Graph Coloring Problem

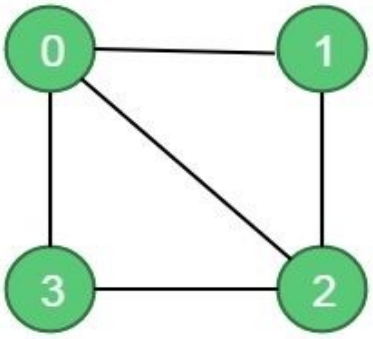
Let G be a graph and m be a given positive integer. We want to discover whether the nodes of G can be colored in such a way that no two adjacent nodes have the same color yet only m colors are used. This is termed the m-colorability decision problem. Note that if d is the degree of the given graph, then it can be colored with d + 1 colors. The m-colorability optimization problem asks for the smallest integer m for which the graph G can be colored. This integer is referred to as the chromatic number of the graph.

Example

In the following problem, an undirected graph is given. There is also provided m colors. The problem is to find if it is possible to assign nodes with m different colors, such that no two adjacent vertices of the graph are of the same colors. If the solution exists, then display which color is assigned on which vertex.

Starting from vertex 0, we will try to assign colors one by one to different nodes. But before assigning, we have to check whether the color is safe or not. A color is not safe whether adjacent vertices are containing the same color.

0	1	1	1
1	0	1	0
1	1	0	1
1	0	1	0



Let the maximum color m = 3.

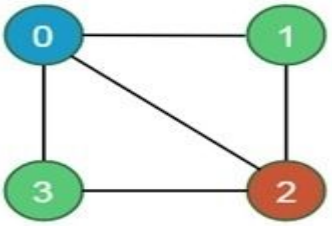
For this input the assigned colors are:

Node 0 -> color 1

Node 1 -> color 2

Node 2 -> color 3

Node 3 -> color 2



Algorithm

Algorithm mColoring(k)

```
// This algorithm was formed using the recursive backtracking
// schema. The graph represented by its boolean adjacency
// matrix G[1:n, 1:n].All assignments of 1,2,....,m to the
// vertices of the graph such that adjacent vertices are
// assigned distinct integers are printed. k is the index
// of the next vertex to color.
{
        repeat
       {//Generate all legal assignments for x[k].
              NextValue(k);                    // Assign to x[k] a legal color.
              if (x[k]= 0) then return;        // No new color possible
              if (k = n) then // At most m colors have been used to color the n
                                  vertices.
              write (x[1 :n]);
              else   mColoring(k + 1);
       }until(false);
}
```

Algorithm NextValue(k)

```
// x[1],...,x[k - 1]have been assigned integer values in
// the range [1,m] such that adjacent vertices have distinct
// integers. A value for x[k] is determined in the range
// [0,m]. x[k] is assigned the next highest numbered color
// while maintaining distinctness from the adjacent vertices
// of vertex k. If no such color exists, then x[k] is 0.
{
        repeat
        {
                x[k] :=(x[k]+ 1) mod (m+ 1);            // Next highest color.
                if (x[k]= 0) then return;               // All colors have been used.
                for j :=1 to n do
                {                       // Check if this color is distinct from adjacent colors.
                        if ((G[k, j]≠0) and (x[k]= x[j]))   // If (k,j) is an edge and if adj.
                                                            // vertices have the same color.
                                then break;
                }
                if (j = n + 1) then return;             // New color found
        }until (false);                         // Otherwise try to find another color.
}
```

Time Complexity

There are total O(m^V) combination of colors. So time complexity is O(m^V). The upper bound time complexity remains the same but the average time taken will be less.