7.4 GRAPH COLORING

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 and it is discussed again in Chapter 11. 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. For example, the graph of Figure 7.11 can be colored with three colors $1, 2$, and 3. The color of each node is indicated next to it. It can also be seen that three colors are needed to color this graph and hence this graph's chromatic number is 3.

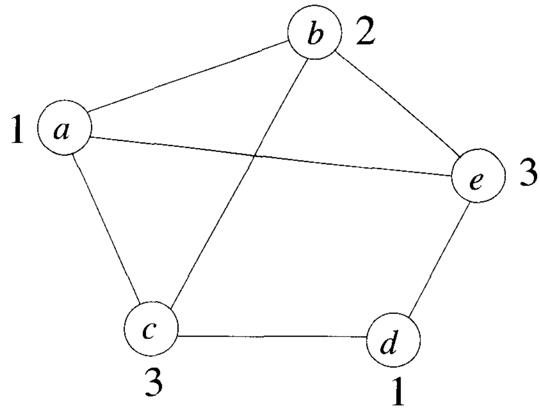

Figure 7.11 An example graph and its coloring

A graph is said to be *planar* iff it can be drawn in a plane in such a way that no two edges cross each other. A famous special case of the m-colorability decision problem is the 4-color problem for planar graphs. This problem asks the following question: given any map, can the regions be colored in such a way that no two adjacent regions have the same color yet only four colors are needed? This turns out to be a problem for which graphs are very useful, because a map can easily be transformed into a graph. Each region of the map becomes a node, and if two regions are adjacent, then the corresponding nodes are joined by an edge. Figure 7.12 shows a map with five regions and its corresponding graph. This map requires four colors. For many years it was known that five colors were sufficient to color any map, but no map that required more than four colors had ever been found. After several hundred years, this problem was solved by a group of mathematicians with the help of a computer. They showed that in fact four colors are sufficient. In this section we consider not only graphs that are produced from maps but all graphs. We are interested in determining all the different ways in which a given graph can be colored using at most m colors.

Suppose we represent a graph by its adjacency matrix $G[1:n, 1:n]$, where $G[i,j] = 1$ if (i,j) is an edge of G, and $G[i,j] = 0$ otherwise. The colors are represented by the integers $1, 2, \ldots, m$ and the solutions are given by the n-tuple $(x_1, \ldots, x_n)$, where x_i is the color of node i. Using the recursive backtracking formulation as given in Algorithm 7.1, the resulting algorithm is mColoring (Algorithm 7.7). The underlying state space tree used is a tree of degree m and height $n+1$. Each node at level i has m children corresponding to the m possible assignments to x_i, $1 \leq i \leq n$. Nodes at

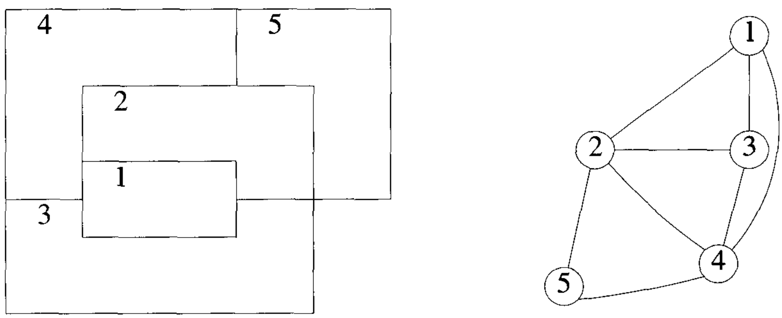

Figure 7.12 A map and its planar graph representation

level $n + 1$ are leaf nodes. Figure 7.13 shows the state space tree when $n = 3$ and $m = 3$.

Function mColoring is begun by first assigning the graph to its adjacency matrix, *setting the array* $x[\,]$ *to zero*, and then invoking the statement mColoring(1);.

Notice the similarity between this algorithm and the general form of the recursive backtracking schema of Algorithm 7.1. Function NextValue (Algorithm 7.8) produces the possible colors for x_k after x_1 through x_{k-1} have been defined. The main loop of mColoring repeatedly picks an element from the set of possibilities, assigns it to x_k, and then calls mColoring recursively. For instance, Figure 7.14 shows a simple graph containing four nodes. Below that is the tree that is generated by mColoring. Each path to a leaf represents a coloring using at most three colors. Note that only 12 solutions exist with *exactly* three colors. In this tree, after choosing $x_1 = 2$ and $x_2 = 1$, the possible choices for x_3 are 2 and 3. After choosing $x_1 = 2$, $x_2 = 1$, and $x_3 = 2$, possible values for x_4 are 1 and 3. And so on.

An upper bound on the computing time of mColoring can be arrived at by noticing that the number of internal nodes in the state space tree is $\sum_{i=0}^{n-1} m^i$. At each internal node, $O(mn)$ time is spent by NextValue to determine the children corresponding to legal colorings. Hence the total time is bounded by $\sum_{i=0}^{n-1} m^{i+1}n = \sum_{i=1}^{n} m^i n = n(m^{n+1} - 2)/(m-1) = O(nm^n)$.

```
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 7.7 Finding all m-colorings of a graph

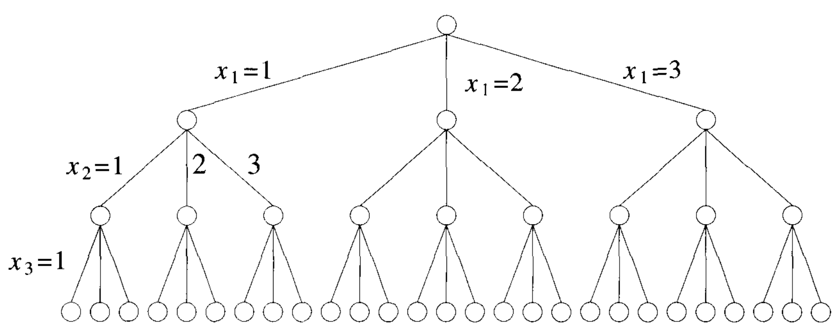

Figure 7.13 State space tree for mColoring when n =3 and m = 3

```
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.
}
```

Algorithm 7.8 Generating a next color

Reference :

Text Book-

E. Horowitz and S. Sahni: Fundamental of Computer Algorithms, Galgotia Pub. /Pitman, New Delhi/London, 1987/1978.