AIM :
Implement solution for a contraint satisfaction problem using branch and bond and backtracking for a graph coloring problem

---

Introduction

Constraint satisfaction problem

ApproachThe idea is to assign colors one by one to different vertices, starting from the vertex 0. Before assigning a color, check for safety by considering already assigned colors to the adjacent vertices i.e check if the adjacent vertices have the same color or not. If there is any color assignment that does not violate the conditions, mark the color assignment as part of the solution. If no assignment of color is possible then backtrack and return false.

Algorithm:

- Create a recursive function that takes the graph, current index, number of vertices, and output color array.
- if the current index is equal to the number of vertices. Print the color configuration in output array.
- Assign a color to a vertex (1 to m).
- For every assigned color, check if the configuration is safe, (i.e. check if the adjacent vertices do not have the same color) recursively call the function with next index and number of vertices
- If any recursive function returns true break the loop and return true.
- If no recursive function returns true then return false.

In [1]:

```
# Python program for solution of M Coloring
# problem using backtracking

class Graph():

    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]\
                              for row in range(vertices)]

    # A utility function to check
    # if the current color assignment
    # is safe for vertex v
    def isSafe(self, v, colour, c):
        for i in range(self.V):
            if self.graph[v][i] == 1 and colour[i] == c:
                return False
        return True

    # A recursive utility function to solve m
    # coloring  problem
    def graphColourUtil(self, m, colour, v):
        if v == self.V:
            return True

        for c in range(1, m + 1):
            if self.isSafe(v, colour, c) == True:
                colour[v] = c
                if self.graphColourUtil(m, colour, v + 1) == True:
                    return True
                colour[v] = 0

    def graphColouring(self, m):
        colour = [0] * self.V
        if self.graphColourUtil(m, colour, 0) == None:
            return False

        # Print the solution
        print ("Solution exist and Following are the assigned colours:")
        for c in colour:
            print (c,end=' ')
        return True

# Driver Code
g = Graph(4)
g.graph = [[0, 1, 1, 1], [1, 0, 1, 0], [1, 1, 0, 1], [1, 0, 1, 0]]
m = 3
g.graphColouring(m)
```

```
Solution exist and Following are the assigned colours:
1 2 3 2
```

Out[1]:

```
True
```