Graph Coloring Problem using Backtracking

GRAPH COLOURING PROBLEM

Let G be a graph and m be a positive integer .

The problem is to color the vertices of G using only m colors in such a way that no two adjacent nodes / vertices have the same color.

It is necessary to find the smallest integer m. m is referred to as the chromatic number of G.

GRAPH COLOURING PROBLEM (Contd..)

A map can be transformed into a graph by representing each region of map into a node and if two regions are adjacent, then the corresponding nodes are joined by an edge.

For many years it was known that 5 colors are required to color any map.

After a several hundred years, mathematicians with the help of a computer showed that 4 colours are sufficient.

Solving the Graph Colouring Problems

The graph is represented by its adjacency matrix Graph (1:n,1:n) where GRAPH (i,j) = true if <i,j> is an edge and Graph (i,j) = false otherwise.

The colours will be represented by the integers 1,2….m and the solution with n–tuple (X(1),….X(n)), where X(i) is the colour of node i.

Solving the Graph Colouring Problems (Contd..)

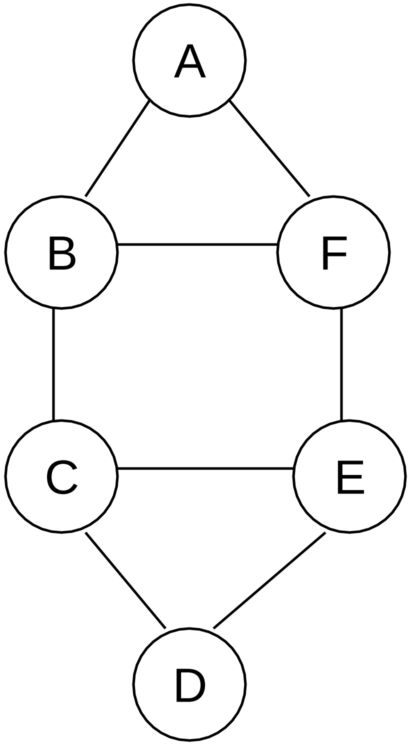
The solution can be represented as a state space tree.

Each node at level i has m children corresponding to m possible assignments to X(i) $1 \le i \le m$.

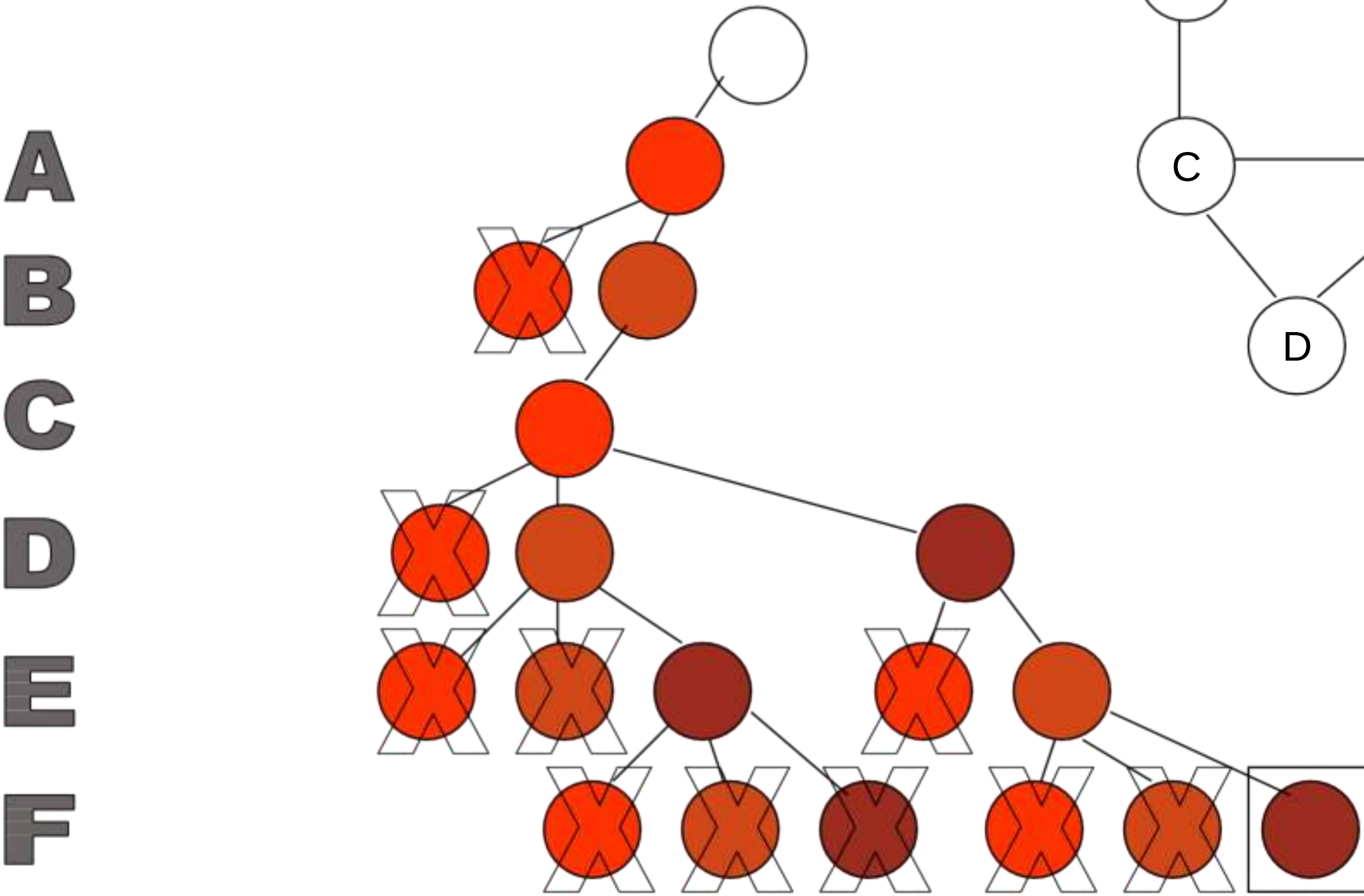
Nodes at level n+1, are leaf nodes. The tree has degree m with height n+1.

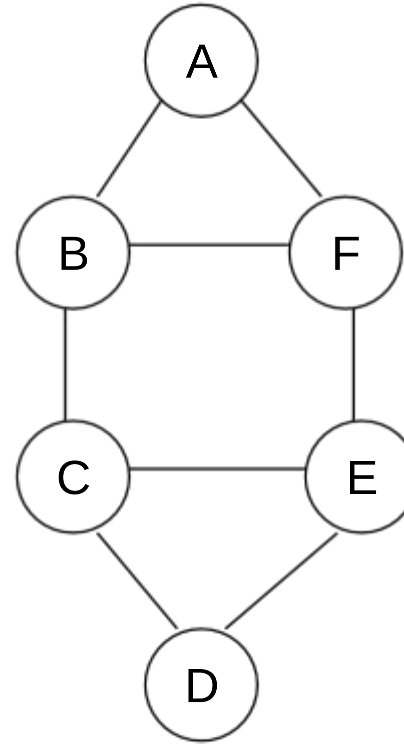
Graph Coloring

- As an example:
  - The vertices are enumerated in order A-F
  - The 3 colors to be used.

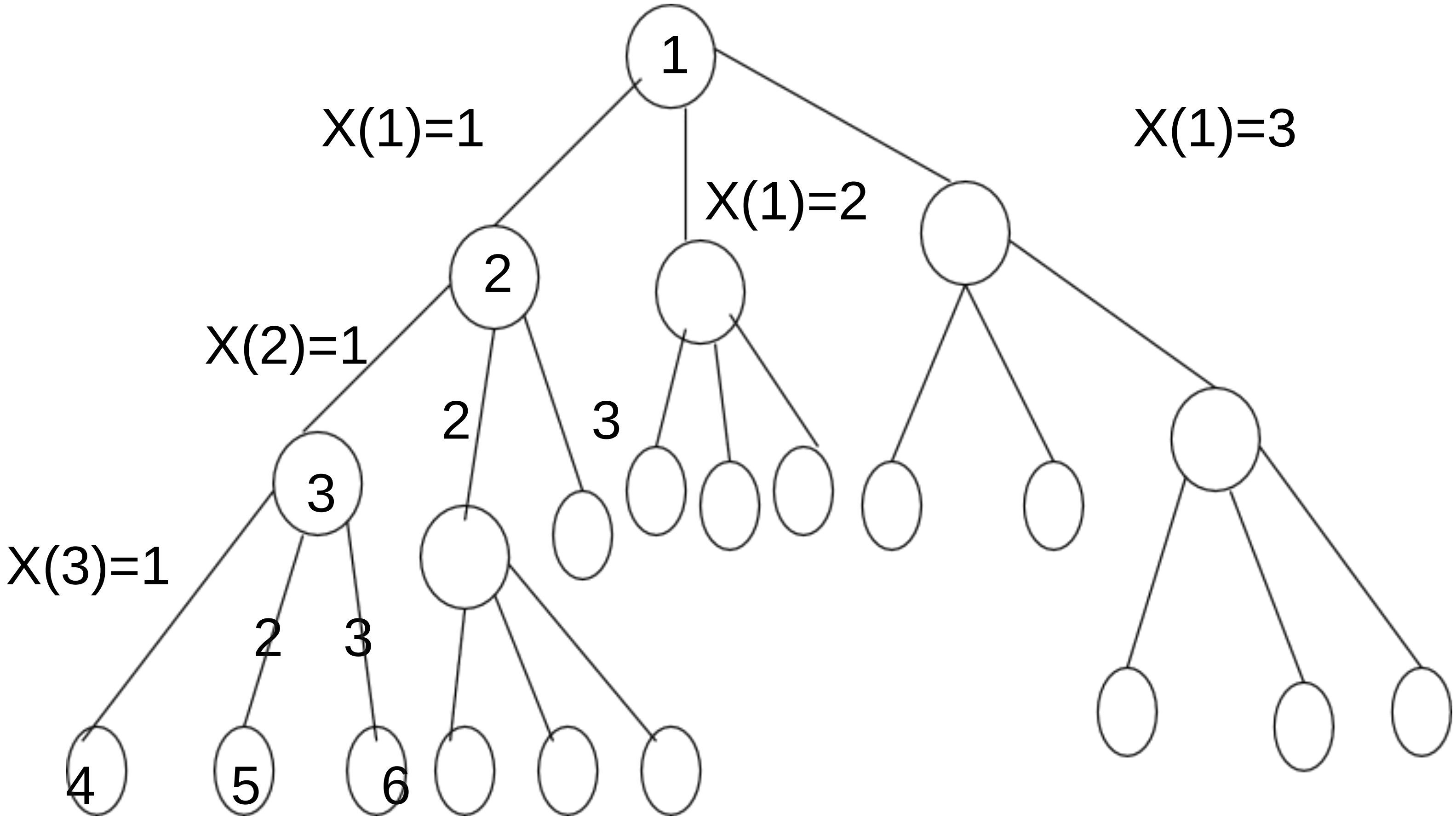


Graph Coloring

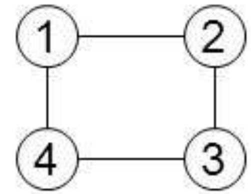


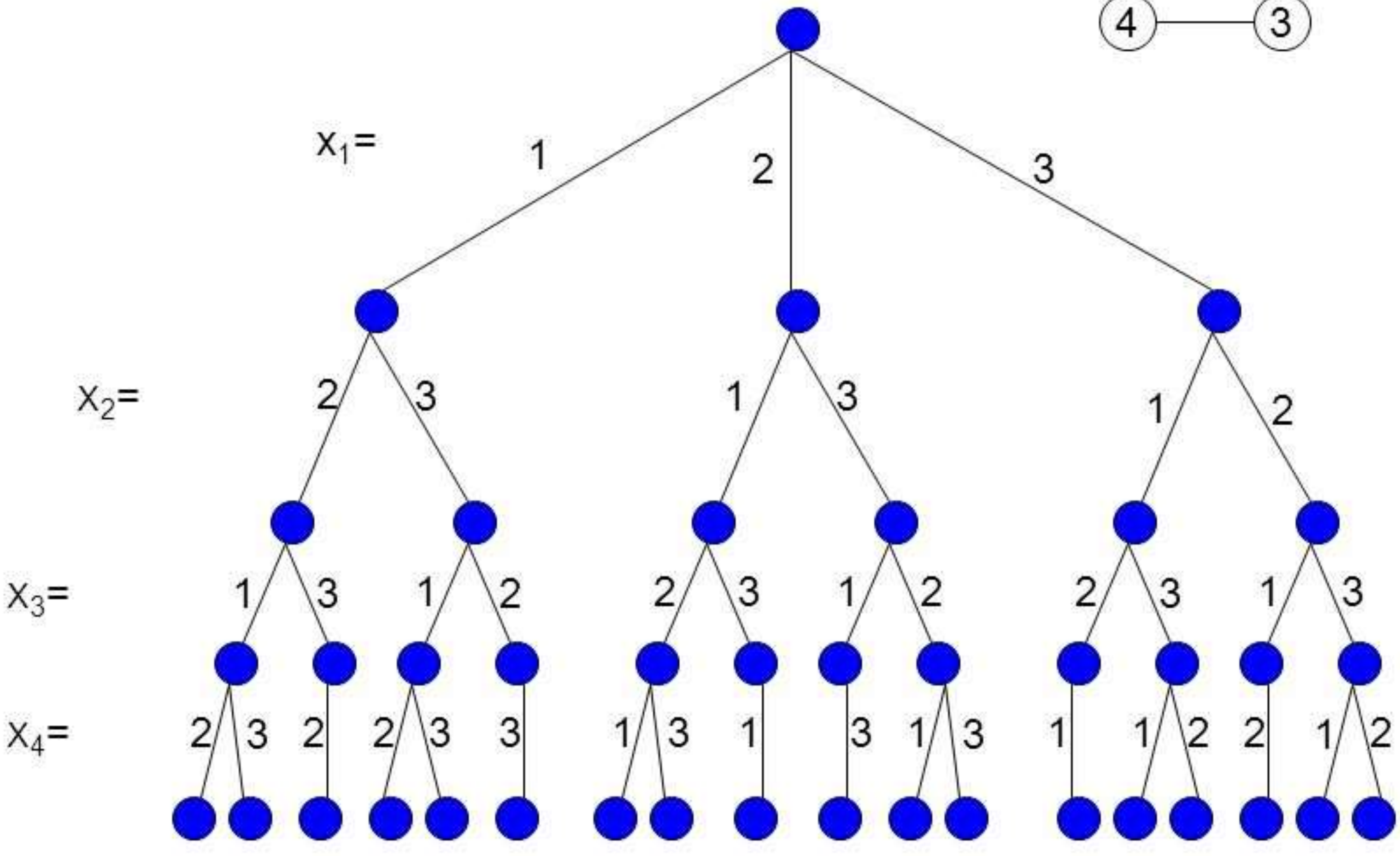


State space tree for m colouring problem with n = 4 and m = 3



A 4-node graph and all possible 3-colorings





Graph Colouring Problem- 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);
}
```

Graph Colouring Problem- Algorithm (Cont…)

```
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.
}
```

Time complexity

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 $\sum_{i=0}^{n-1} m^{i+1}n = \sum_{i=1}^{n} m^i n = n(m^{n+1} - 2)/(m - 1) = O(nm^n)$.