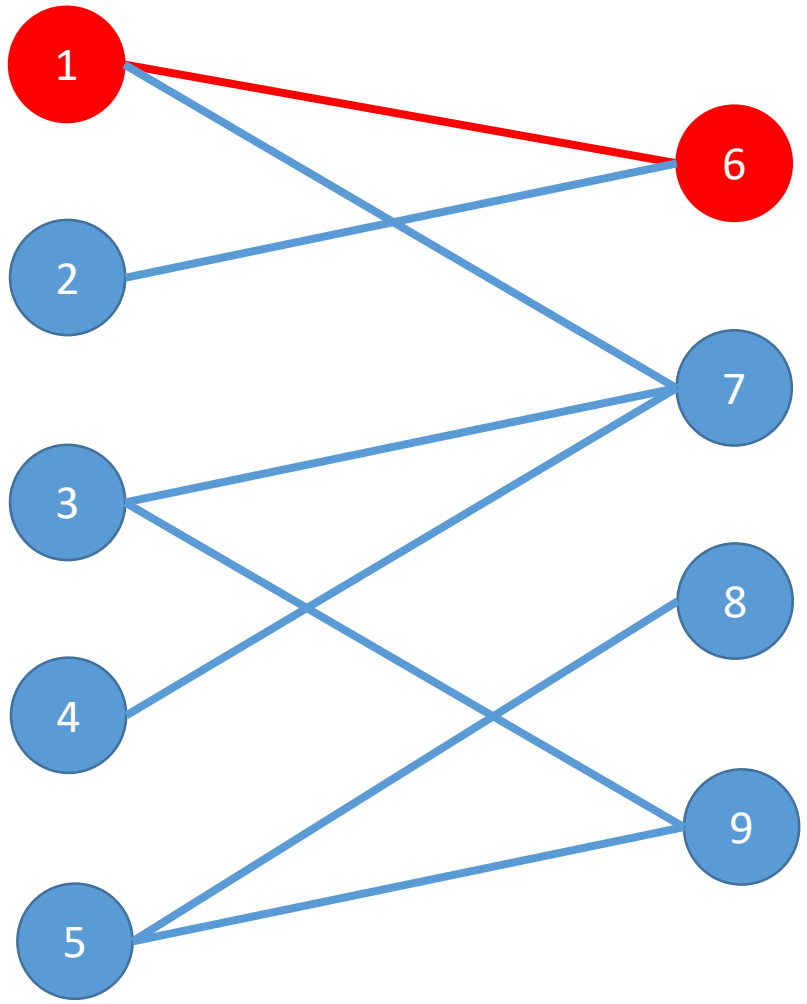
- We do a depth-first-search (DFS) from X and stop as soon as we reach some vertex in Y, thus giving an M-augmenting path P.
- We then augment M along P, and repeat the same process with the new matching M XOR P.
- If we cannot reach any vertex of Y, then M is maximum.

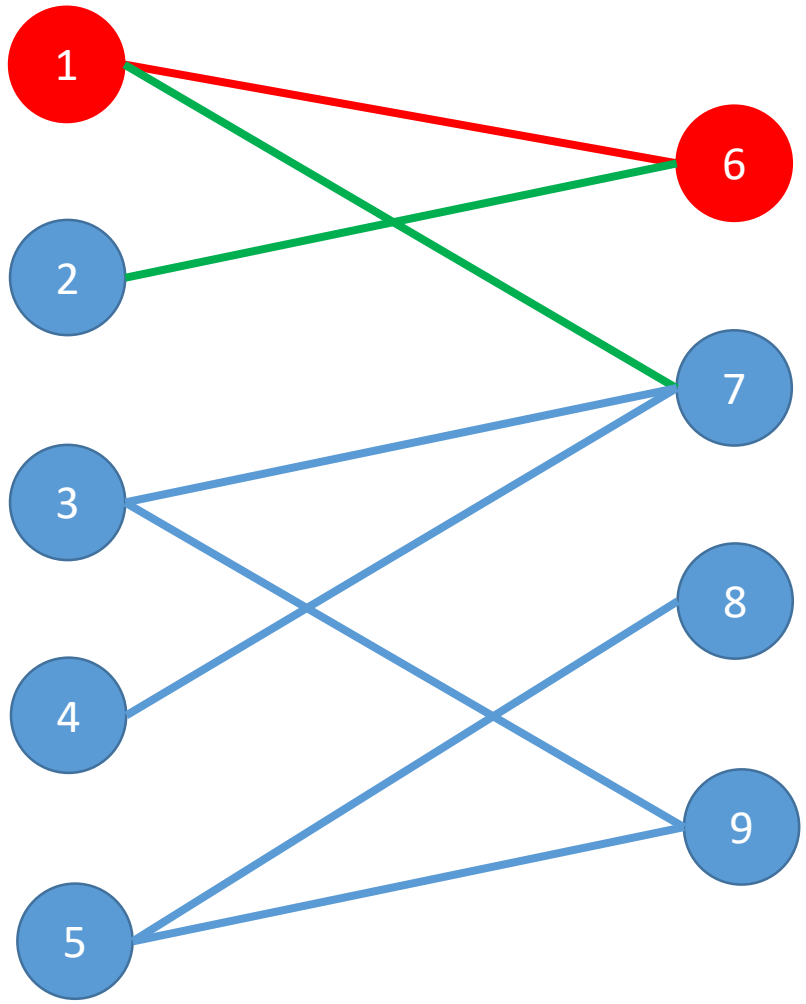
Example 1

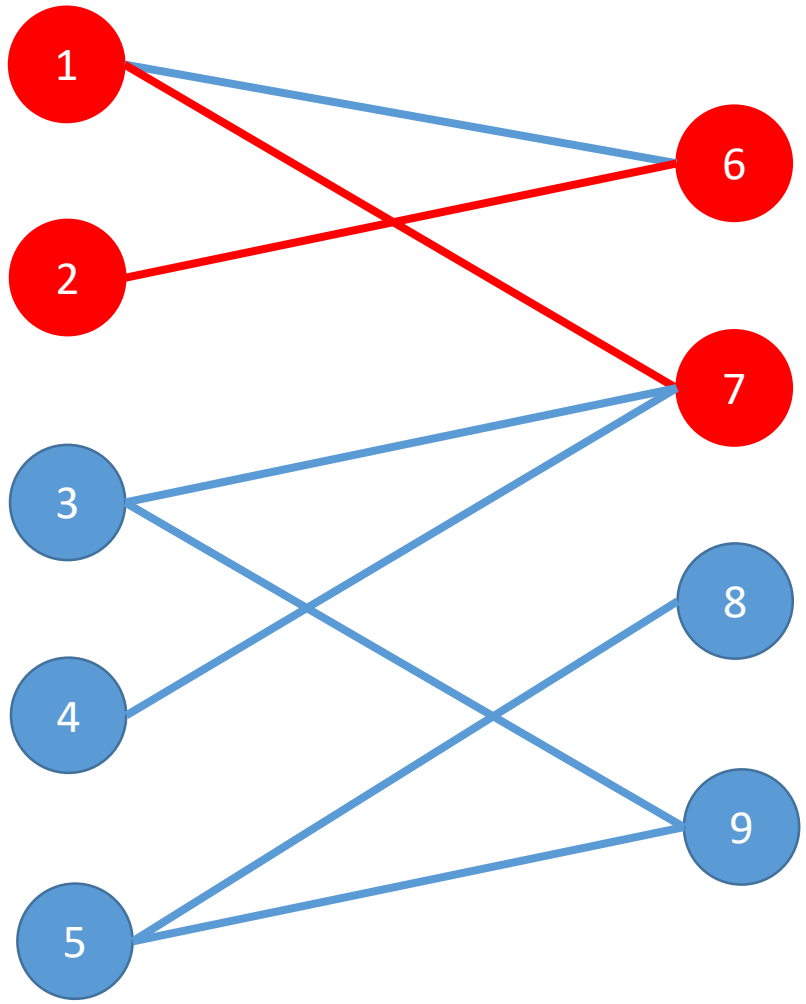

1
2
3
4
5
6
7
8
9

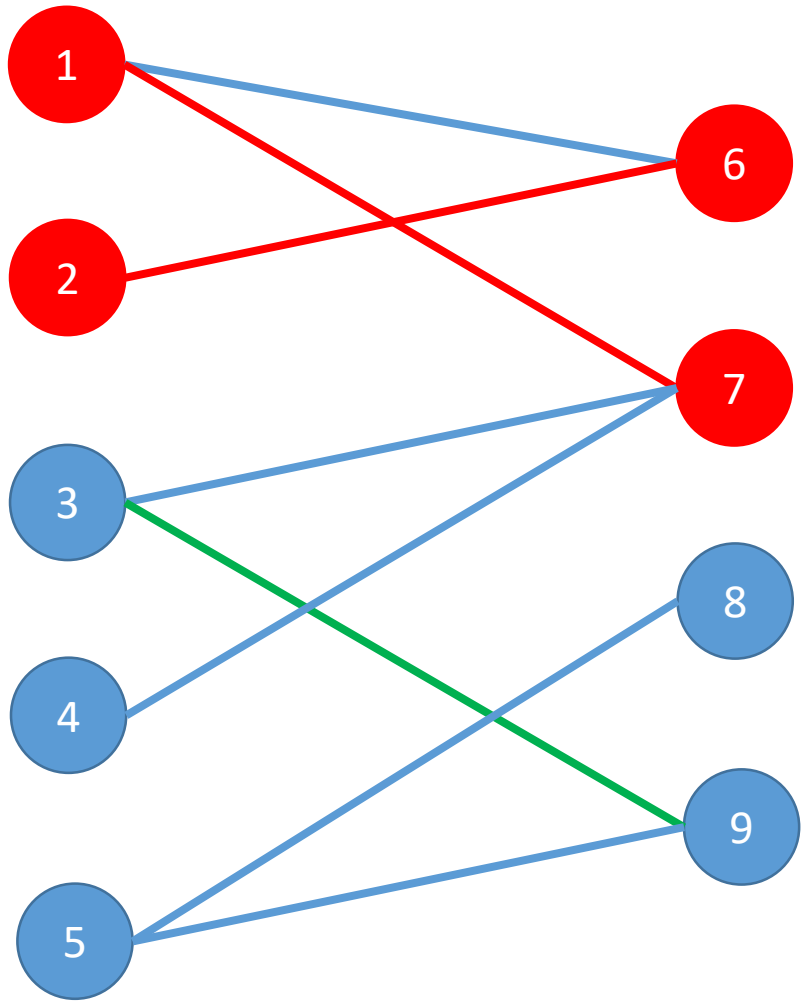

Free Vertex
Unmatched Edges
Saturated Vertex
Matched Edges

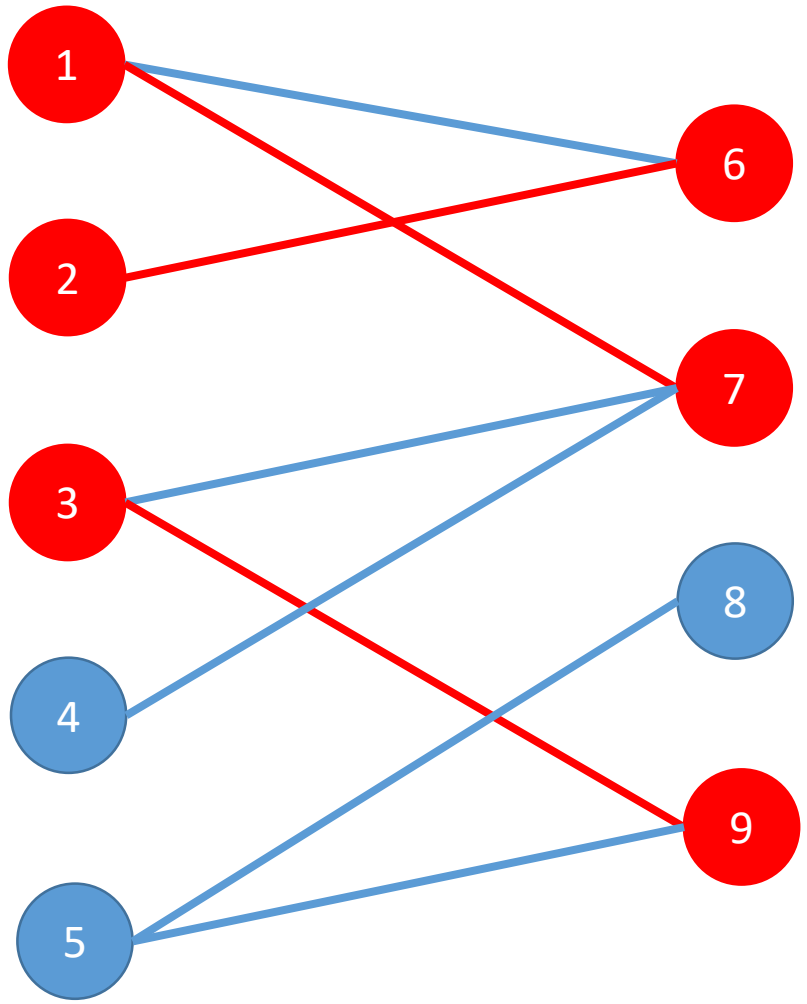

1
2
3
4
5
6
7
8
9

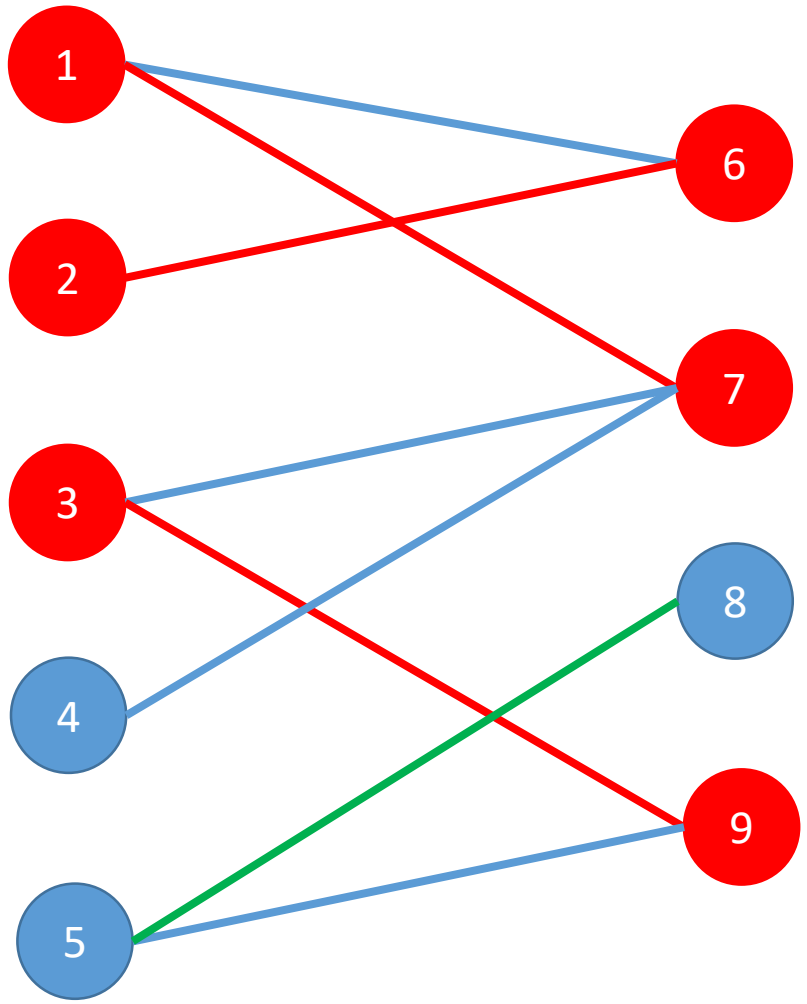

1
6
2
7
3
8
4
9
5

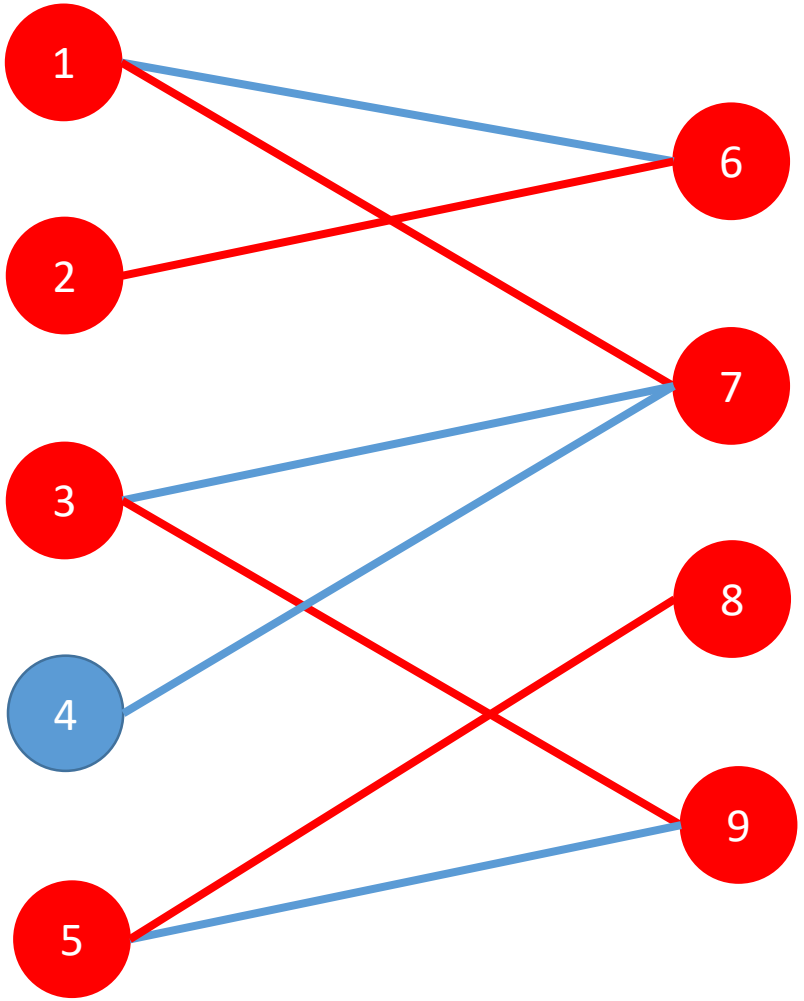

1
6
2
7
3
8
4
9
5


1
2
3
4
5
6
7
8
9

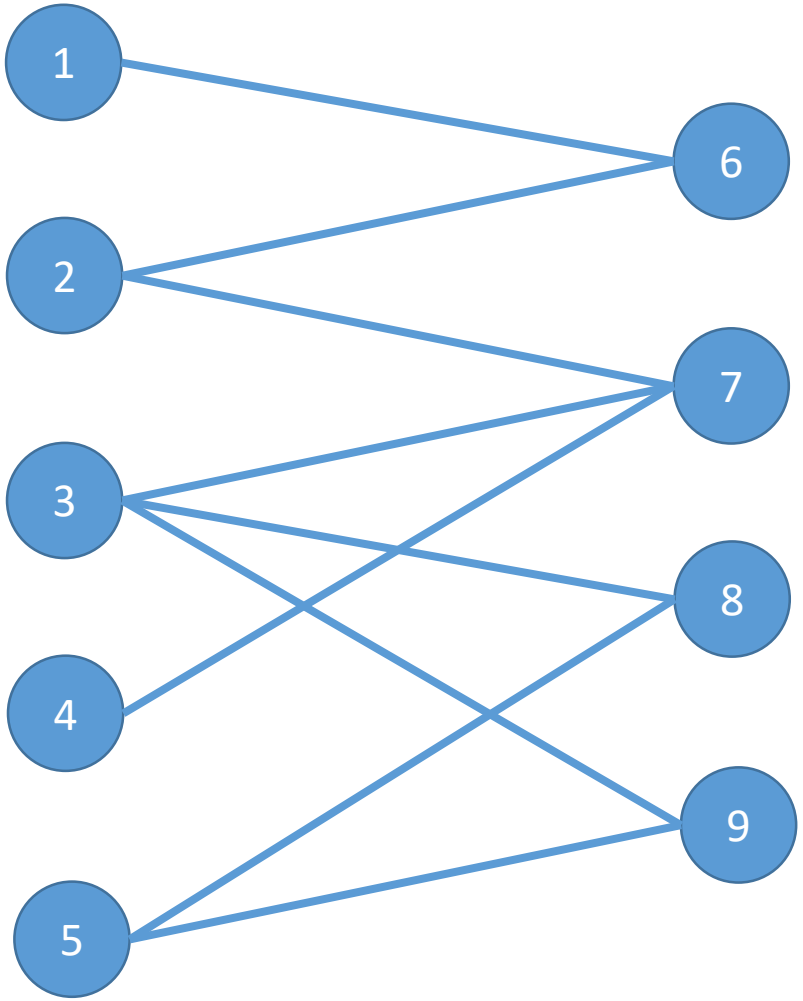

1
2
3
4
5
6
7
8
9

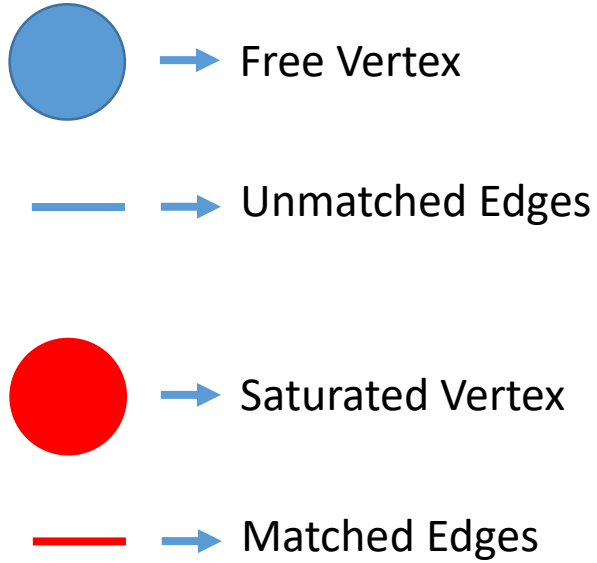

1
2
3
4
5
6
7
8
9

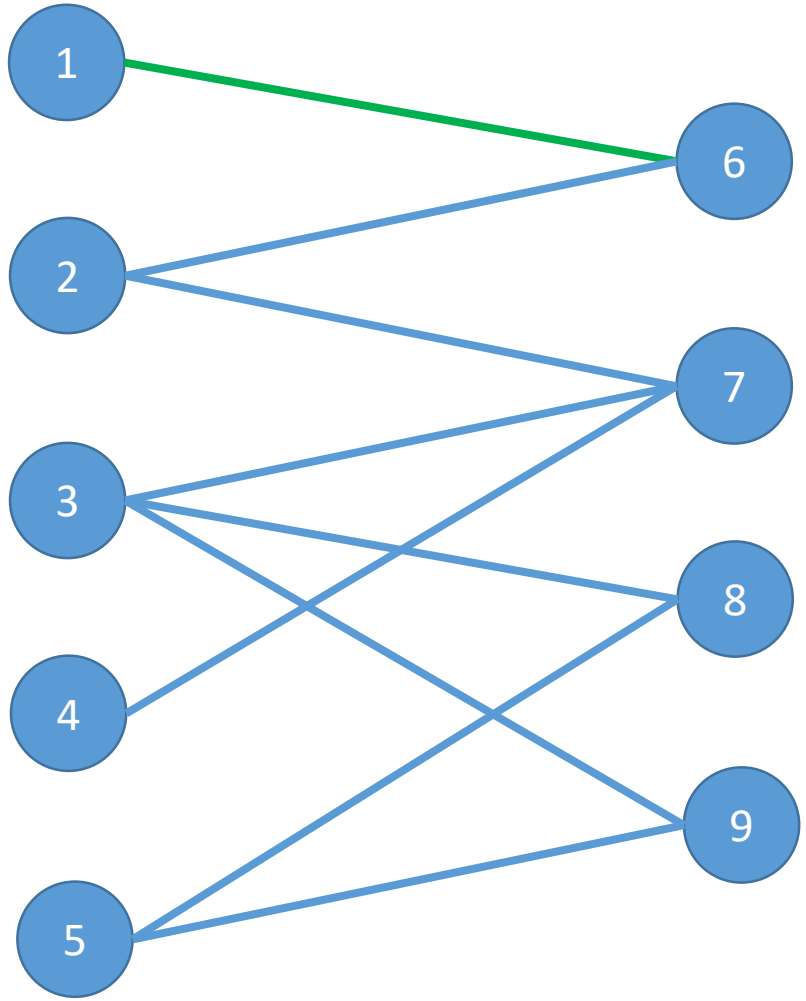

1
2
3
4
5
6
7
8
9

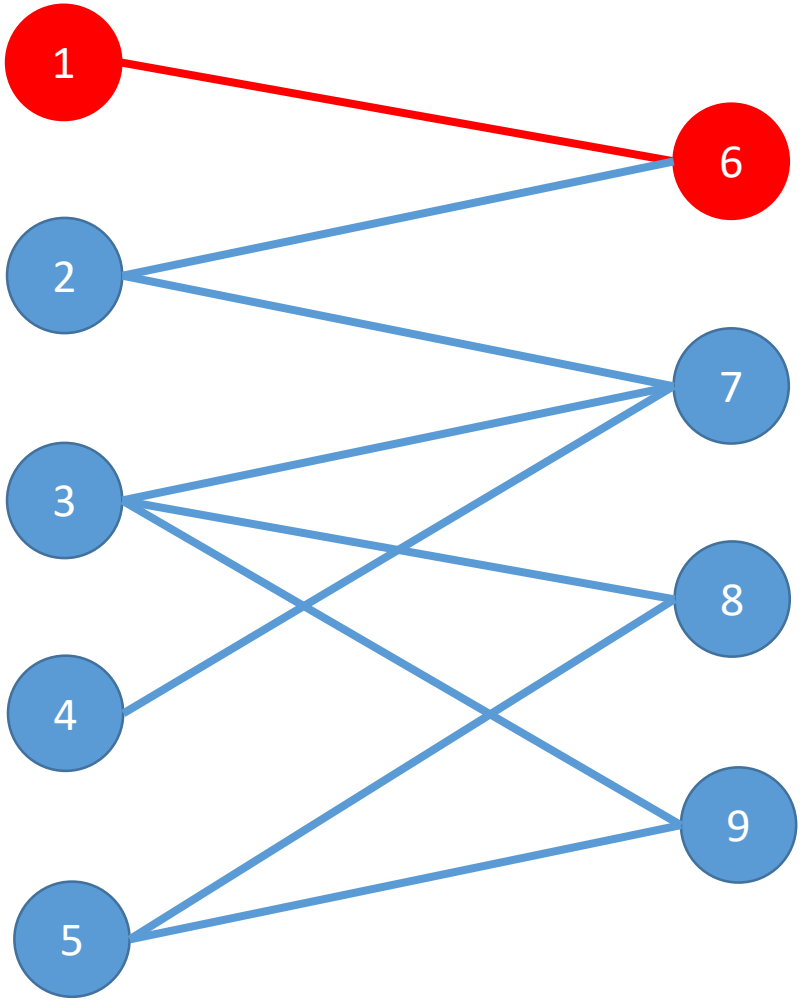

1
2
3
4
5
6
7
8
9

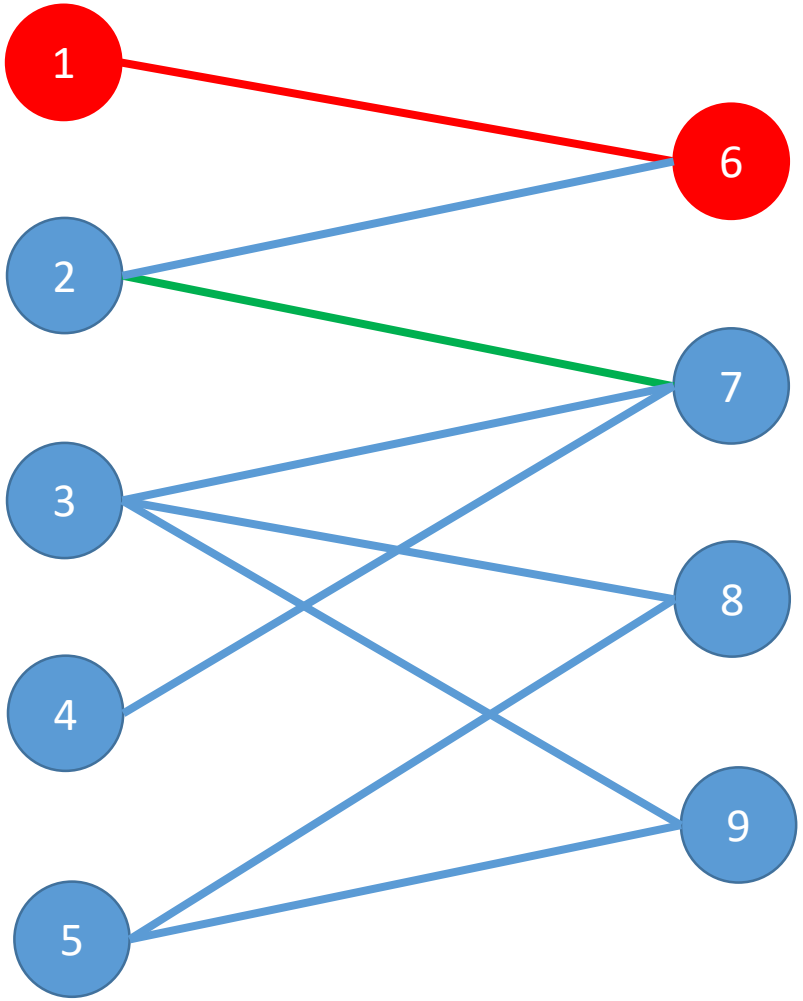
Example 2

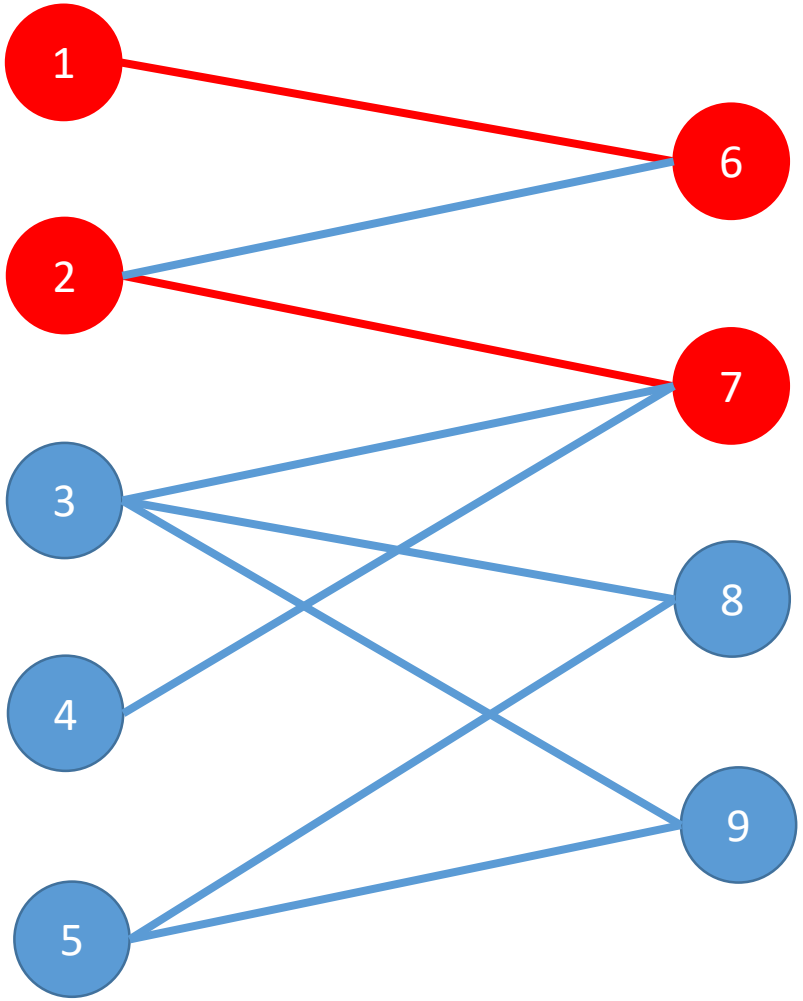

1
2
3
4
5
6
7
8
9

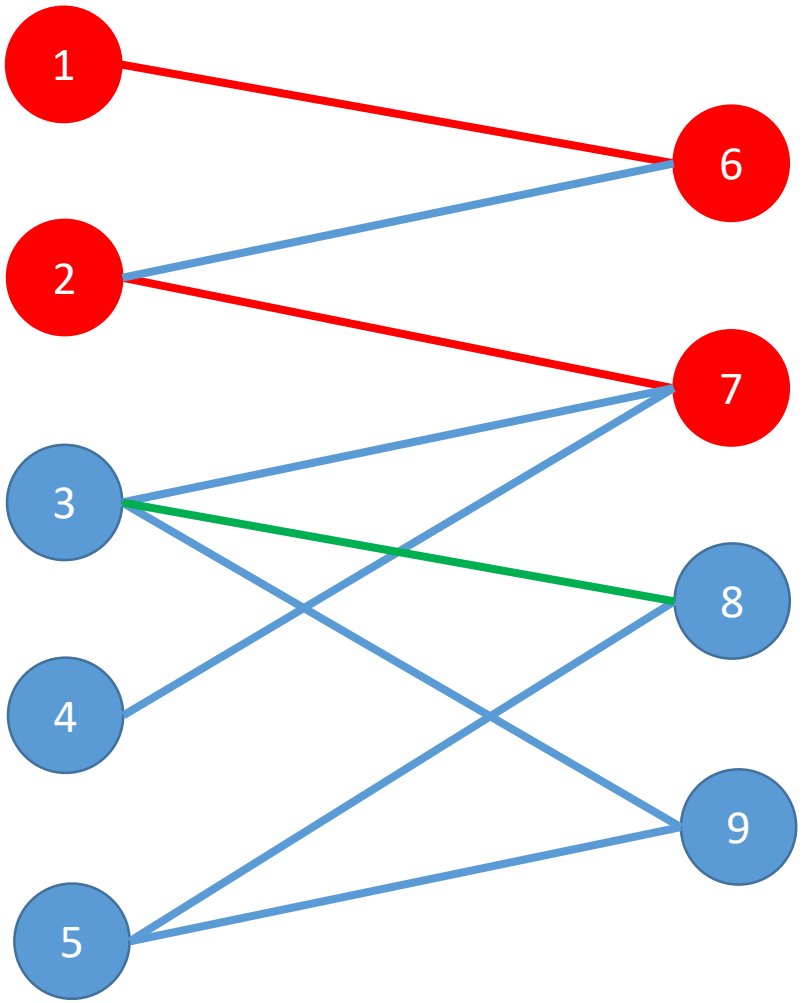

Free Vertex
Unmatched Edges
Saturated Vertex
Matched Edges

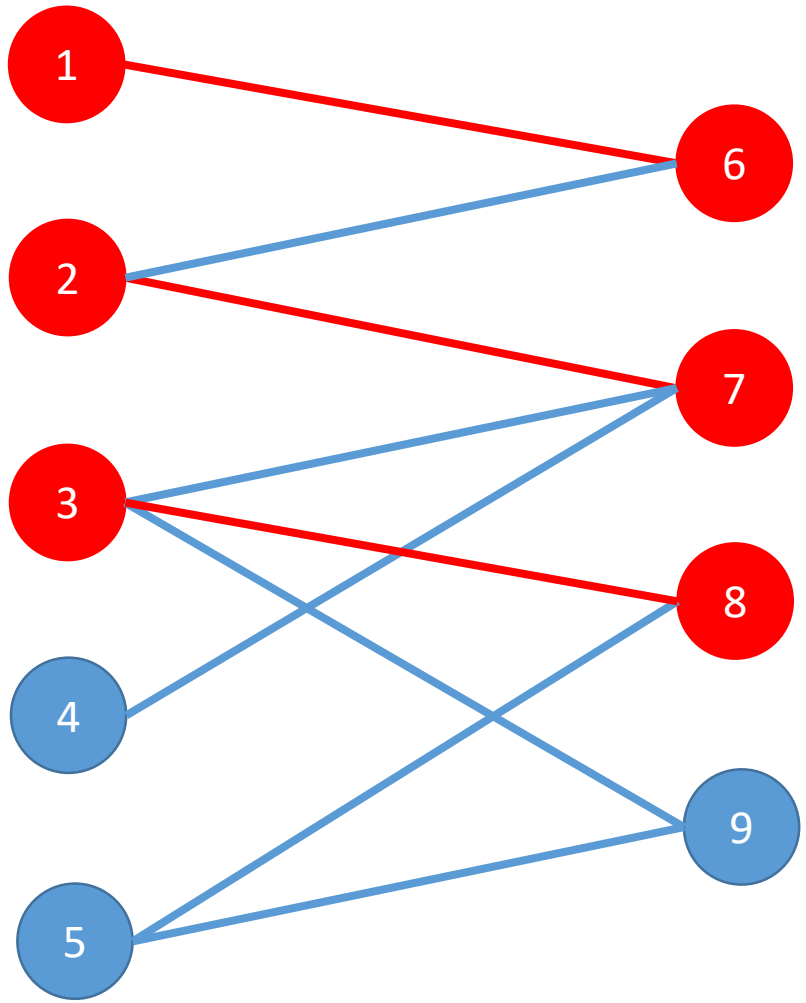

1
2
3
4
5
6
7
8
9

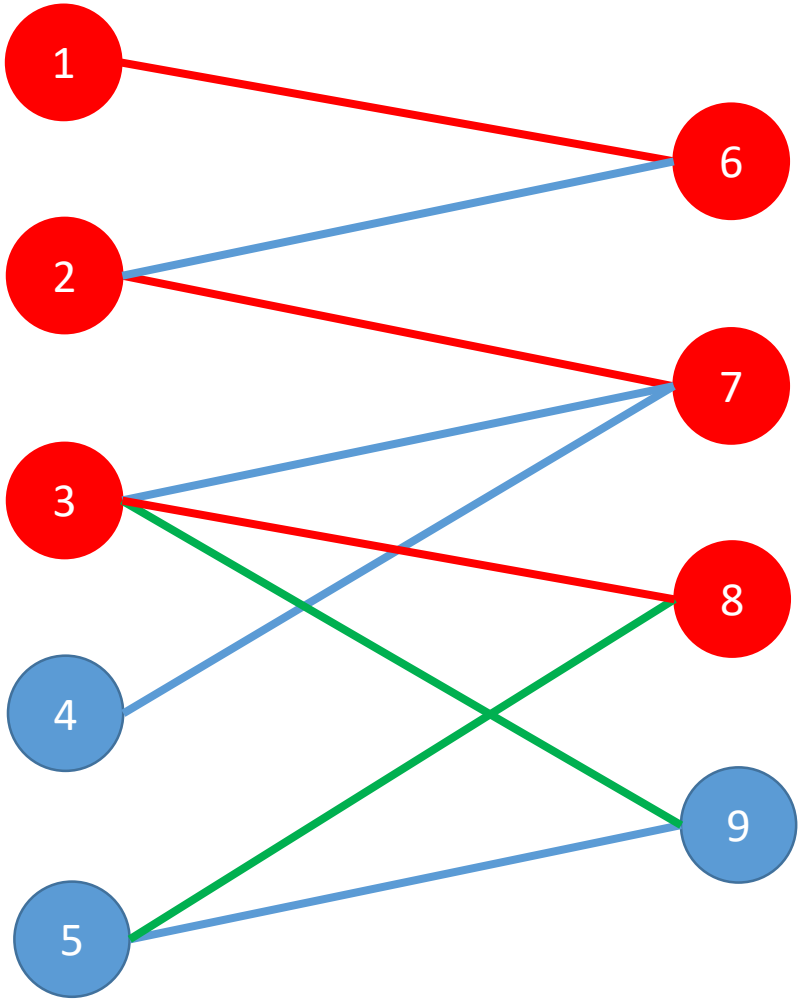

1
6
2
7
3
8
4
9
5

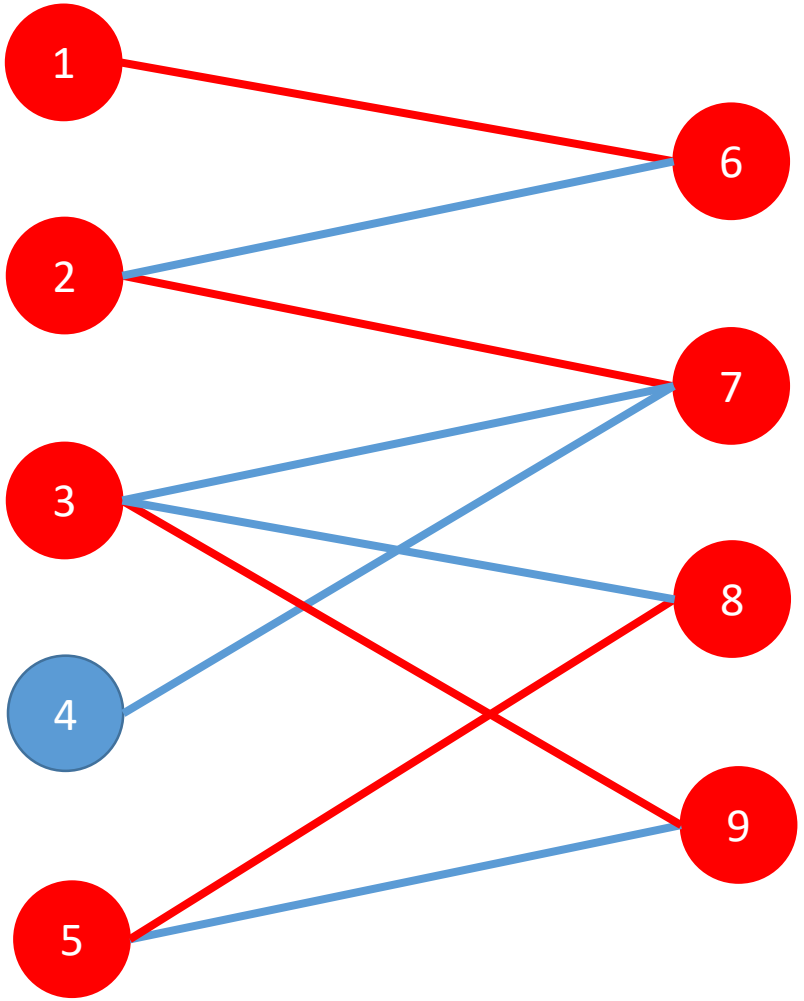

1
6
2
7
3
8
4
9
5


1
2
3
4
5
6
7
8
9


1
2
3
4
5
6
7
8
9


1
2
3
4
5
6
7
8
9


1
2
3
4
5
6
7
8
9


1
2
3
4
5
6
7
8
9

Time Complexity

- Each DFS takes O(|V| + |E|) time
- Number of DFS in the worst case O(|V|/2) Why?
  - Upper bound on matching i.e. upper bound on finding the number of augmenting paths
  - Size of the maximum matching = min(|X|, |Y|)
- Time Complexity = O(|V|/2 X (|V| + |E|)) = O(|VE|)