Graph Theory and its applications
Assignment 2 & 3

Name : Adarsh Subhas Nayak
SRN : PES1UG20CS620
Roll No : 54
Section : K
Date : 28-11-2022

Problem statement :

Consider course allotment to faculty in department of CSE in PESU.Each faculty submit their preference (at least 3) to the chairperson. The department chairperson allots one course per faculty. Find a suitable allotment by modeling the problem as Maximum Bipartite Matching.

Referring to the lecture from the : COMPSCI 330: Design and Analysis of Algorithms, Lecture 16 : Bipartite Matching by Rong Ge.

3 Maximum Bipartite Matching

Now we present the Maximum Bipartite Matching problem. Given a bipartite graph with n vertices on one part, and m vertices on the other part. We try to find a maximum matching in the graph. Relating back to our motivating problem: matching corresponds to an assignment of courses to classrooms and maximum matching corresponds to scheduling max number of courses.

Bipartite Graph: A Bipartite Graph $G = (V_1, V_2, E)$, E is a subject of (i, j) where $i \in V_1, j \in V_2$.

Matching: A matching M is a subset of E, such that edges in M do not share vertices. The size of a matching M is just the number of edges in M.

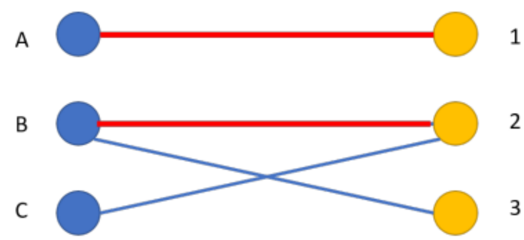
3.2 Augmenting Path Algorithm

Before introducing augmenting path algorithm, we first state several definitions. Given a Bipartite Graph G and matching M:

Matched Edge: An edge e is matched if $e \in M$ and unmatched if otherwise.

Matched Vertex: A vertex v is matched if it's connected to some $e \in M$ and unmatched if otherwise.

Augmenting Path: An augmenting path is a path from an unmatched course to an unmatched classroom, that alternates between unmatched edges and matched edges. In below example, path (C, 2), (2, B), (B, 3) is an augmenting path. By definition, we can easily derive the following, **Claim:** An augmenting path P has an odd number of edges, and it has exactly 1 more unmatched edges than matched edges.



Algorithm 1 Augmenting Path Algorithm

```
procedure FindPath(u)
    Mark u as visited
    for all edges (u, v) do                ▷ Enumerate over classrooms that course u can use
        if v is not visited then
            if v is unmatched or FindPath(matchRoom[v]) = true then    ▷ if either we found an empty classroom, or the current instructor of the classroom is able to switch to another room
                matchCourse[u] = v
                matchCourse[v] = u                       ▷ I will take this room
                return true                              ▷ I have found a room for course u.
            end if
        end if
    end for
    return false        ▷ I have tried all the possible rooms, they are not empty and their instructor cannot switch to another room, so I cannot find a room for course u.
end procedure
procedure MaxMatching
    Initially set all nodes to be unmatched
    for u=1 to n do                                      ▷ enumerate the courses
        Mark all vertices as unvisited                   ▷ initialize for the DFS
        FindPath(u)                                      ▷ Try to schedule course u.
    end for
```

Code :

```
#include<bits/stdc++.h>
using namespace std;

#define gcd(a,b)                    __gcd(a,b)
#define lcm(a,b)                    (a/gcd(a,b))*b
const int MAX_N = 1e5 + 5;
const int MOD = 1e9 + 7;
const int INF = 1e9;
#define pb(x)                       push_back(x)
#define M 6
#define N 6

bool bipartiteGraph[M][N] =
{
   {0, 1, 1, 1, 0, 1},
   {1, 0, 1, 0, 1, 0},
   {1, 0, 1, 1, 0, 1},
   {1, 0, 1, 0, 1, 0},
   {0, 0, 1, 1, 1, 0},
   {1, 1, 1, 0, 0, 1}
};

bool bipartiteMatch(int u, bool visited[], int assign[]) {
   for (int v = 0; v < N; v++) {
      if (bipartiteGraph[u][v] && !visited[v]) {
         visited[v] = true;
         if (assign[v] < 0 || bipartiteMatch(assign[v], visited, assign))
{
            assign[v] = u;
            return true;
         }
      }
   }
   return false;
}

void solve() {
   int assign[N];
   for(int i = 0; i<N; i++)
```

```
      assign[i] = -1;
   int jobCount = 0;

   for (int u = 0; u < M; u++) {
      bool visited[N];
      for(int i = 0; i<N; i++)
         visited[i] = false;
      if (bipartiteMatch(u, visited, assign))
         jobCount++;
   }
   for(int i=0;i<N;i++)
   {
       if(assign[i]==-1)
       {
           cout<<"The course number "<<i<< " has not been assigned to any
faculty"<<endl;
       }
       else
       {
            cout<<"The course number "<< i << " has been assigned to
faculty "<<assign[i]<<endl;
       }
   }

}

int main() {
   // ios::sync_with_stdio(false);
   // cin.tie(NULL);
   // cout.tie(NULL);
   // int tc = 1;
   // cin >> tc;
   // for (int t = 1; t <= tc; t++) {
   // cout << "Case #" << t << ": ";
      solve();
   // }

}
```

Output Screenshots :

```
cd "c:\Users\Hp\Desktop\GTA
Assignments\Assingnment 2&3\" ; if ($?) { g++ PES1UG20CS620_Adarsh_S_Nayak.cpp -o PES1UG20CS
620_Adarsh_S_Nayak } ; if ($?) { .\PES1UG20CS620_Adarsh_S_Nayak }
The course number 0 has been assigned to faculty 5
The course number 1 has been assigned to faculty 0
The course number 2 has been assigned to faculty 3
The course number 3 has been assigned to faculty 4
The course number 4 has been assigned to faculty 1
The course number 5 has been assigned to faculty 2
C:\Users\Hp\Desktop\GTA Assignments\Assingnment 2&3>
```