Week 8:

Write a program to color the nodes in a given graph such that no two adjacent can have the same color using backtracking.

**Graph coloring problem:**

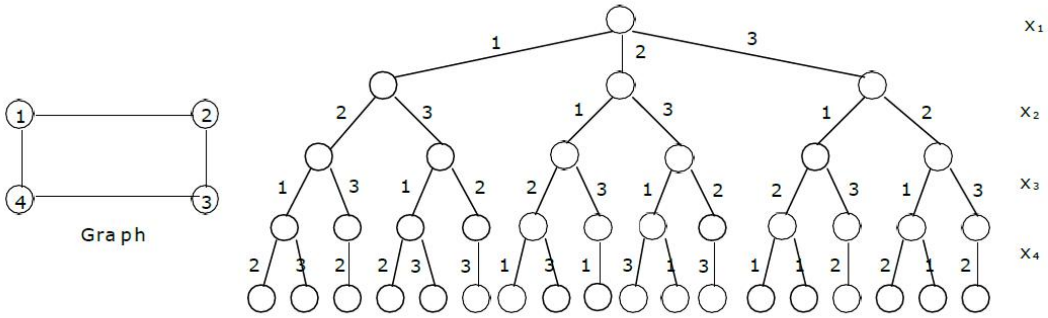
- ✓ Let G be a graph and m be a given positive integer.
- ✓ Find whether the nodes of G can be colored in such a way that no two adjacent nodes have the same color, yet only m colors are used.
- ✓ This is termed the m-colorabiltiy decision problem.
- ✓ The m-colorability optimization problem asks for the smallest integer m for which the graph G can be colored.
- ✓ The function m-coloring will begin by first assigning the graph to its adjacency matrix, setting the array x [] to zero.
- ✓ The colors are represented by the integers 1, 2, . . . , m and the solutions are given by the n-tuple ($x_1$, $x_2$, . . ., $x_n$), where $x_i$ is the color of node i.
- ✓ A recursive backtracking algorithm for graph coloring is carried out by invoking the statement mcoloring(1);

**Algorithm:**

```
Algorithm mColoring(k)
// This algorithm was formed using the recursive backtracking
// schema. The graph is 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 and 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.
}
```

Example:



A 4-node graph and all possible 3-colorings

**Sample Implementation:**

```
#include<stdio.h>
#include<conio.h>
int m,n;
int c=0;
int sol=0;
int g[10][10];
int x[10];
void nextvalue(int k);
void graphcoloring(int k);
void main()
{
   int i,j,t;
   printf("\n enter the number of vertices in a graph: " );
   scanf("%d", &n);
   printf("\n enter graph edges\n");
     for(i=1;i<=n;i++)
     {
        for(j=1;j<=n;j++)
        {
           scanf("%d",&g[i][j]);
        }
     }
   printf("\n All possible solutions are\n");
       for(m=1;m<=n;m++)
       {
          if(c==1)
            { break; }
          graphcoloring(1);
        }
       printf("\n chromatic number is %d",m-1);
       printf("\n total number of solutions is %d",sol);
getch();
}
void graphcoloring(int k)
{
int i;
      while(1)
      {
          nextvalue(k);
            if(x[k]==0)
```

```
            { return; }
           if(k==n)
            {
                 c=1;
                 for(i=1;i<=n;i++)
                    {    printf("%d    ",x[i]); }
                      sol++;
                      printf("\n");
                 }
                 else
         graphcoloring(k+1);
     }
}

void nextvalue(int k)
{
int j;
   while(1)
     {
       x[k]=(x[k]+1)%(m+1);
            if(x[k]==0)
            {
                 return;
               }
            for(j=1;j<=n;j++)
            {
                 if(g[k][j]==1&&x[k]==x[j])
                      break;
            }
            if(j==(n+1))
            {
                 return;
            }
      }
}
```

**Sample Output:**

```
enter the number of vertices in a graph: 4

enter graph edges

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

All possible solutions are

1 2 1 2
2 1 2 1

 chromatic number is 2
 total number of solutions is 2
```