Bipartite Matching

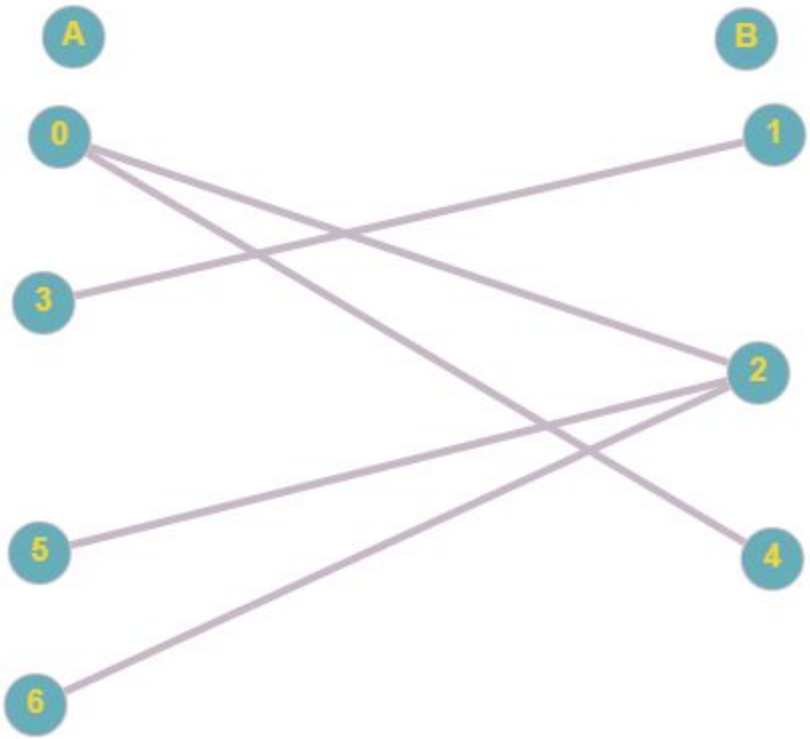
Emmanuel Rassou • 29.01.2022

What is a bipartite graph ?

A undirected, unweighted graph with two sets of vertices.

Call them A and B

Only allowed edges from a vertex in A to a vertex in B


A
B
0
1
3
2
5
4
6

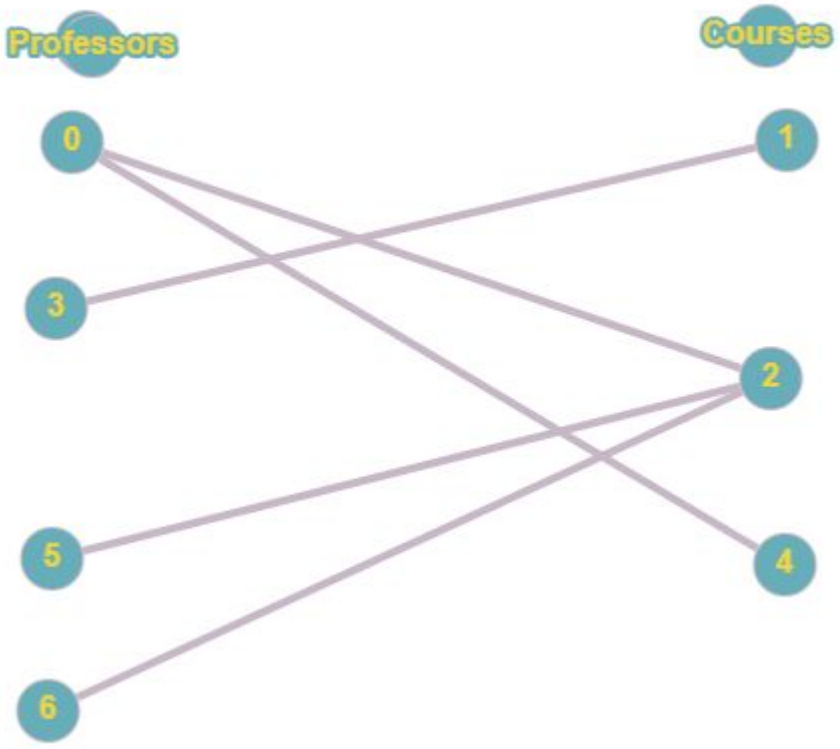
What is a matching?

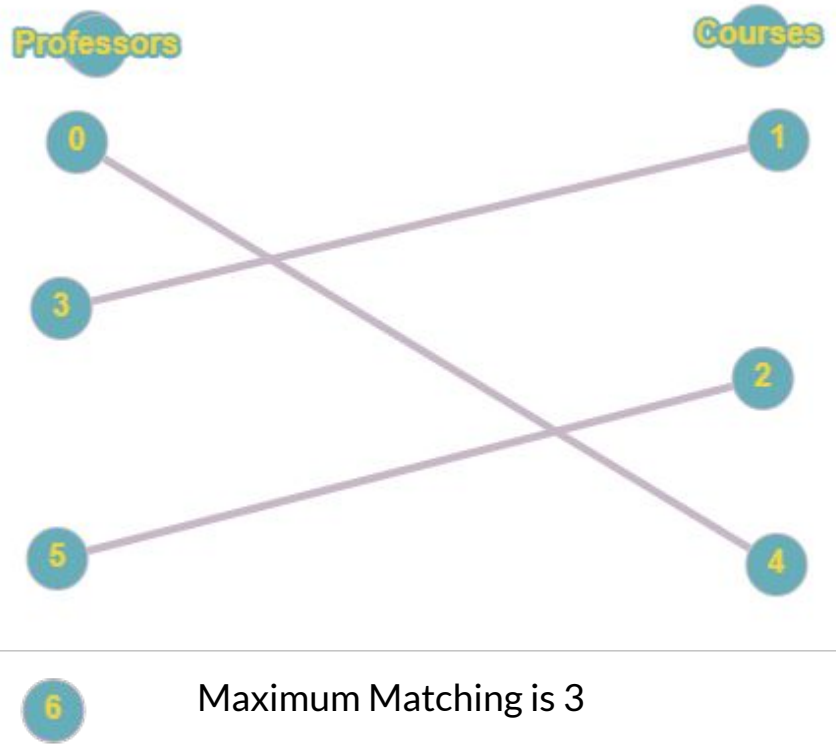
Let one vertex set be professors (p) and the other be courses (c)

Then each professor can only teach at most one course and each course can only be taught by at most one professor

In other words two edges in the matching cannot belong to the same vertex

A MAXIMUM matching will have the maximum amount of edges





Maximum Matching is 3

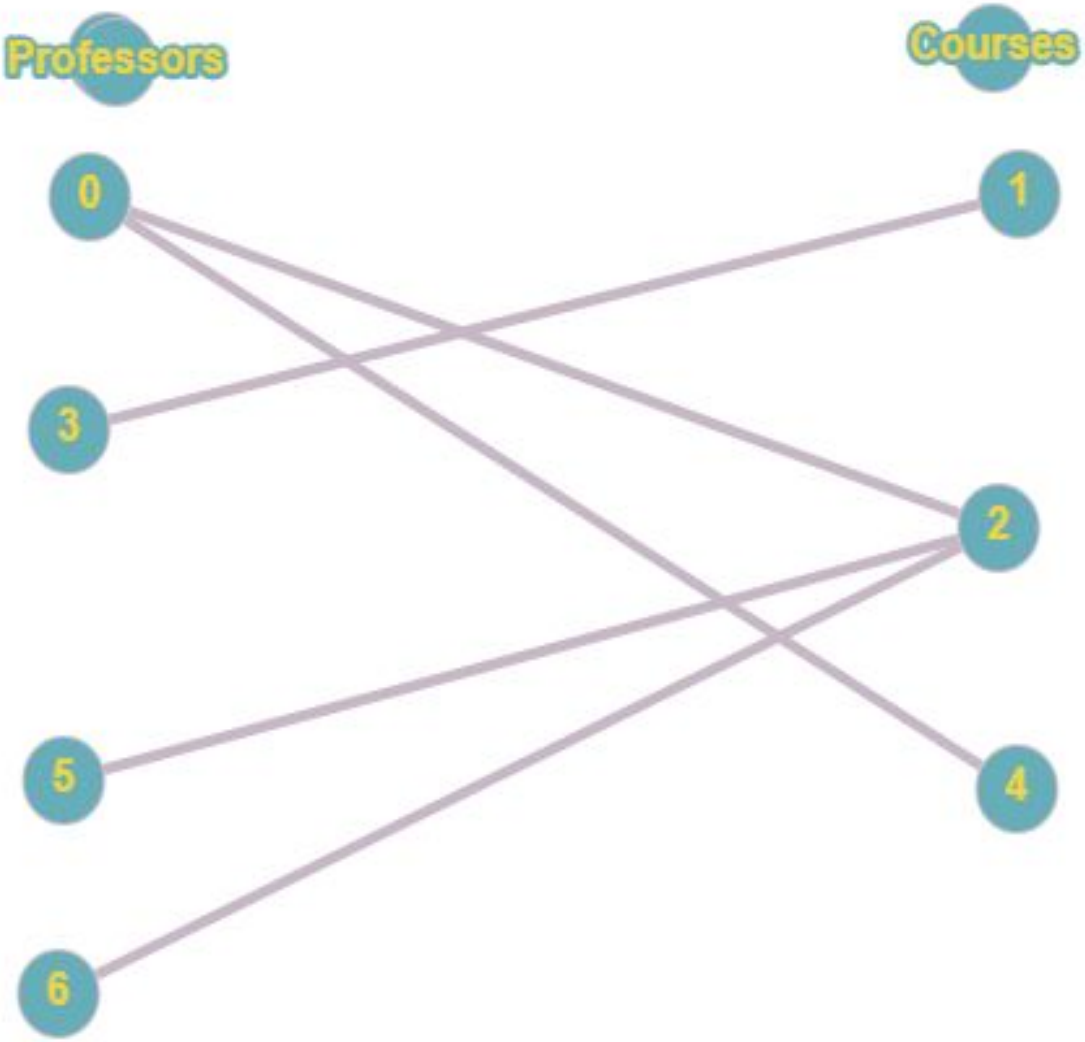
For every professor p

Look at every course c where there is an edge from p to c

- If course c is not matched yet then we just match p and c and then increase count
- Else course c is taught already by professor q
  - Now we want to move **q** to a different course
  - If q can be matched to another course then we rematch q and then match p and c, and increase count
  - Else q can't be matched to another open course which means q keeps teaching c and then p does not match to anyone (count doesn't increase)

Return count

*To check if q can be moved to another course we use recursion on a check function


Professors
Courses
0
3
5
6
1
2
4

Time Complexity

This is basically a DFS being run for each professor p

Let there be n professors and m courses.

Then V = n + m E <= m * n

So complexity will be n * O(E + V)

$O(n * (mn + m + n)) = O(mn^2 + mn + n^2) = O(mn^2)$

So basically O(V*E)

```cpp
#include <bits/stdc++.h>
using namespace std;
const int MAX_N = 505;

int n , m, e; // number of professors, courses and edges
int adj[MAX_N][MAX_N]; //adjacency matrix
int seen[MAX_N]; //check if course was visited already
int matchR[MAX_N]; //records the professor for each course
int matchL[MAX_N]; //records the course for each professor

bool bpm(int p);//our check function

int main(){
    memset(matchR, -1, sizeof matchR);
    memset(matchL, -1, sizeof matchL);
    cin >> n >> m >> e;
    while(e --){
        int p, c;
        cin >> p >> c;
        adj[p][c] = 1;
    }
    int count  = 0; //answer
    for(int p = 0; p < n; p ++){
        memset(seen, 0, sizeof seen);
        if (bpm(p)){
            count ++;
        }
    }
    cout << count << '\n';
```

```
bool bpm(int p){
    for(int c = 0; c < m ; c++){
        if (adj[p][c]){ //check if there is edge from p to c
            if (seen[c]){ //check if course c has already been visited
                continue;
            }
            seen[c] = 1;
            if (matchL[c] < 0 || bpm(matchL[c])){ //check if professor q exists or can be moved
                matchL[c] = p;
                matchR[p] = c;
                return true; //yay
            }
        }
    }
    return false;
}
```