L19 M-Coloring Problem

Monday, May 22, 2023 9:38 PM

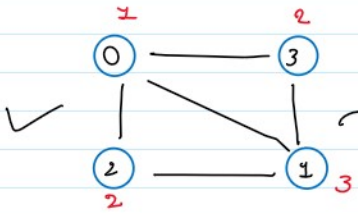
Given an undirected graph and an integer **M**. The task is to determine if the graph can be colored with at most M colors such that no two adjacent vertices of the graph are colored with the same color. Here coloring of a graph means the assignment of colors to all vertices. Print 1 if it is possible to colour vertices and 0 otherwise.

Example 1:
Input:
N = 4
M = 3
E = 5
Edges[] = {(0,1),(1,2),(2,3),(3,0),(0,2)}
Output: 1
Explanation: It is possible to colour the given graph using 3 colours.

→ Basically in this problem we identify that we coloured Node Graph with M Colors are Not

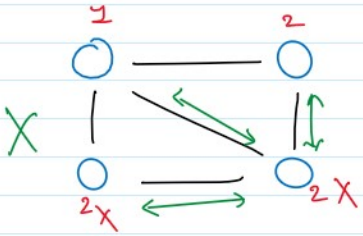
Example 2:
Input:
N = 3
M = 2
E = 3
Edges[] = {(0,1),(1,2),(0,2)}
Output: 0

Let's understand



M=3 → Number of the Colors

→ Now if we arranged colour in this way then Non of the Adjacent are same and our Graph get Coloured



M=2

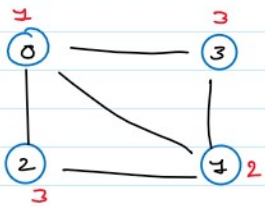
→ we can Not Make the Coloured Graph using (2) colors

Thought Process

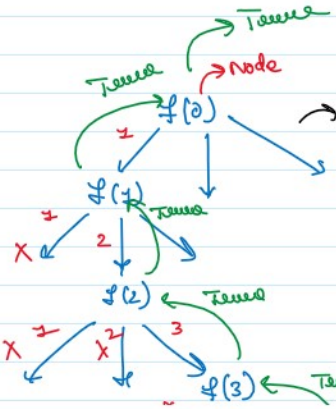
→ Now if I want do colour the complete Graph without breaking the property then obviously I find the **All the possible way** to Coloured My Graph

→ If I talk about All possible way so best Approach is **Recursion**

→ I keep every Colors on the every Node



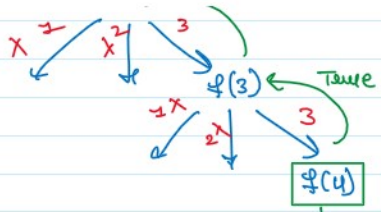
M=3



→ If we get a one possible path then No need to Required check another path.

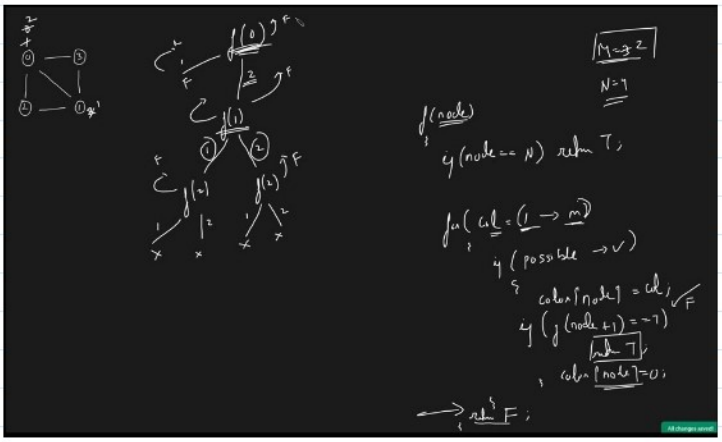
```
if (node) {
    if (node==n) {
```

```
if (node) {
    if (node == n) {
        return True;
    }
    for (1 → m) {
        if (possible to Assign Coloured) {
            Colour[Node] = colour;
        }
        if f(node+1) == True {
            return True;
        }
        Colour[Node] = 0;
    }
    return false;
}
```



→ Now this is the base case if I Reach the end of the Graph Node that means we Successfully Coloured All the Node

→ Return True



→ Because we try every colour on every Node

Time Complexity: O(N^M) (n raised to m)

Space Complexity: O(N)

→ used for recursion Space

```python
def isSafe(node, color, graph, n, col):
    for k in range(n):
        if k != node and graph[k][node] == 1 and color[k] == col:
            return False
    return True


2 usages
def coloreTheGraph(node, color, m, N, graph):
    if node == N:
        return True


    for i in range(1, m + 1):
        if isSafe(node, color, graph, N, i):
            color[node] = i
            if coloreTheGraph(node + 1, color, m, N, graph):
                return True
            color[node] = 0


    return False



def graphColoring(graph, m, N):
    color = [0] * N
    if coloreTheGraph(0, color, m, N, graph):
        return True
    return False
```


```java
public class L19_M_Coloring_Grapg {
    public static void main(String[] args) {

        System.out.println("M Coloring Problem");

    }
}

no usages
class Solution13 {
    no usages
    public boolean graphColoring(boolean graph[][], int m, int n) {

        int [] color= new int [n];

        if(coloreTheGraph(graph, n, node: 0, color, m)){
            return true;
        }

        return false;
    }


    2 usages
    private boolean coloreTheGraph(boolean graph[][], int n, int node, int [] color, int m){

        if(node==n){
            return true;
        }

        for(int i=1; i<=m; i++){
            if(isSafe(graph, n, node, color, i)){
                color[node]=i;
                if(coloreTheGraph(graph, n, node: node+1, color, m) == true){
                    return true;
                }
                color[node]=0;
            }
        }
        return false;
    }


    1 usage
    private boolean isSafe(boolean graph[][] , int n, int node, int [] color, int c){

        for(int i=0; i<n; i++){
            if(graph[node][i] == true && color[i]==c){
                return false;
            }
        }
        return true;
    }
}
```