INT202
Complexity of Algorithms

Minimum Spanning Tree and Network Flow

Xi'an Jiaotong-Liverpool University
2020-2021

Bipartite Matching - Definitions

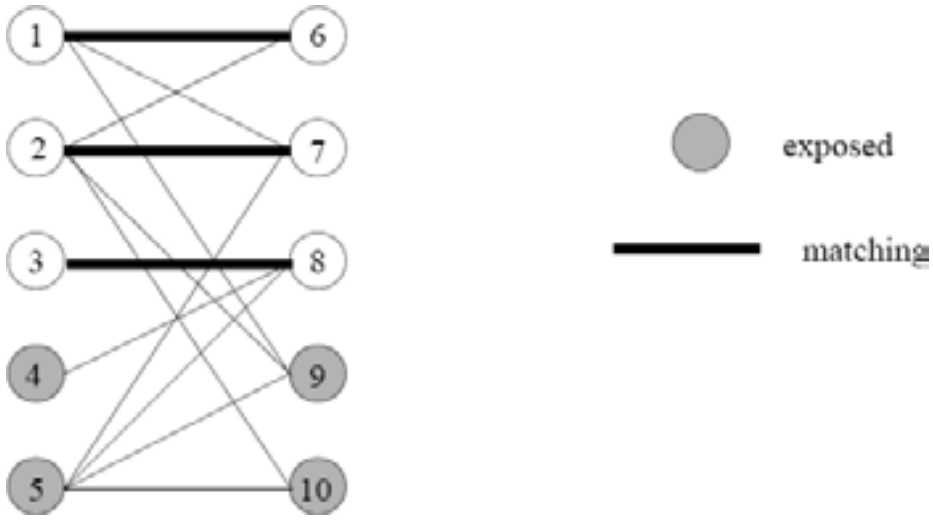
- A graph $G = (V, E)$ is called *bipartite* if and only if
  - The set V can be partitioned into two sets, X and Y.
  - Every edge of G has an endpoint in X and the other endpoint in Y.

- A *matching* $M \subseteq E$ is a set of edges that have no endpoints in common. Each vertex from one set has at most one "partner" in the other set.

- If a vertex v has no edge of M incident to it then v is said to be *exposed (or unmatched))*.

- A matching is *perfect* if no vertex is exposed.

Bipartite Matching - Example



The maximum bipartite matching problem is to find a matching M with the greatest number of edges (over all matchings).

Maximum Bipartite Matching

Maximum Bipartite Matching

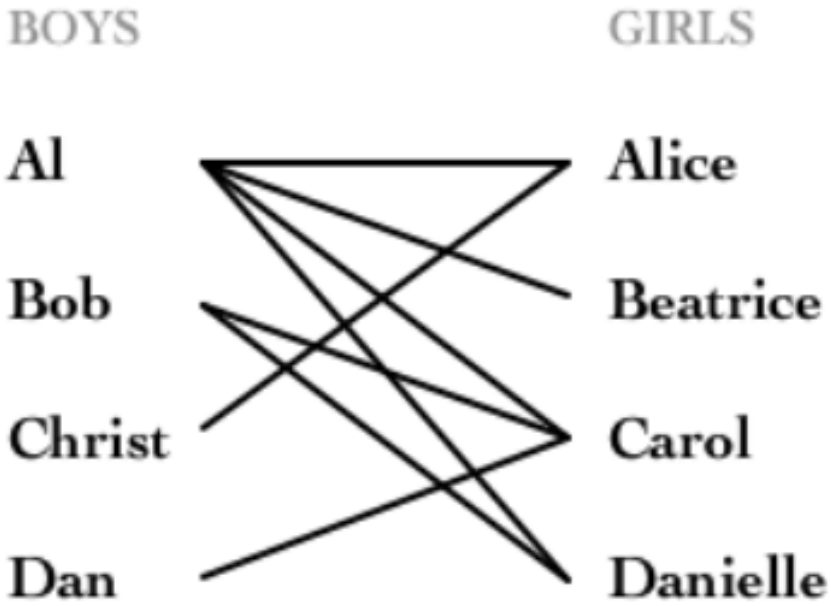
Given a bipartite graph $G = (A \cup B, E)$, find an $S \subseteq A \times B$ that is a matching and is as large as possible.

Notes:

- We're given A and B so we don't have to find them.
- S is a **perfect matching** if every vertex is matched.

Maximum Bipartite Matching

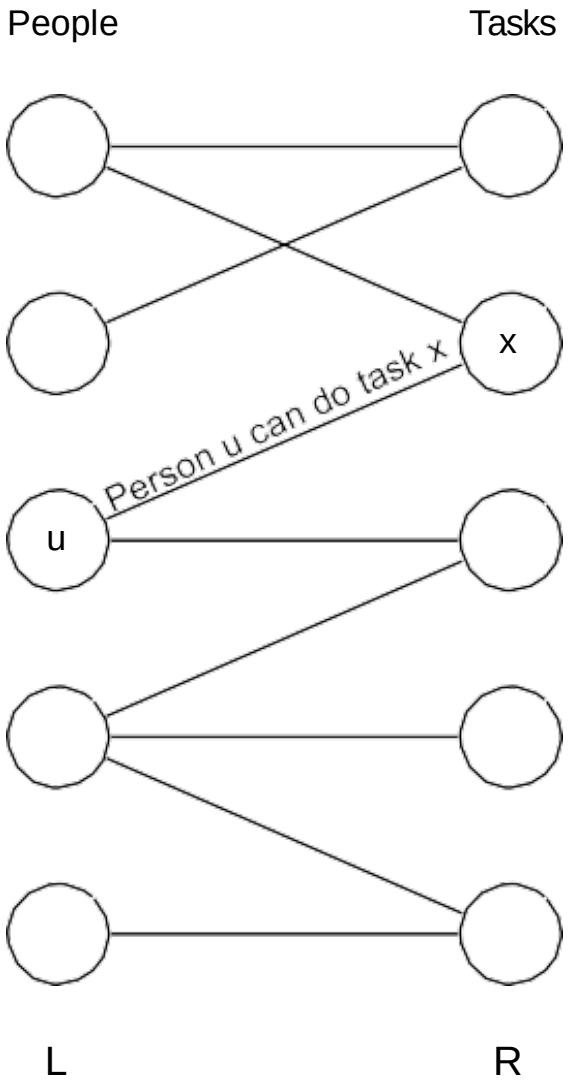
Example 1: Let G be a bipartite graph where the set X represents a group of boys and the set Y a group of girls, who are all together at a community dance. Let there be an edge joining $x \in X$ and $y \in Y$ if x and y are willing to dance with one another. A maximum matching in G corresponds to a largest set of compatible pairs of boy and girl who can all happily dancing at the same time.



Maximum Bipartite Matching

Example 2: Let G be a bipartite graph Suppose we have a set of people L and set of jobs R.

- Each person can do only some of the jobs.
- Can model this as a bipartitegraph



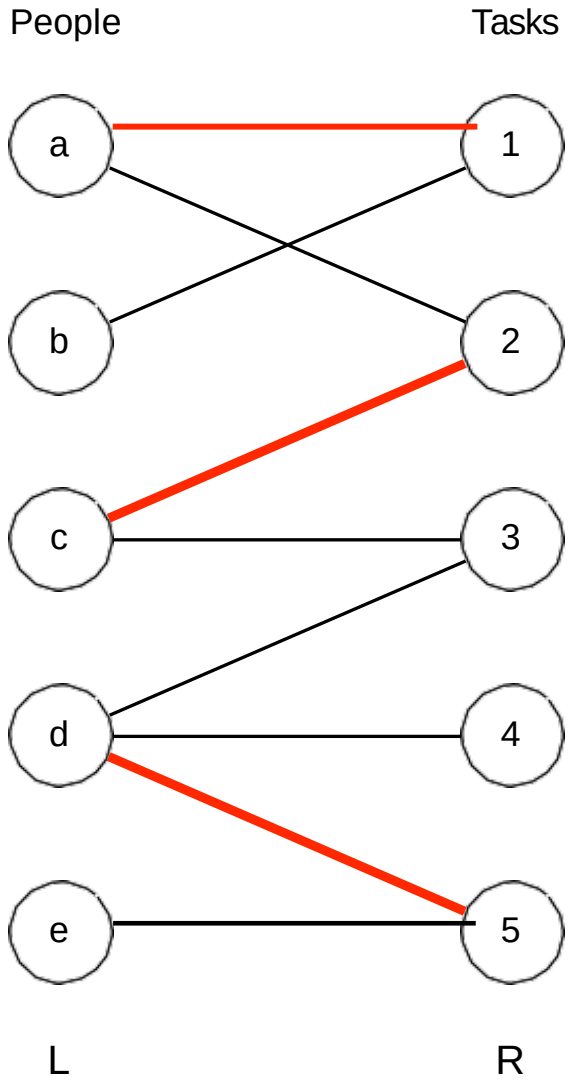
Maximum Bipartite Matching

- A **matching** gives an assignment of people to tasks.

If a vertex v has no edge of M incident to it then v is said to be **exposed** (**or unmatched**)).

A **maximal bipartite matching** is the largest subset of edges in a bipartite graph such that no two selected edges share a common vertex.

- (This one is not maximum.)



(a,1), (c,2),(d,5) are matched.
b,e,3,4 are exposed.

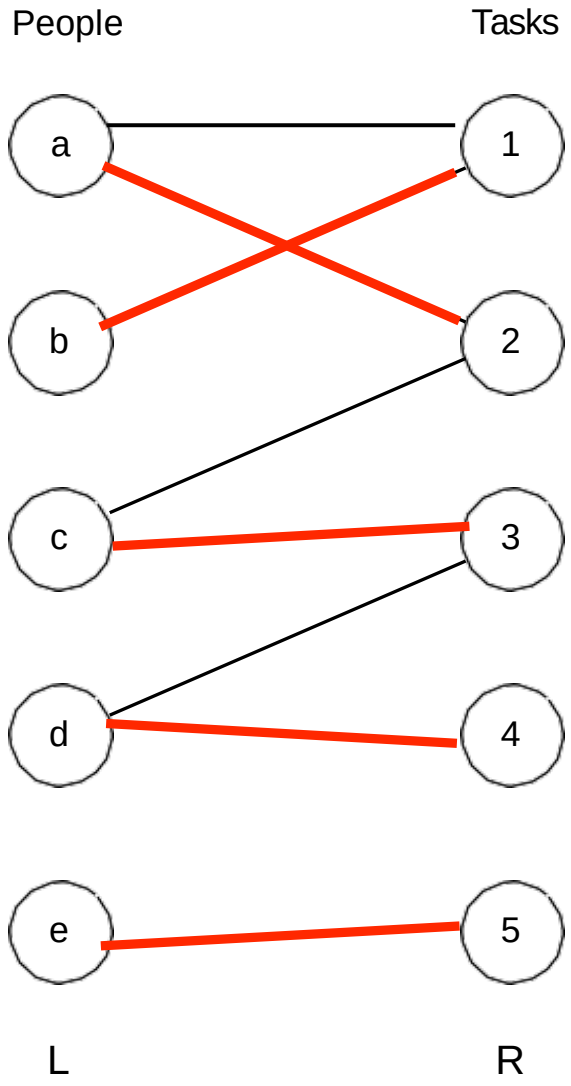
Maximum Bipartite Matching

- A **matching** gives an assignment of people to tasks.

If a vertex v has no edge of M incident to it then v is said to be **exposed** (**or unmatched**)).

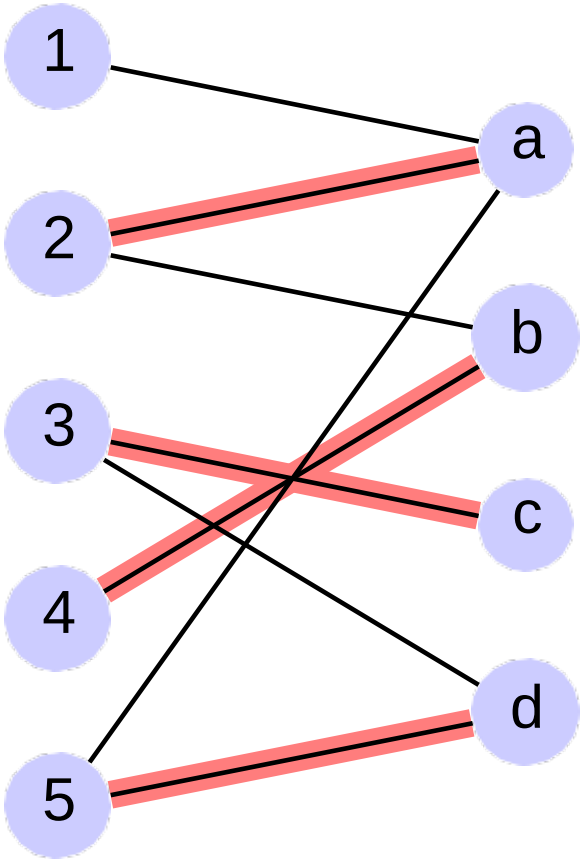
A **maximal bipartite matching** is the largest subset of edges in a bipartite graph such that no two selected edges share a common vertex.

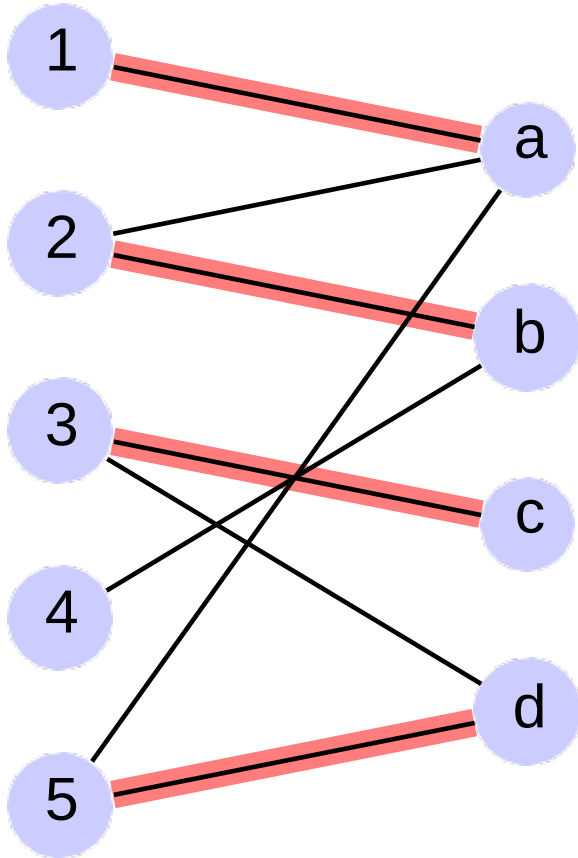
A matching is **perfect** if no vertex is exposed.



Maximum Bipartite Matching

Matching not guaranteed to be unique.



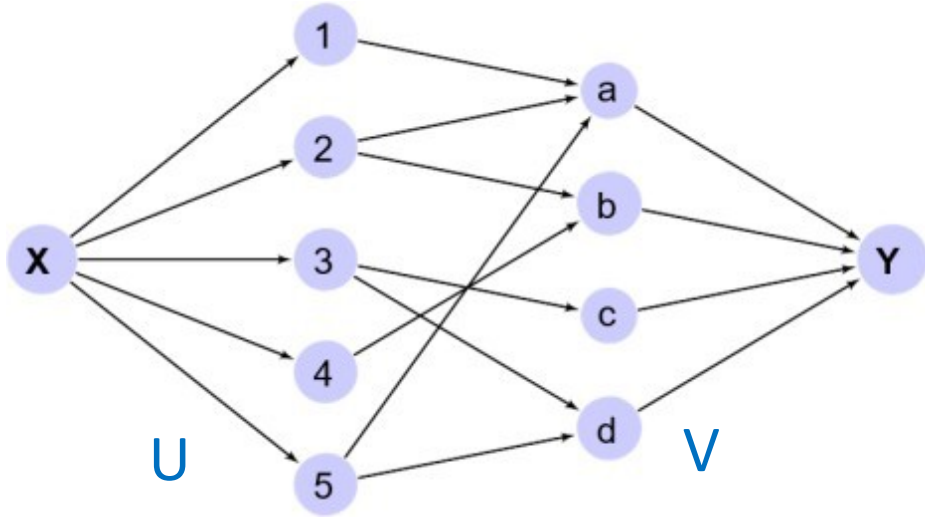


Reduce

We can solve the maximum bipartite matching problem using a *network flow* approach.

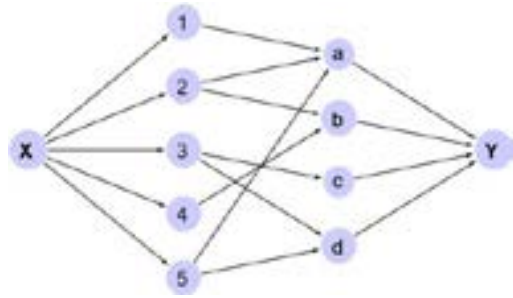
- We first ensure that all edges from *U* to *V* are directed.
- We introduce a single **source** node that has outgoing edges to each node in *U*
- We then add a **sink** node that has incoming edges from each node in *V* .

The problem now becomes finding the *maximal flow* in the graph.



We denote the source node as **X** and the sink node as **Y**

Reduce



One can more intuitively consider flow as water running through pipes represented by the edges.

There are many algorithms one could use to solve *network flow* problems (such as Ford–Fulkerson algorithm). For the specific case of **bipartite graphs**, we don't need the full generality of most network flow algorithms.

Since we've converted the problem to an unweighted directed graph, we can therefore use the following simplified algorithm which solves the max flow problem for any general unweighted directed graph.

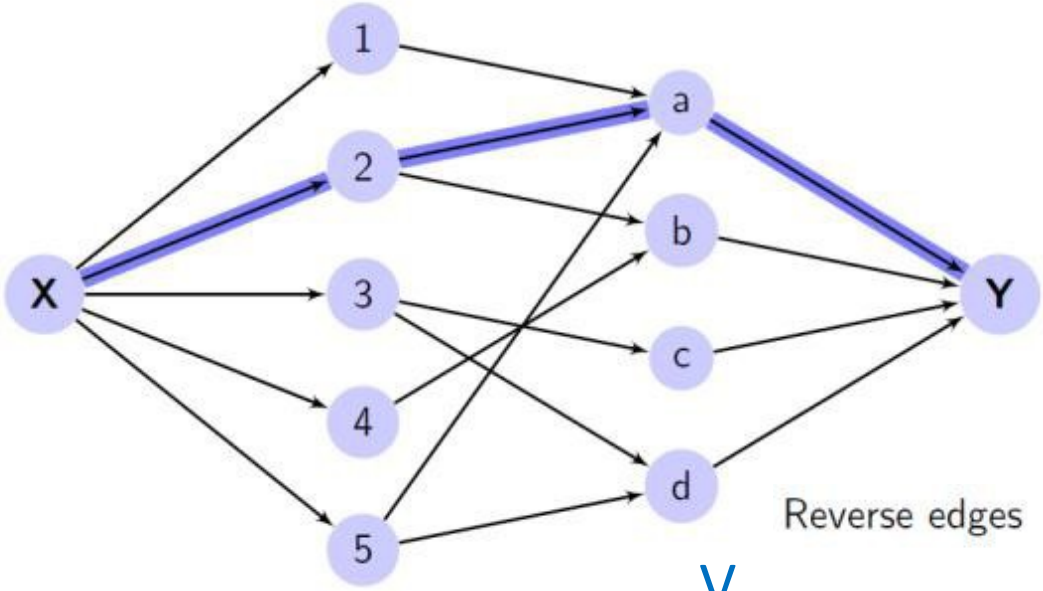
Maximum Bipartite Matching

Solving maximal flow for an unweighted directed graph:

1. Construct the directed graph with a source and sink node.
2. Using a search method to find a path from the source to the sink.
3. Once a path is found, reverse each edge on this path.
4. Repeat step 2 and 3 until no more paths from the sink to source exist.
5. The final matching solution is then the set of edges between U and V that are reversed (i.e. run from V to U).

Maximum Bipartite Matching

Found path: **X** → 2 → a → **Y**

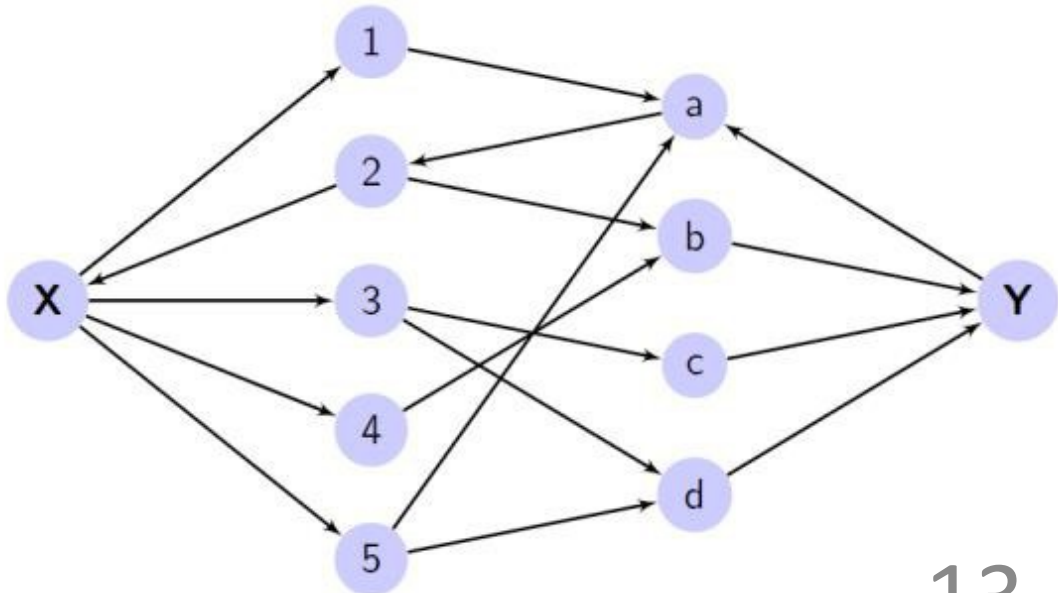


1st phase

Reverse edges

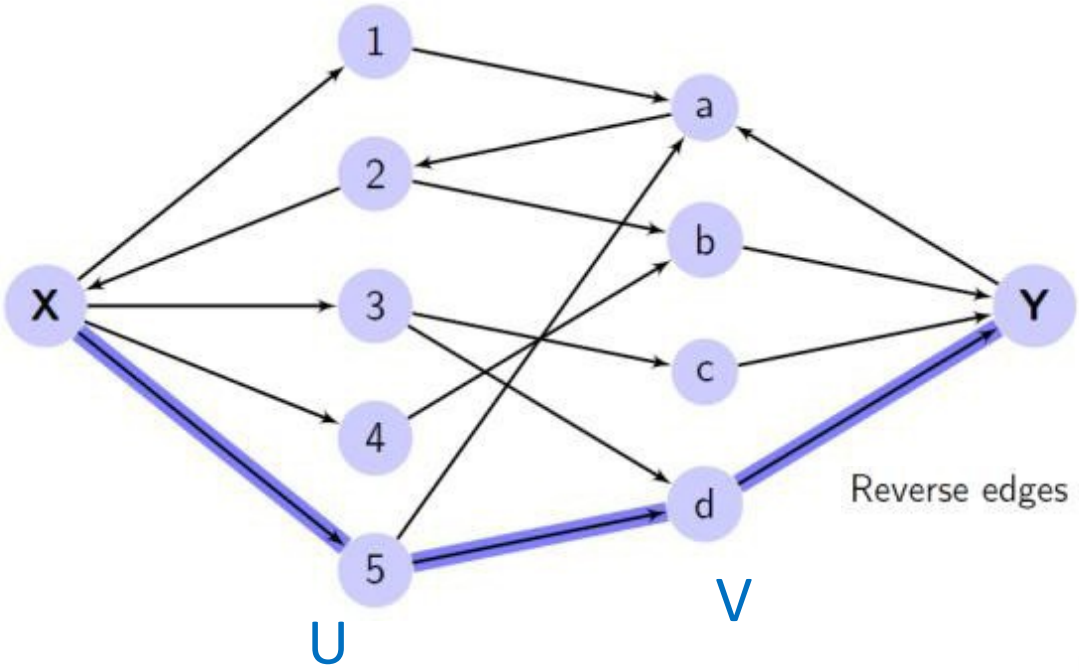
U

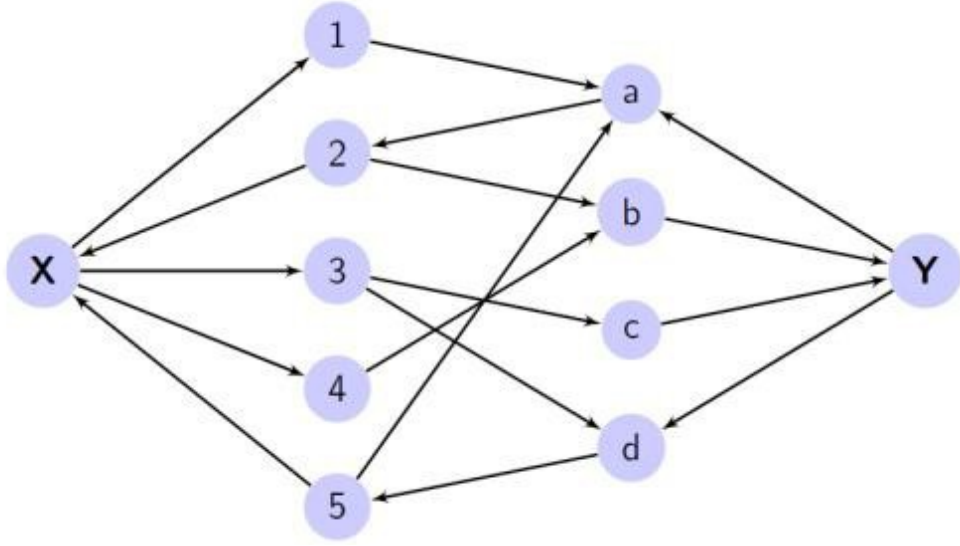
V



Maximum Bipartite Matching

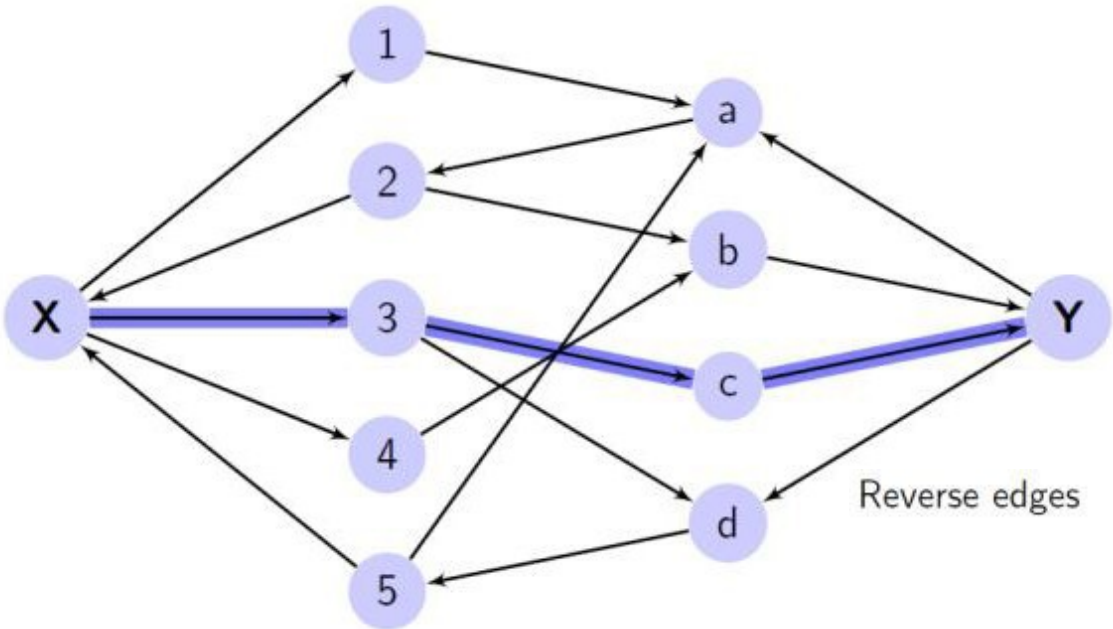
Found path: **X** → 5 → d → **Y**

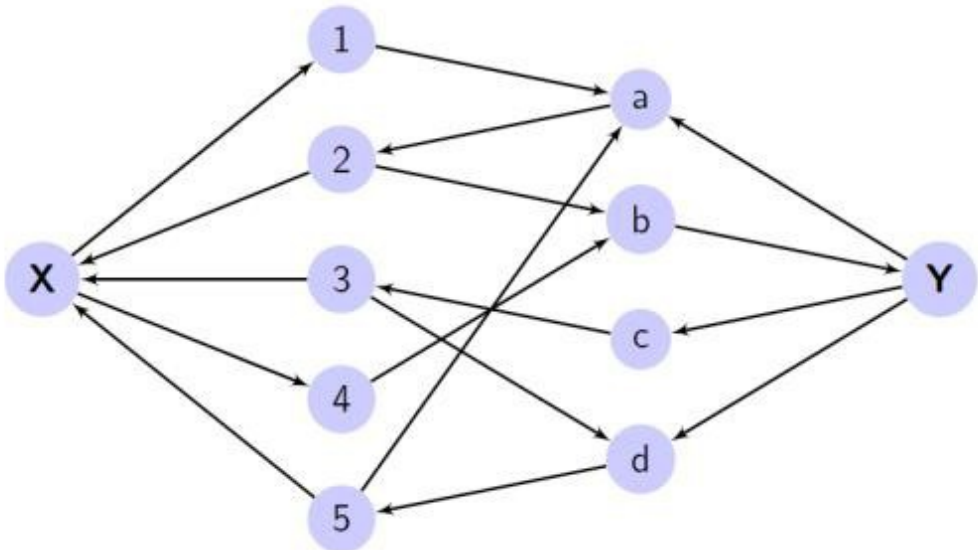




Maximum Bipartite Matching

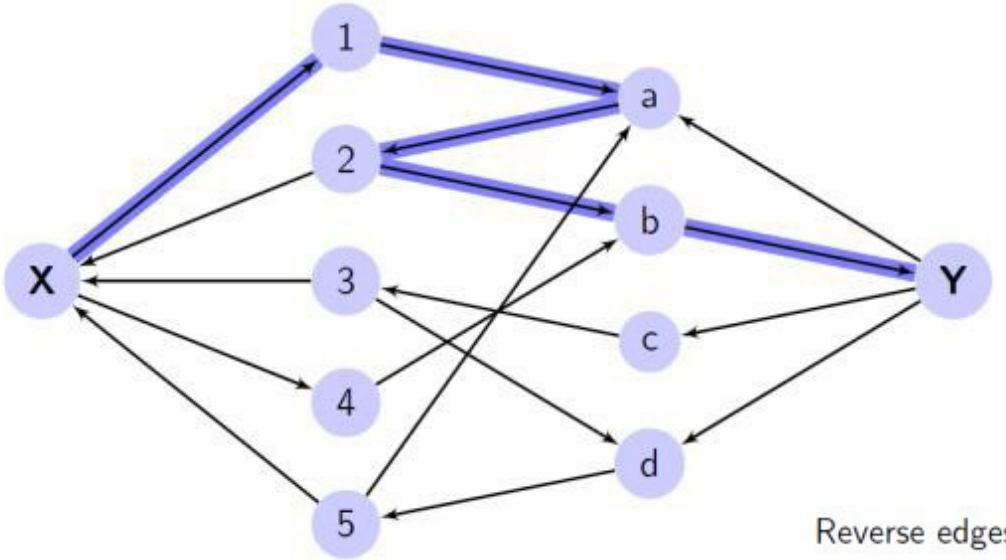
Found path: **X** $\rightarrow$ 3 $\rightarrow$ c $\rightarrow$ **Y**





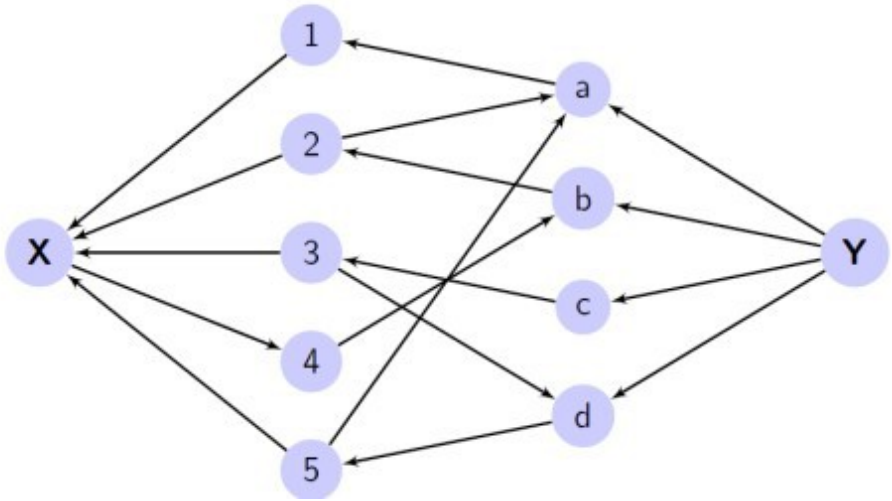
Maximum Bipartite Matching

Found path: **X** → 1 → a → 2 → b → **Y**



U V

Reverse edges



Maximum Bipartite Matching

No more paths found. Matching is reversed edges.