Backtracking

Graph Coloring

⇒ **Problem :-** Let G be a graph and m be a given positive number. Problem is to find whether the nodes of G can be colored in such a way that no two adjacent nodes have the same color yet only m colors are used.
This is also termed as m-colorability decision problem.

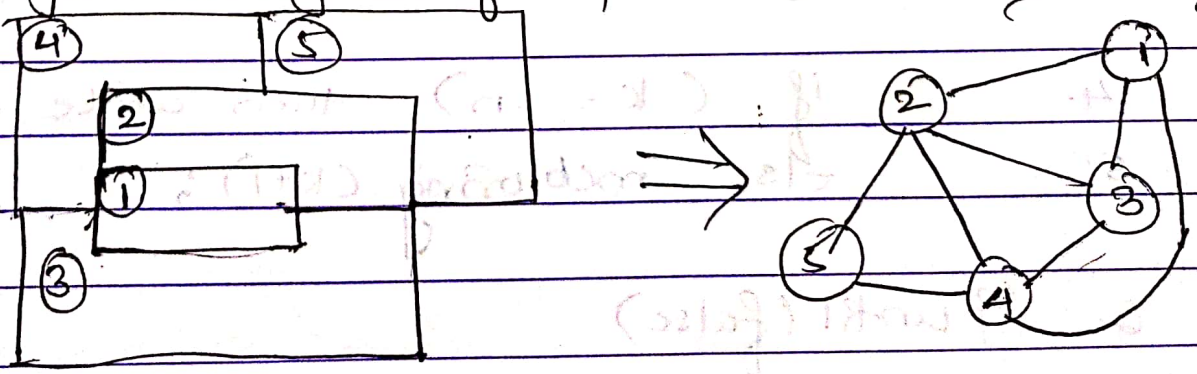
- If 'd' is the degree of graph, then it can be colored with 'd+1' colors.

- Problem is m-colorability optimization i.e problem asks for smallest integer 'm' for which graph G can be colored.
This smallest integer is referred to as 'CHROMATIC NUMBER'

⇒ **Application :-**

- Given any map, can the regions be colored in such a way that no two adjacent regions have the same color yet only 'm' colors are needed?
- A map can be transformed into planar graphs where each region of a map can be node and two adjacent regions of map are connected by an edge

eg.



⇒ Solution : Problem is determining all different ways in which a given graph can be colored using atmost m colors.

Let,

(i) Graph be represented as adjacency matrix.
G[1...n, 1...n] , where G[i,j] = 1 if (i,j) ∈ E
G[i,j] ≠ 0 if (i,j) ∉ E

(ii) Colors are represented by integers 1, 2, ... m

(iii) Solution is represented with n-tuple $(x_1, x_2 \dots x_n)$
where x_i – the color of ith node
n – No. of nodes.

⇒ Algorithm : mcoloring

I] Input : Index of next vertex to color
Output : using 'm' colors, all different combination in which a graph can be colored.

```
mcoloring (k)
{
1.  repeat  //do
    {
2.      x[k] = getNodeColor(k).   // 0....m
3.      if (x[k] == 0) return;

4.      if (k == n) then write (x[1...n])
5.      else mcoloring (k+1);
    }
6.  } until (false) //while (true)
```

ii) Algorithm : getNodeColor (k)

Assumption : x[1] .... x[k-1] have been assigned integer values in the range [1,m] such that values are distinct integers.

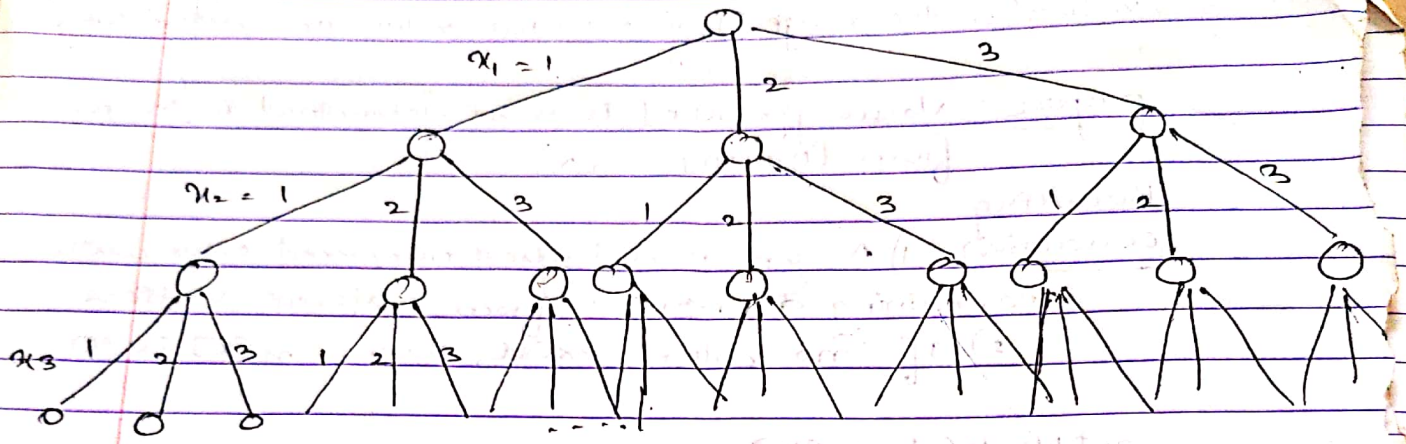
Purpose : Value for x[k] is to be determined in the range from [0....m]

Bounding condition : 1) Assign next highest numbered color while maintaining distinctness from adjacent vertices.
2) If no color exist, then x[k] is 0

```
getNodeColor (k)
{
1.  repeat
    {
2.      x[k] = (x[k]+1) mod (m+1)  // Next highest color
3.      if (x[k] == 0) then return x[k] // All colors used
4.      for j = 1 to k-1 do
5.          if (G[k][j] ≠ 0) and (x[k] == x[j])
6.              then break
7.      if (j == k) then return x[k]
8.  } until (false)
```

Analysis :-

State space tree for m-coloring when n=3 & m=3



- Each node at level i has 'm' children corresponding to m possible values of x_i, $1 \le i \le n$
- Node at the last level ie 'n' are leaf nodes.
- An upper bound on time complexity can be calculated by noticing that,

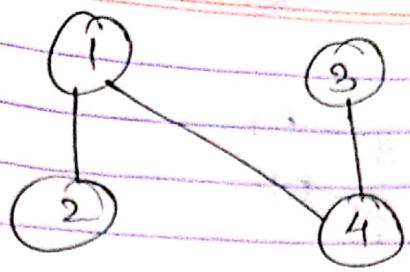
No. of internal nodes $= \sum_{i=0}^{n-1} m^i$

At each internal node, GetNewNode(color) takes O(mn) time to compute next legal color value.

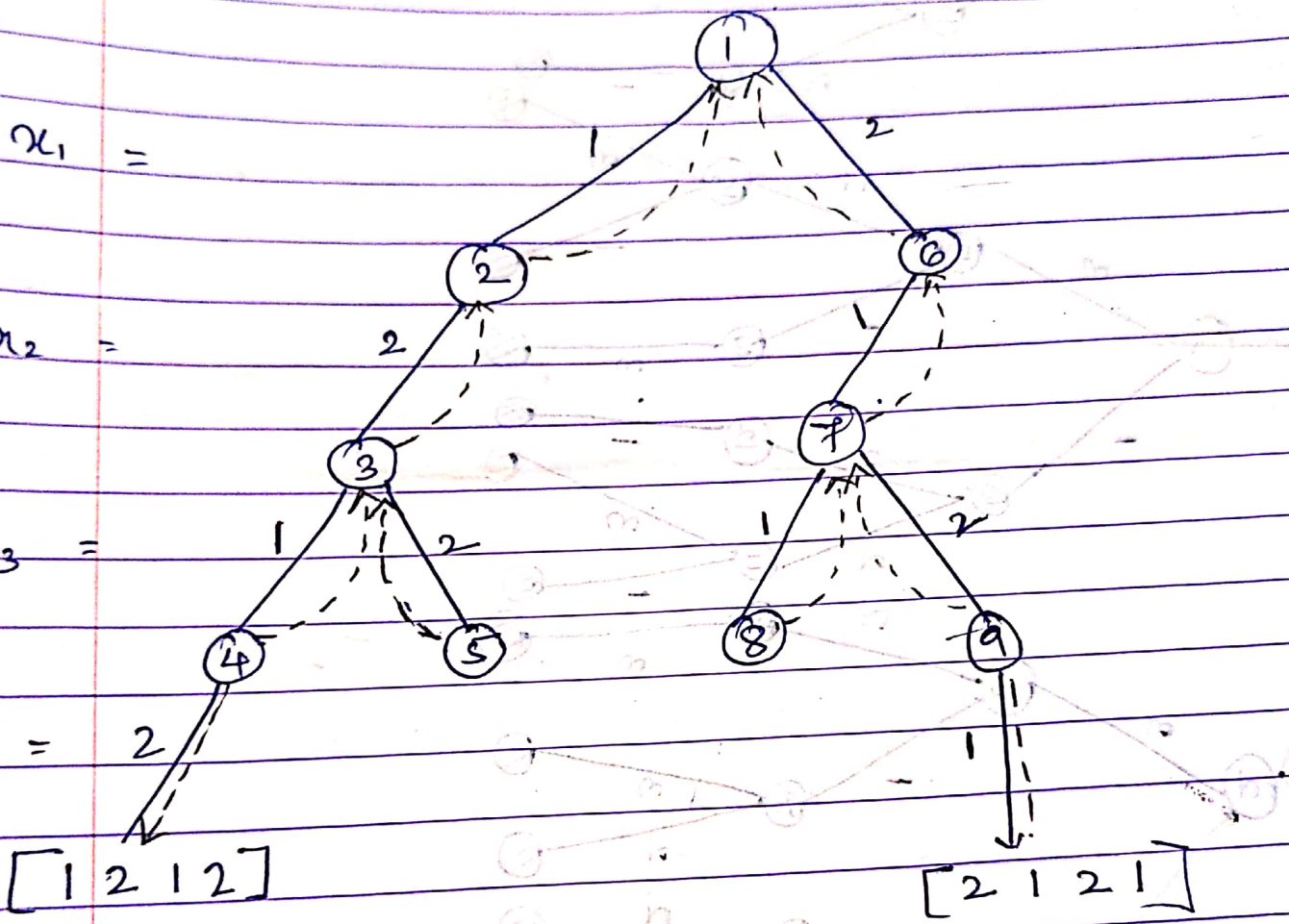
$\therefore$ Total time $= \sum_{i=0}^{n-1} m^i \times mn = \sum_{i=0}^{n-1} m^{i+1} \times n$

$= \sum_{i=1}^{n} m^i \times n = O(n \times m^n)$

9 Ex (1).



m = 2
n = 4



x_1 =

x_2 =

x_3 =

x_4 = 2

[1 2 1 2]

[2 1 2 1]