Competitive Programming

From Problem 2 Solution in O(1)

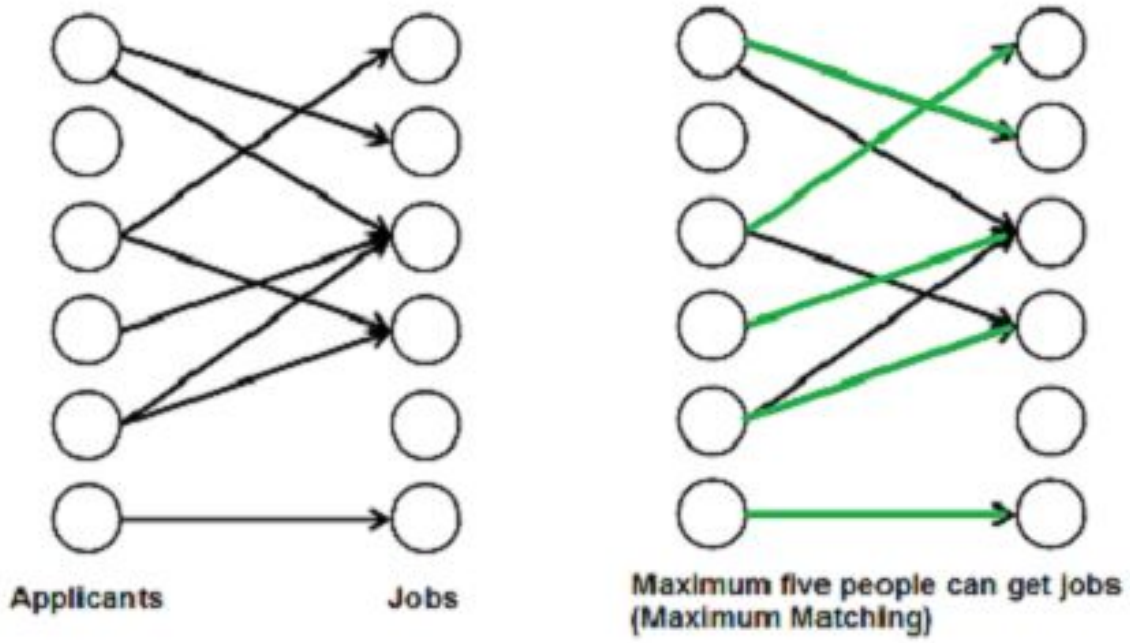
Graph Theory

Maximum Bipartite Matching

Mostafa Saad Ibrahim
PhD Student @ Simon Fraser University

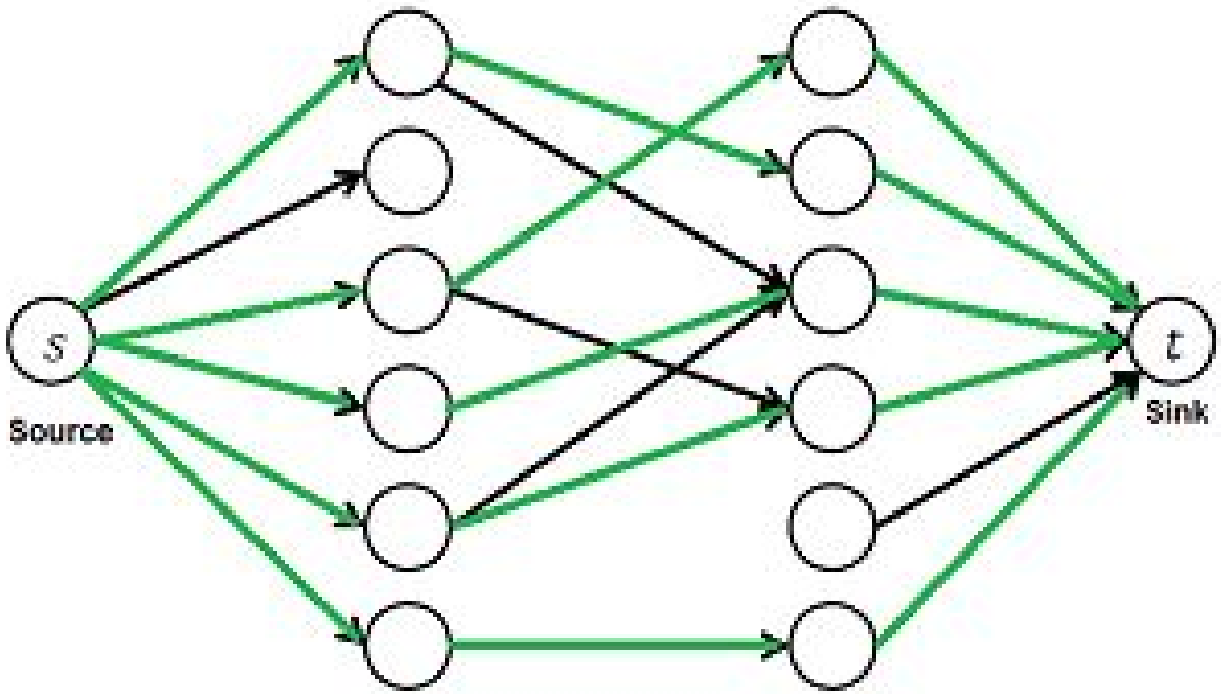
Recall: Reduction to Flow

- One way to solve this problem is to reduce to maximum flow problem



Src: http://www.geeksforgeeks.org/maximum-bipartite-matching/

Recall: Reduction to Flow



The maximum flow from source to sink is five units. Therefore, maximum five people can get jobs.

Src: http://www.geeksforgeeks.org/maximum-bipartite-matching/

Recall: Ford-Fulkerson Algorithm

- While (**augmenting path**)
  - Flow += path flow

Recall: Augmenting Path

- Augmenting = Increase
- Path From Source to Sink in **residual** network
  - The updated network given previous paths
- Path Edges = Positive Capacities
- Flow p to push = Min Capacity on path
- $p > 0 \Rightarrow$ We can increase flow

Today Algorithm

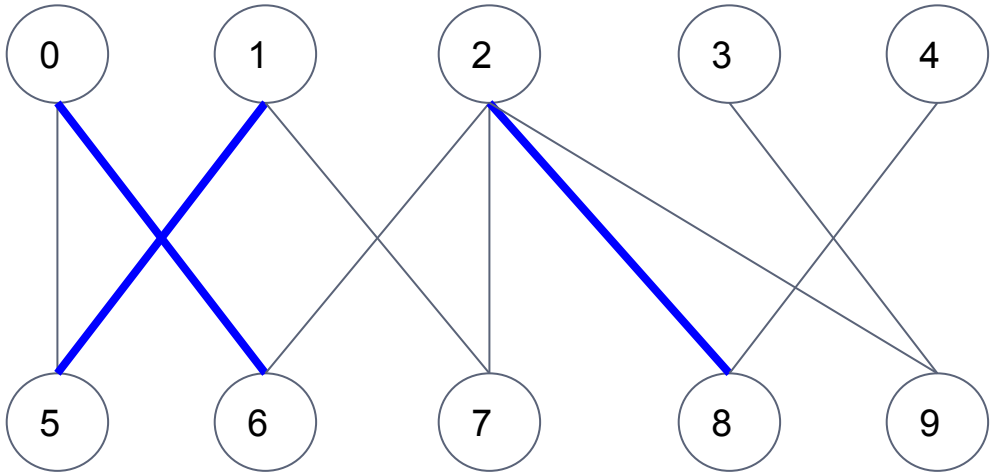
- A simplified Ford-Fulkerson Algorithm
  - Called **Kuhn's** algorithm
- Customize augmenting path for bipartite graph
- Simpler code...intuitive..easier to improve
- Notice, a bipartite graph has **V/2 Flow** maximum, where each vertex matched to other
- Find path is **O(E)**
- Hence Same order: O(EV) for bipartite graph

Initial Matching

Flow = 3
(0, 6), (1, 5), (2, 8)

10 nodes => max possible flow = 5

Can we increase the flow?
E.g. Match one more vertex?



4 has only one connection to 8

Can we add it?

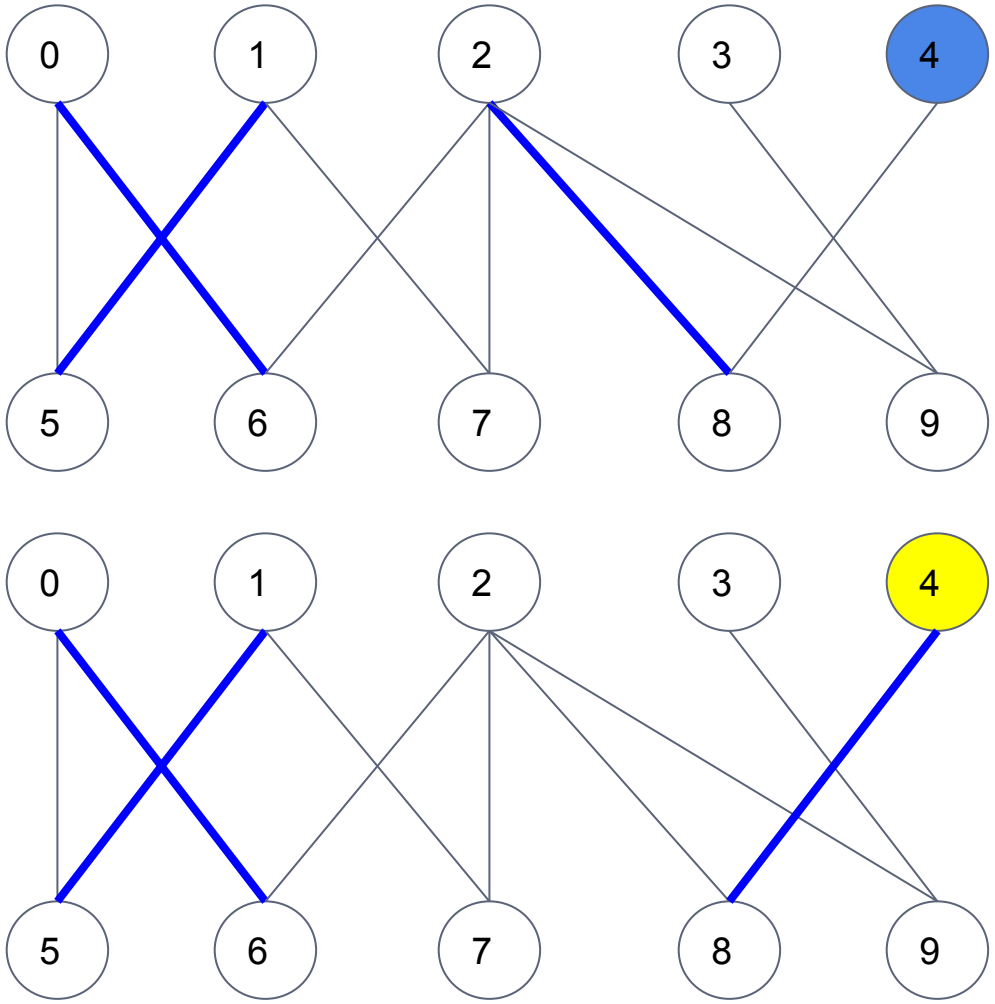
but 8 already connected! Can we rematch it with other node?

Graph Src: http://www.dis.uniroma1.it/~sankowski/lecture2.pdf

canMatch(4)

4 has 1 choice only: 8
Try match 4-8

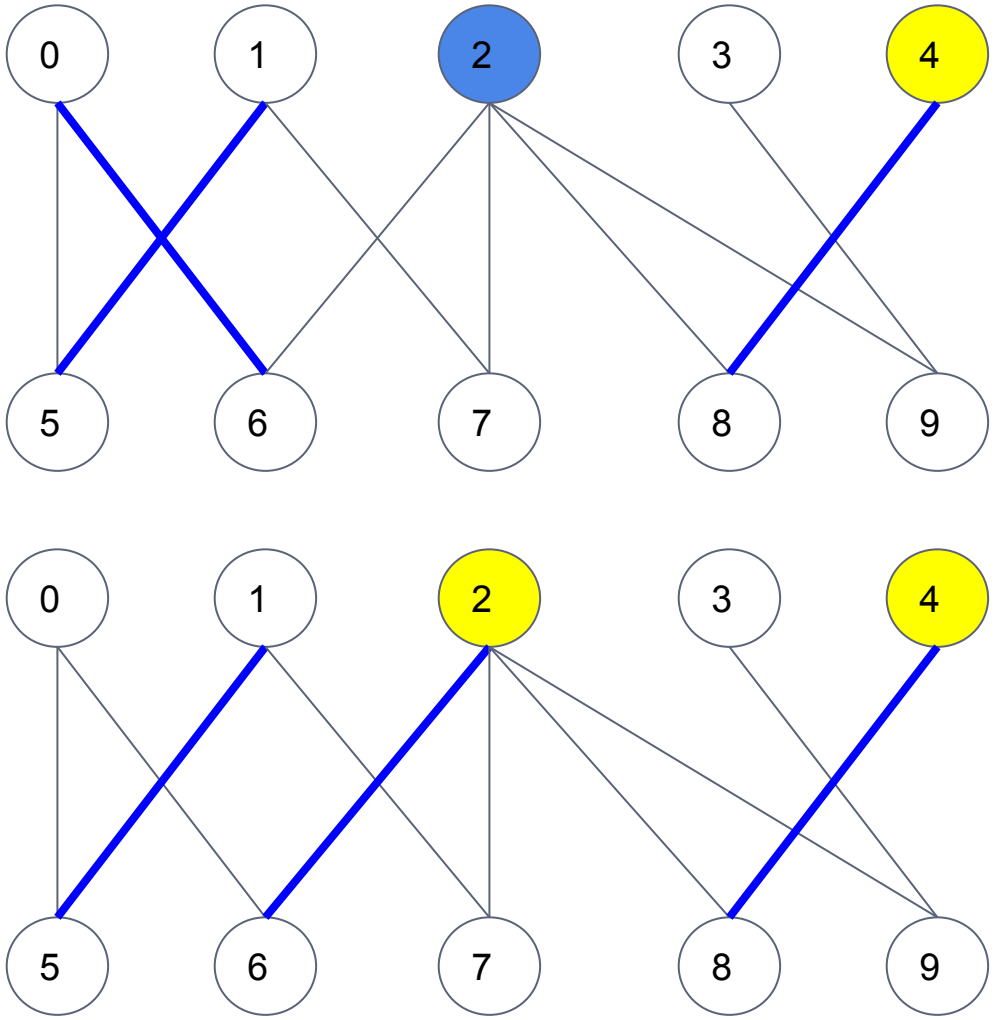
Remove 2 - 8 connection
Match 4 to 8
canMatch(4) = canMatch(2)?



canMatch(2)

2 has 4 choices: 6, 7, 8, 9
Try match 4-6

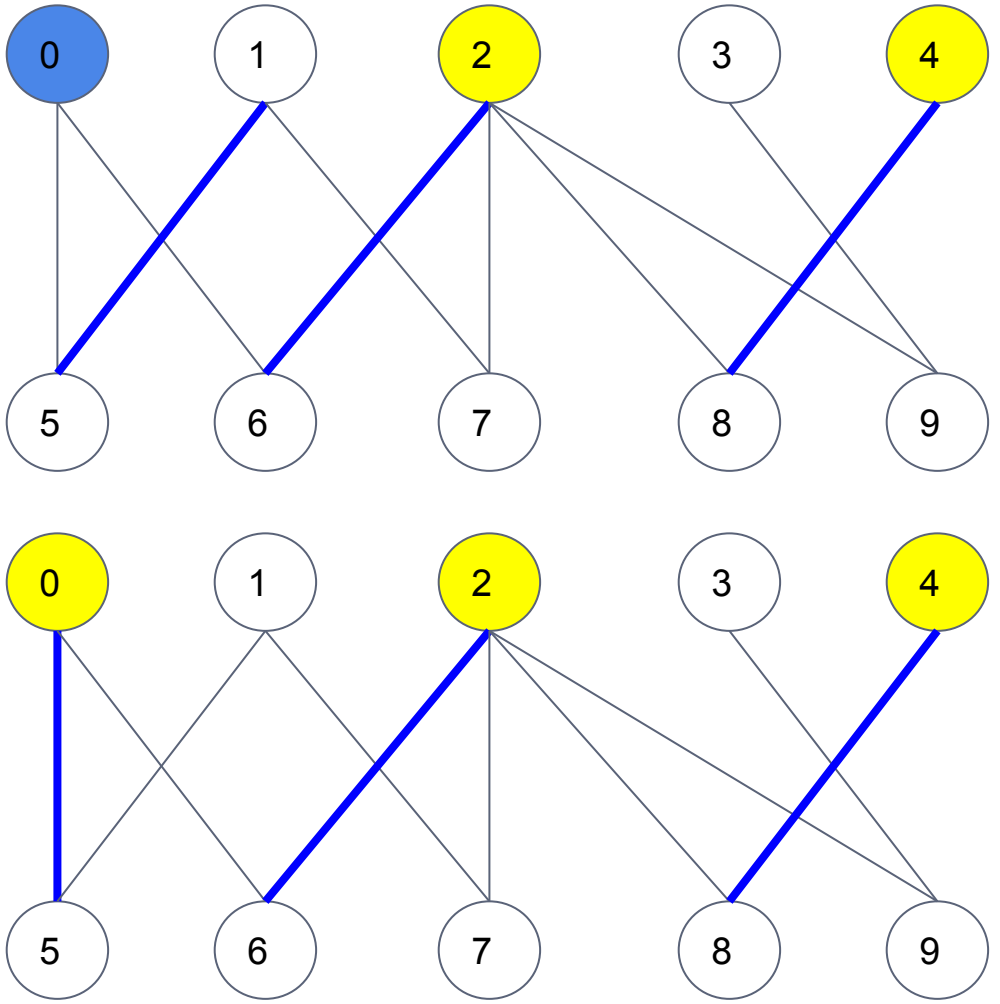
Remove 0 - 6 connection
Match 2 to 6
canMatch(2) = canMatch(0)?



canMatch(0)

0 has 2 choices: 5, 6
Try match 4-6

Remove 0 - 5 connection
Match 0 to 5
canMatch(0) = canMatch(1)?

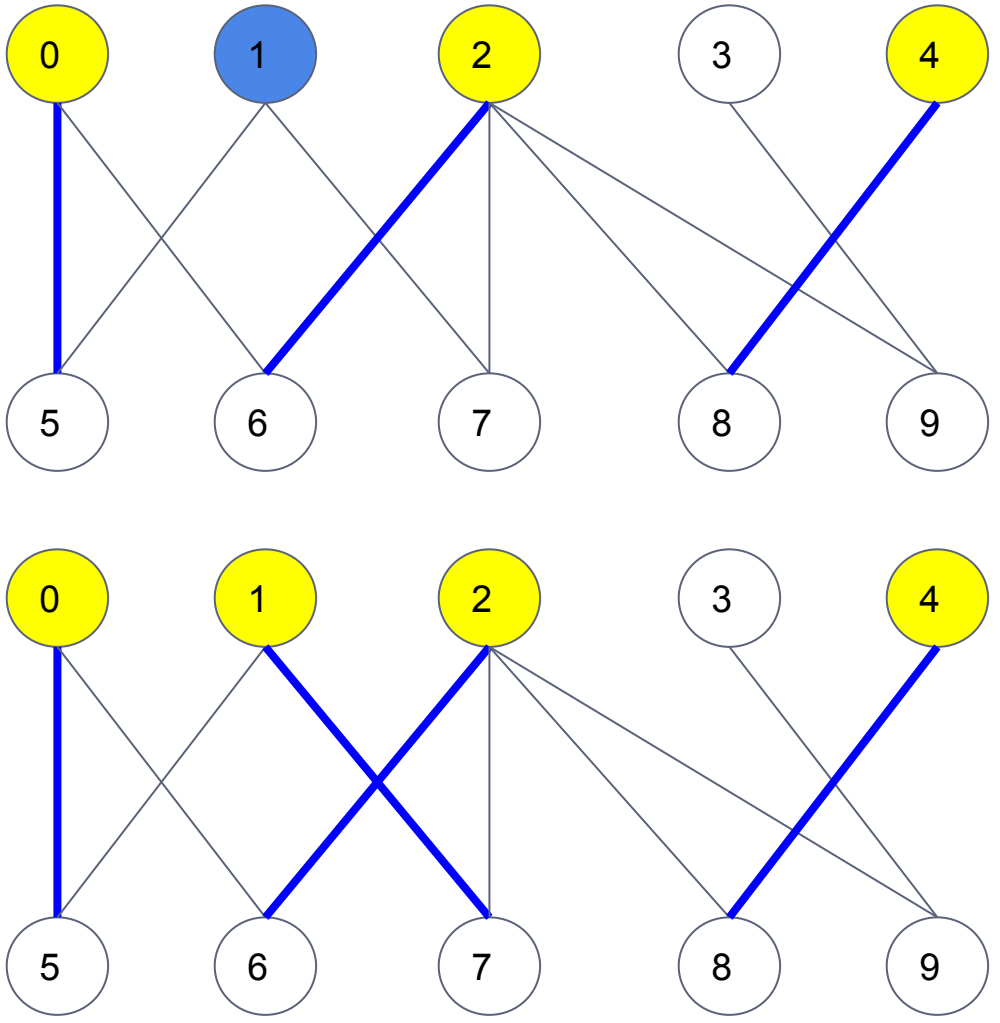


canMatch(1)

1 has 2 choices: 5, 7
Try match 1-7

Available. Just add it

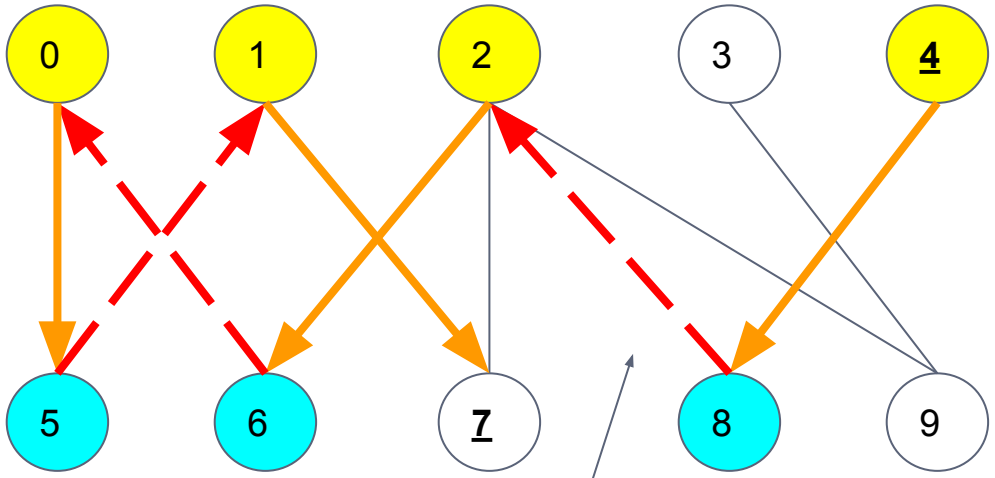
canMatch(1) = true
then originally
canMatch(4) = true



Augmenting path

Augmenting path:
4-8-2-6-0-5-1-7 =
Add 4-8 Cancel 8-2
Add 2-6 Cancel 6-0
Add 0-5 Cancel 5-1
Add 1-7

Odd length



Paths to Down
Originally unmatched

Paths to Up
Originally **matched**

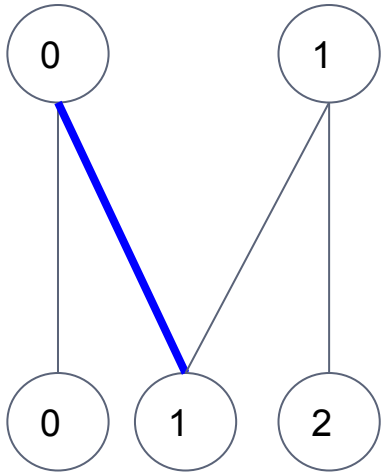
First/Last node
Originally un**matched**

Visited Node?

- canMatch(7)
  - canMatch(5)
    - CanMatch(2)
      - CanMatch(4) = False
      - CanMatch(11) = False
      - Nothing more = False
    - CanMatch(6)
      - **CanMatch(2)** = Visited before = False
- If node can't match, it will never match
- Then overall order of augmented path: O(E)

Code

- Build adjacency[m][n] with (i, j) = 1 if edge from ith top node to jth bottom node
  - E.g. matrix rows = top, cols = bottom



Adjacency matrix 2 x 3 = represents **5 nodes**

1	1	0
0	1	1

Matched Edges{ (0, 1) }, -1 = Not assigned

BottomAssign = ColAssign = {-1, 0, -1}
TopAssign = RowAssign = {1, -1}

Code

```
typedef vector<int>       vi;
vector<vi> adjMax;
vi vis, colAssign, rowAssign;

vector< pair<int, int> > bipartiteMatching()    // O(EV)
{   // In case spares graph, use adjList
    vector< pair<int, int> > matches;

    if(sz(adjMax) == 0)
        return matches;

    int maxFlow = 0, rows = sz(adjMax), cols = sz(adjMax[0]);
    colAssign = vi(cols, -1), rowAssign = vi(rows, -1);

    lp(i, rows) {
        vis = vi(cols, 0);
        if( canMatch(i)  )
            maxFlow++;
    }

    lp(j, cols) if(colAssign[j] != -1)
        matches.push_back( make_pair(colAssign[j], j) );
    // this is col sorted list...u can use rowAssign for reverse
    // as you see, rowAssign was not important now
    return matches;
}
```

Code

```
bool canMatch(int i) // O(E)
{
    rep(j, adjMax[i]) if(adjMax[i][j] && !vis[j]) {
        vis[j] = 1;
        if( colAssign[j] < 0 || canMatch(colAssign[j]) ) {
            colAssign[j] = i, rowAssign[i] = j;
            return true;
        }
    }
    return false;
}
```

Min Path Coverage in DAG

- Please revise this problem in max flow video

```
vector< vector<int> > mnPathCvs;

lp(j, n) if(colAssign[j] == -1) {
    vector<int> v(1, j+1);
    int t = rowAssign[j];
    while(t != -1) {
        v.push_back(t+1);
        t = sz(v)%2 ? rowAssign[j] : colAssign[j] ;
    }
    mnPathCvs.push_back(v);
}
rep(i, mnPathCvs)
{
    rep(j, mnPathCvs[i])
        cout<<mnPathCvs[i][j]<<" ";
    cout<<"\n";
}
```

Readings

- Link 1
- Hopcroft–Karp Algorithm for O(E Sqrt(v))

تم بحمد الله

علمكم الله ما ينفعكم

ونفعكم بما تعلمتم

وزادكم علماً