# DESIGN & ANALYSIS OF ALGORITHMS

## Session –22

# Queen Placement on a 4x4 board

Queens on a chessboard can travel laterally, vertically and diagonally. So, we have to place queens on the board in such a manner that no two queens attack each other. Each row of the chessboard will have one queen each.

| | | | | |
|---|---|---|---|---|
| Row – 1 | **Q1** | -- | -- | -- |
| Row – 2 | -- | -- | | |
| Row – 3 | -- | | -- | |
| Row – 4 | -- | | | -- |

**(Path of a queen on a 4x4 board)**

So, Can we place 4 Queens in this board?

| | | | |
|---|---|---|---|
| Q1 | -- | -- | -- |
| -- | -- | | |
| -- | | -- | |
| -- | | | -- |

# Let's Try

| | | | |
|---|---|---|---|
| Q1 | -- -- | -- -- | -- -- |
| -- -- | -- -- | Q2 | -- |
| -- | -- | -- -- | -- |
| -- -- | | -- | -- |

After placing Q1 and Q2, there is no place for Q3 in row 3

# Let's Try

| | | | |
|---|---|---|---|
| -- | Q1 | -- | -- |
| -- | -- | -- | |
| | -- | | -- |
| | -- | | |

We go back and change the place of Q1. Now, we have place for Q2.

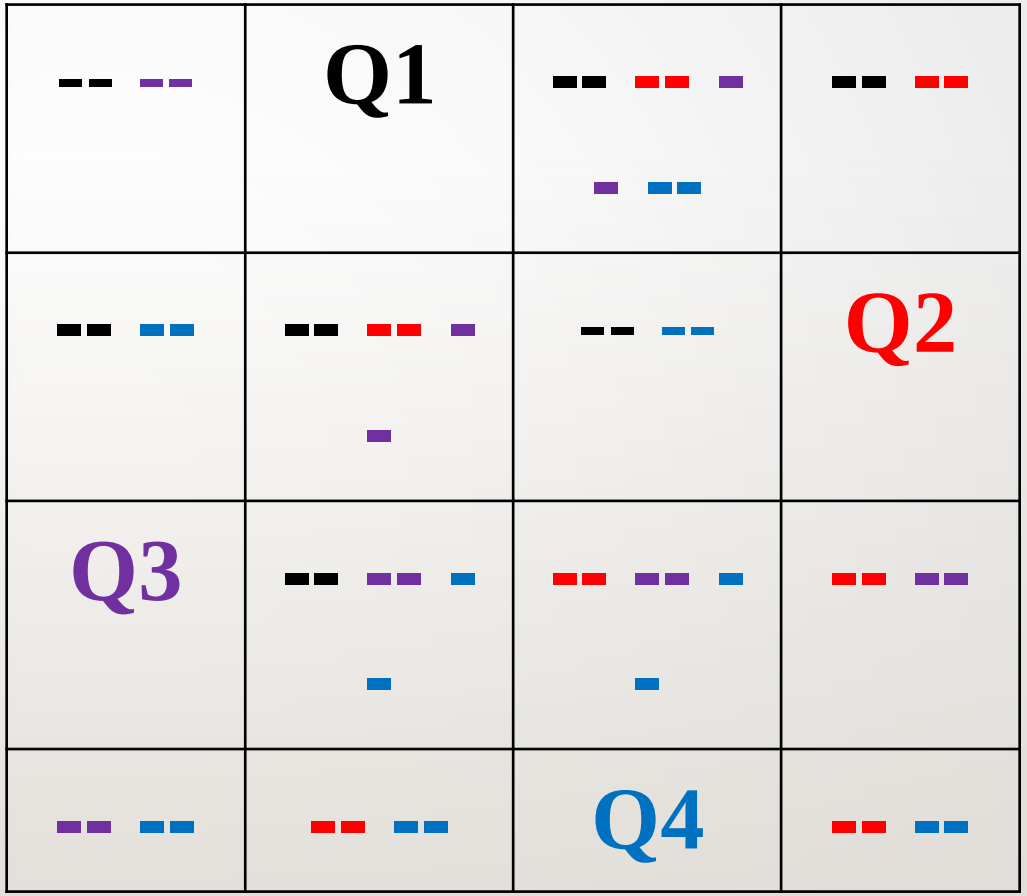
# Let's Try

| | | | |
|---|---|---|---|
| -- | Q1 | -- -- | -- -- |
| -- -- | -- -- | -- -- | Q2 |
| | -- | -- | -- |
| | -- -- | | -- |

After placing Q1 and Q2, there is place both for Q3 and Q4

## So, Can we place 4 Queens within the board?

YES, we can. If we just go back, move Q1 right by 1 position and we find that all the queens have been placed within the board.



**This is not the only solution. We have more than 1 solution**

Try and Let me know the other solutions

| | | | |
|---|---|---|---|
| | Q1 | | |
| | | | Q2 |
| Q3 | | | |
| | | Q4 | |

# Our methodology to solve 4-Queen problem

- Systemetically search for a Solution to a problem

- Build the solution one at a time
- If we reach at a dead-end i.e., the solution leads to infeasibility
  - Undo the last step
  - Try the next option

# Backtracking methodology

- Many problems which deal with searching for a set of solutions or for an optimal solution satisfying some constraints can be solved using the backtracking formulation.

- In Backtracking method, the desired solution is expressed as an n-tuple vector : $(x_1 \ldots x_i \ldots x_n)$ where $x_i$ is chosen from some finite set $S_i$.

- The problem is to find a vector, which maximizes or minimizes a criterion function $P(x_1 \ldots x_i \ldots x_n)$.

- The major advantage of this method is, once we know that a partial vector$(x_1, \ldots x_i)$ will not lead to an optimal solution then the generation of remaining components i.e.,$(x_{i+1} \ldots x_n)$ are ignored entirely.

# Backtracking methodology

- Many problems solved using backtracking require that all the solutions satisfy a complex set of constraint.

- The constraints are classified as:
  - Explicit
  - Implicit

# Backtracking methodology Explicit Constraints

- Explicit constraints are rules that restrict each $x_i$ takes values from a given set $S_i$.

- All solutions to the 4-queens problem can therefore be represented as 4-tuple $(x_1, x_2, x_3, x_4)$, where $x_i$ is the column on which queen $Q_i$ is placed.

  Ex: $S_i = \{1, 2, 3, 4\}$ where $1 \leq i \leq 4$

- The explicit constraints depend on the particular instance I of the problem being solved.

  Ex: The placement of $Q_2$ depend upon the placement of $Q_1$

All tuples that satisfy the explicit constraints define a possible solution space for I. The solution space, therefore, consists of $4^4$ (= 256) 4-tuples.

- All tuples that satisfy the explicit constraints define a possible solution space for I.

  Ex: $I_1 = \{x_2, x_4, x_1, x_3\}$

# Backtracking methodology Implicit Constraints

- The implicit constraints are rules that determine which of the tuples in the solution space I satisfy the criterion function.
- Thus, implicit constraints describe the way in which the $x_i$ must relate to each other.

Ex: (a) No two $x_i$'s can be the same (i.e. all queens must be on different columns)

(b) No two queens on the same diagonal

(c) No two queens on the same row

The constraints (a) and (b) implies that all solutions are permutations of the 4-tuple (1, 2, 3, 4).

This realization reduces the size of the olution space from $4^4$ to 4! tuples.

# STATE SPACE TREE TERMINOLOGY

1. **Criterion function:** It is a function $P(x_1, x_2 \ldots x_n)$ which needs to be maximized or minimized for a given problem.
2. **Solution Space:** All tuples that satisfy the explicit constraints define a possible solution space for a particular instance 'I' of the problem
3. **Problem state:** Each node in the tree organization defines a problem state.
4. **Solution States:** These are those problem states S for which the path form the root to S define a tuple in the solution space.
5. **State space tree:** If we represent solution space in the form of a tree then the tree is referred as the state space tree.
6. **Answer States:** These solution states S for which the path from the root to S defines a tuple which is a member of the set of solution (i.e. it satisfies the implicit constraints) of the problem.

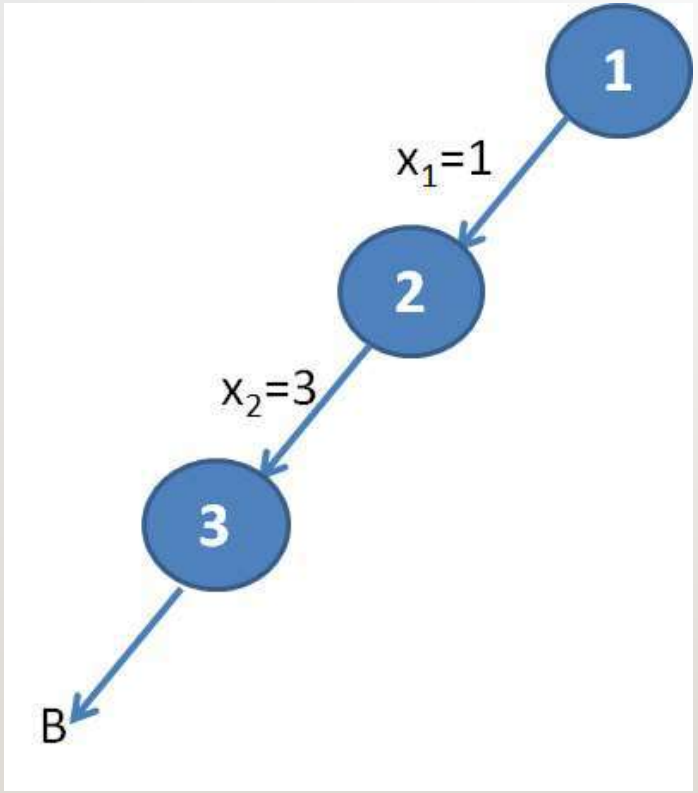
# STATE SPACE TREE TERMINOLOGY

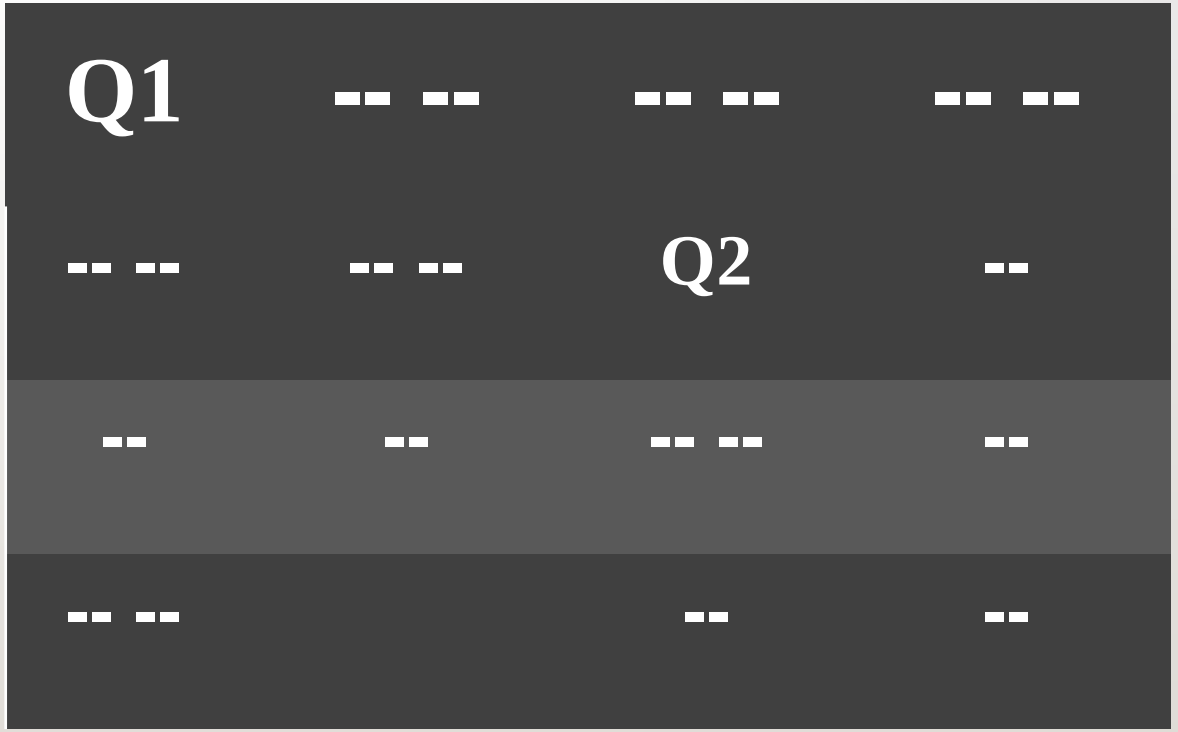
7. **Live node**: A node which has been generated and all of whose children have not yet been generated is live node.
8. **E-node:** The live nodes whose children are currently being generated is called E-node (node being expanded)
9. **Dead node:** It is a generated node that is either not to be expanded further or one for which all of its children has been generated.
10. **Bounding function:** It will be used to kill live nodes without generating all their children
11. **Branch and bound:** It is a method in which E-node remains E-node until it is dead

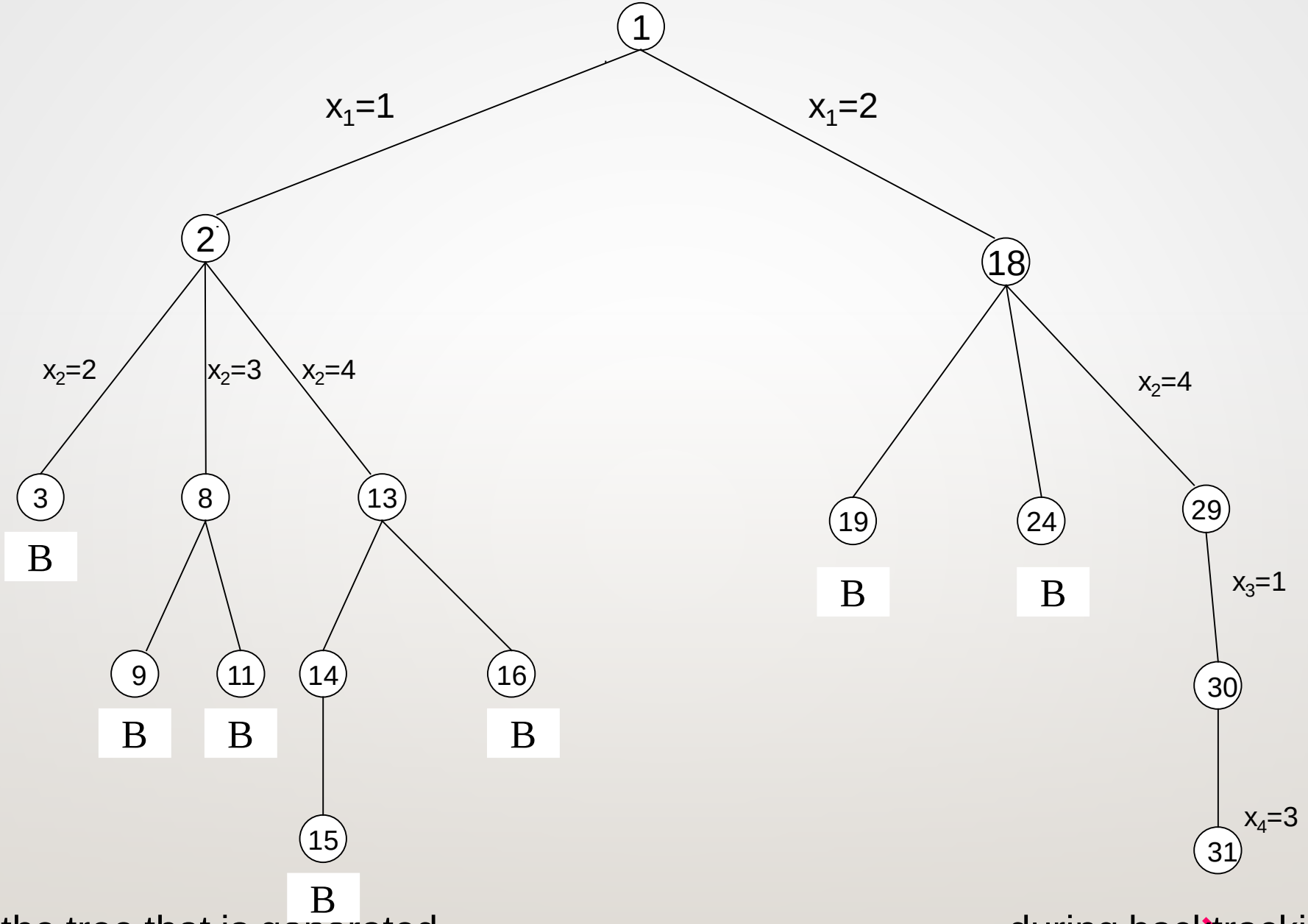
# Backtracking methodology

## Solution Space Tree

- Backtracking algorithms determine problem solutions by systematically searching the solution space for the given problem instance. This search is facilitated by using a tree organization for the solution space.
- For a given solution space many tree organizations may be possible.

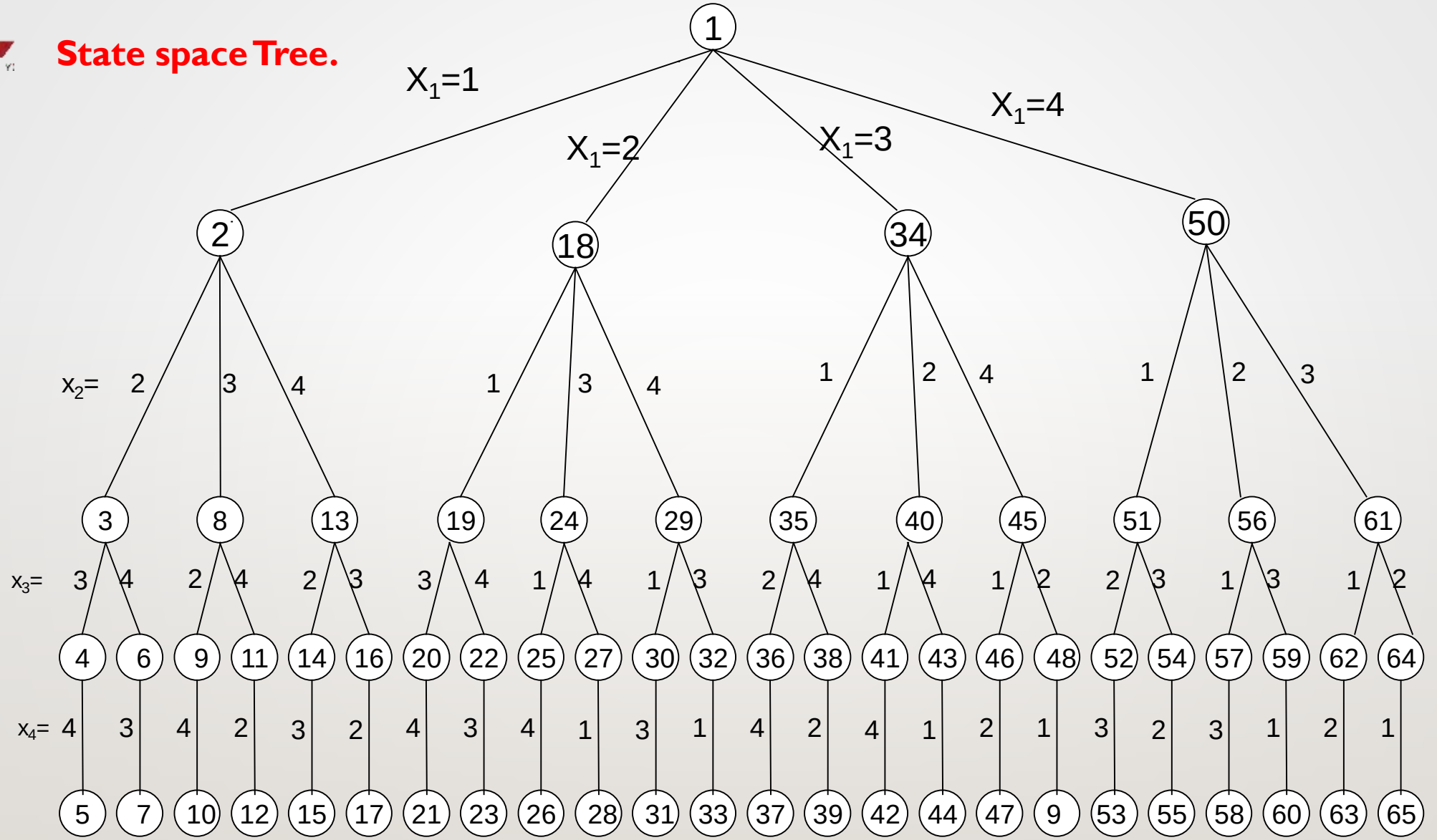





Portion of the tree that is generated during backtracking( n=4 ).

# State space Tree.




Tree organization of the 4-queens solution space. Nodes are numbered as in **depth first search.**

# Iterative Backtracking Algorithm

```
Backtrack (n)
// This schema describes the backtracking process .
// All solutions are generated in x[1:n] and printed as soon as //they are determined
{
      k = 1;
      while ( k ≠ 0 ) do
      {
           if (there remains an untried x[k] ∈ T(x[1], x[2], …, x[k-
1]) and  B_k(x[1],x[2],…,x[k]) is true)
           then
           {
                    if (x[1],x[2],…,x[k] is a path to an answer node) then
                         write (x[1 : k]);
            }
            else
                              k = k – 1;    }
}
```

# Recursive Backtracking Algorithm

```
Backtrack (k)
// This schema describes the backtracking process using recursion.
// On entering the first k – 1 values x[1], x[2], …, x[k – 1 ] of the solution vector
// x[1:n] have been assigned.
// x[] and n are global
{
     for( each x[k] ∈ T(x[1], … x[k – 1]) do
    {
        if (B_k(x[1], x[2], …, x[k]) ≠ 0) then
       {
              if(x[1],x[2],…,x[k]) is a path to an answer node) then write (x[1:k]);
              if(k < n) then Backtrack(k+1);
         }
    }
}
```

# APPLICATIONS OF BACKTRACKING

1. Producing all permutations of a set of values
2. Parsing languages
3. Games: anagrams, crosswords, word jumbles, 8 queens
4. Combinatory and logic programming

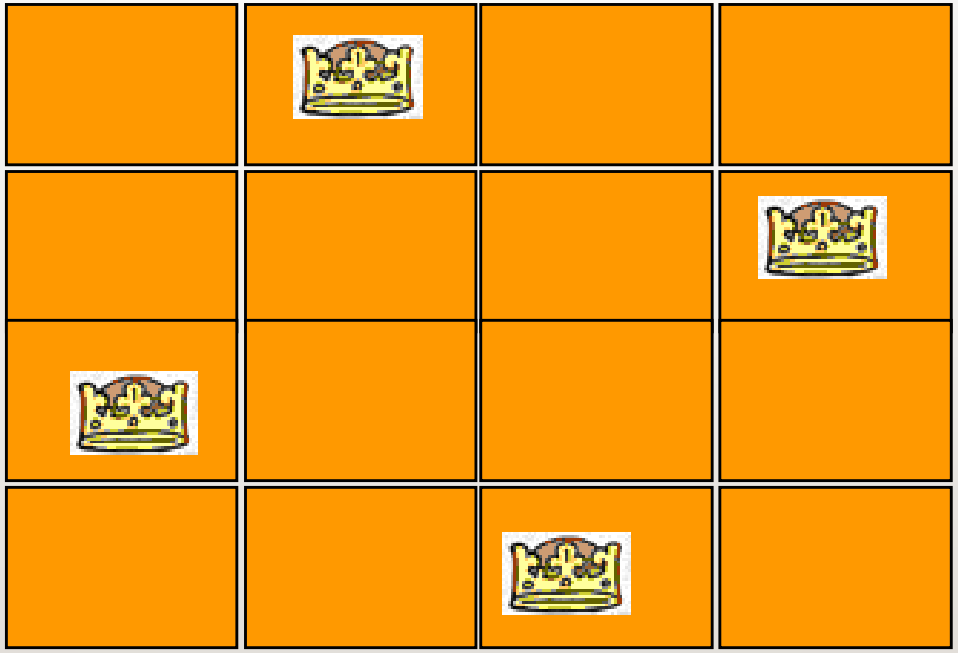
## Example Problems

1. N queen's problem
2. Sum of subsets problem.
3. Graph coloring problem
4. 0/1 knapsack problem
5. Hamiltonian cycle problem

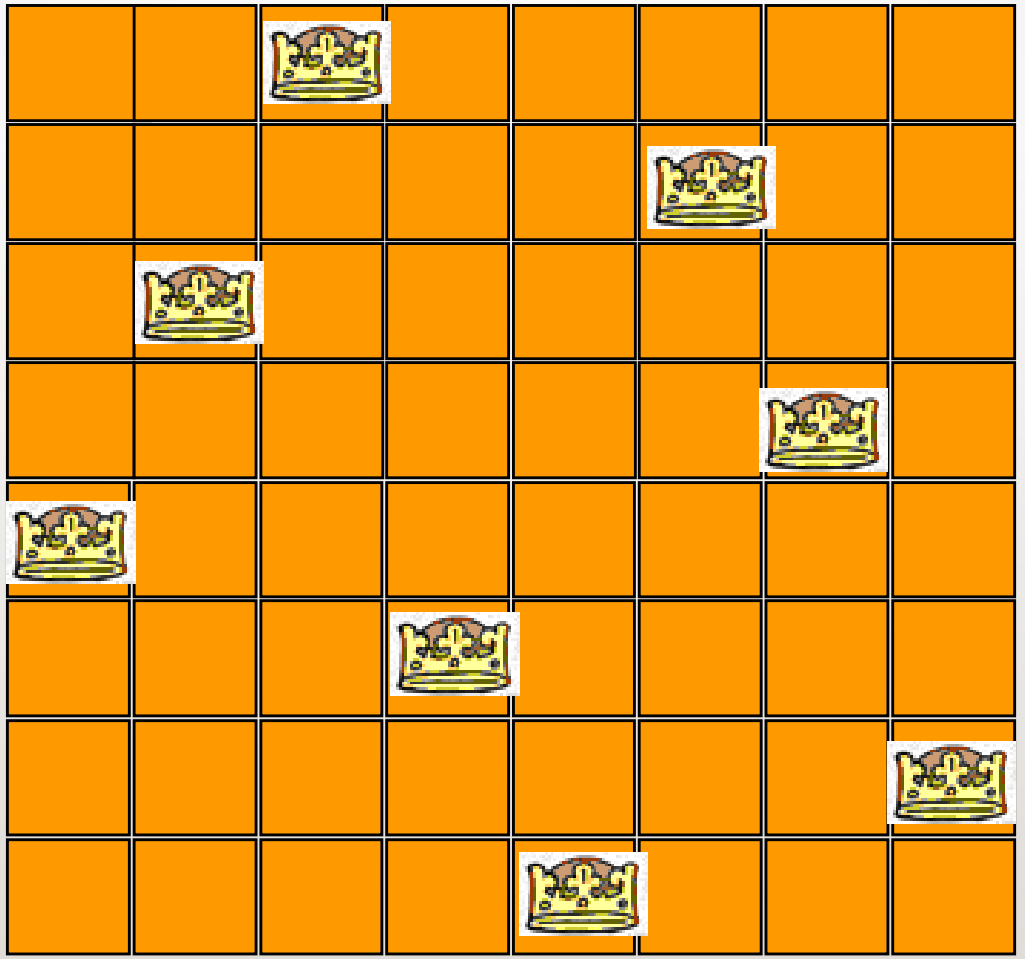
**In summary, backtracking consists of**
**Doing a depth-first search of a state space tree by applying boundary conditions.**

# 4-QUEENS PROBLEM


Can n queens be placed on an n x n chessboard so that no queen may attack another queen?



# 8-QUEENS PROBLEM



# • n– queens : Defining the problem:-

- The problem is to place n queens on an n x n chessboard so that no two queens "attack" i.e.no two queens on the same row, column, or diagonal.
- Assume rows and columns of chessboard are numbered 1 through n. Queens also be numbered 1 through n.
- **Condition to ensure - No two queens on same Row**: Each queen must be on a different row ,hence queen i is to be placed on row i. Therefore, all solutions to the n-queens problem can be represented as n-tuples ( $x_1, x_2, \ldots . x_n$), where $x_i$ is the column on which queen i is placed.
- **Condition to ensure - No two queens on same Column**:
- Select a distinct value for each $x_i$

# Condition to ensure - No two queens on same Diagonal

If two queens are placed at positions ( i, j ) and ( k, l ). They are on the same diagonal then following conditions hold good.

1) Every element on the same diagonal that runs from the upper left to the lower right has the same row – column value.

$$i - j = k - l \text{ -----(1)}$$

2) Similarly, every element on the same diagonal that goes from the upper right to the lower left has the same row + column value.

$$i + j = k + l \text{ -----(2)}$$

- First equation implies : $j - l = i - k$

  Second equation implies : $j - l = k - i$

- Therefore, two queens lie on the same diagonal iff $|j - l| = |i - k|$

```
Algorithm place( k,  i )
// It returns true if a queen can be placed in kth row and ith
// column . Otherwise it returns false.  x[] is a goal array whose
// first( k-1) values have been set. Abs(r) returns the absolute value of
   r.
   {
        for j = 1 to k-1 do
        {          // Two in the same column  or in the same diagonal
             if (  ( x [ i ] = i )   or    ( abs( x[ j ] - i ) = abs( j - k ) )
   )  then
                  return false;
        }
    return true ;
   }
```

```
Algorithm Nqueen(k, n)
// Using backtracking,  this procedure prints all possible placements
// of n queens on an n×n chessboard so that they are non-attacking.
{
        for i = 1 to n do        // check place of column for queen k
        {
                if place( k, i ) then
                {
                        x[ k ] = i;
                        if( k = n ) then write ( x[1:n] );
                        else  Nqueen( k+1, n);
                }
        }

}
```

# GRAPH COLORING

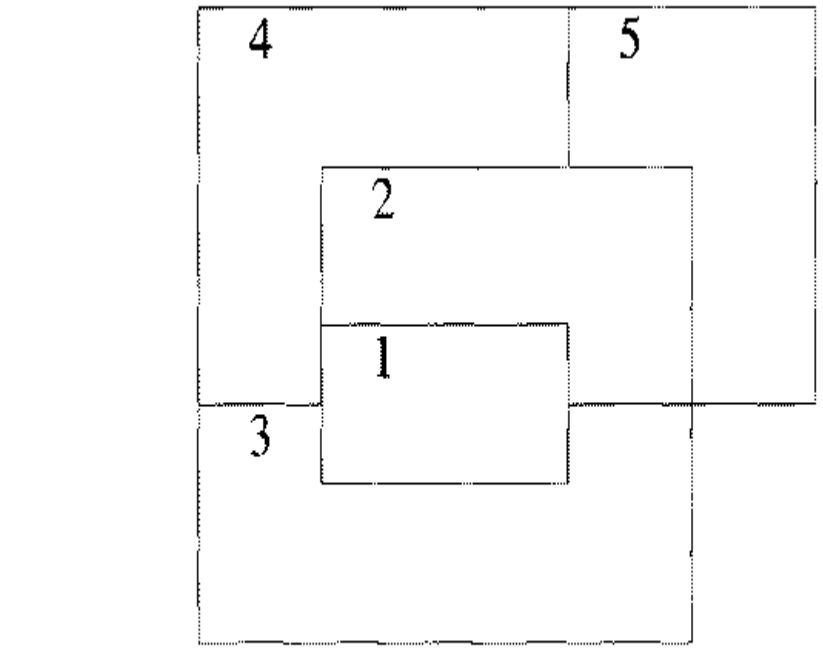
- Let ‘G’ be a graph and ‘m’ be a given positive integer. If the nodes of ‘G’ can be colored in such a way that no two adjacent nodes have the same color. Yet only ‘M’ colors are used. So it’s called M-color ability decision problem.

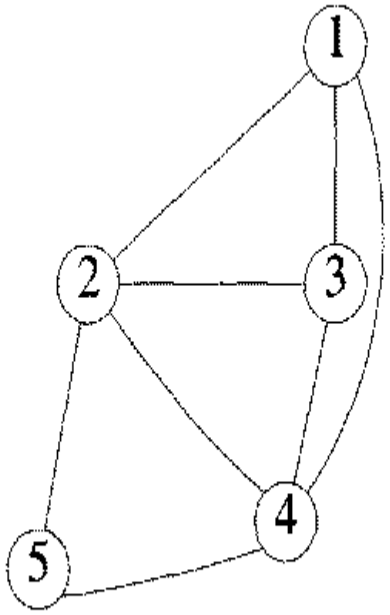
- The graph G can be colored using the smallest integer ‘m’. This integer is referred to as chromatic number of the graph.

- A graph is said to be planar iff it can be drawn on plane in such a way that no two edges cross each other.

- Suppose we are given a map then, we have to convert it into planar. Consider each and every region as a node. If two regions are adjacent then the corresponding nodes are joined by an edge.

# A map and its planar graph representation



# STEPS TO COLOR THE GRAPH:

- First create the adjacency matrix graph(1:m,1:n) for a graph, if there is an edge between i,j then C(i,j) = 1 otherwise C(i,j) =0.

- The Colors will be represented by the integers 1,2,…..m and the solutions will be stored in the array X(1),X(2),……..,X(n) ,X(index) is the color, index is the node.

- He formula is used to set the color is,
  X(k) = (X(k)+1) % (m+1)

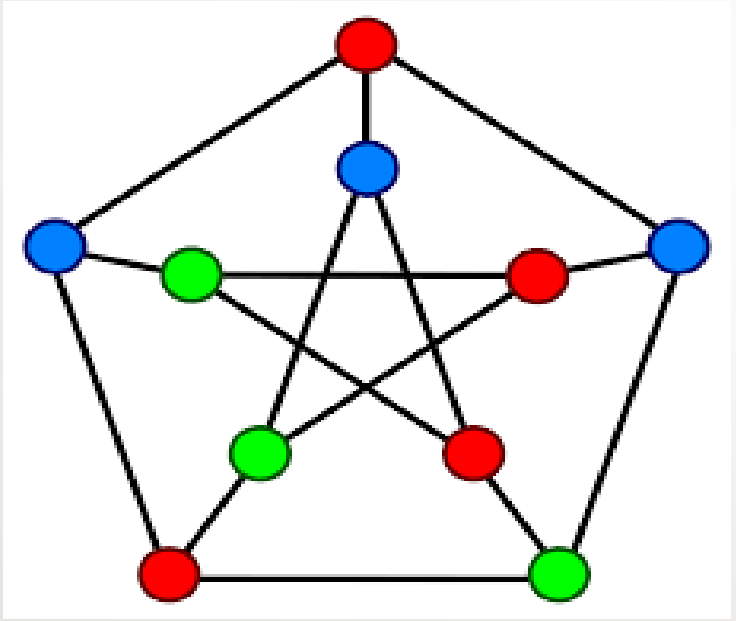
- First one chromatic number is assigned ,after assigning a number for ‘k’ node, we have to check whether the adjacent nodes has got the same values if so then we have to assign the next value.

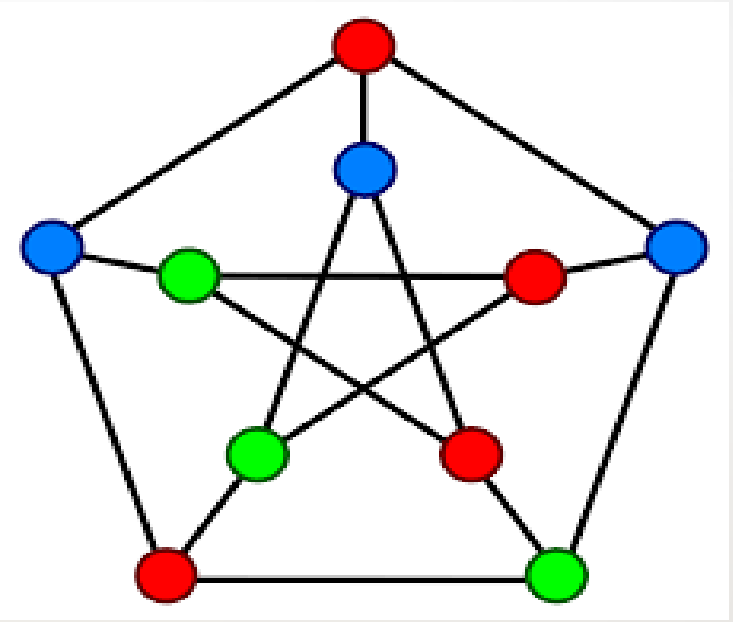
- Repeat the procedure until all possible combinations of colors are found.

- The function which is used to check the adjacent nodes and same color is,

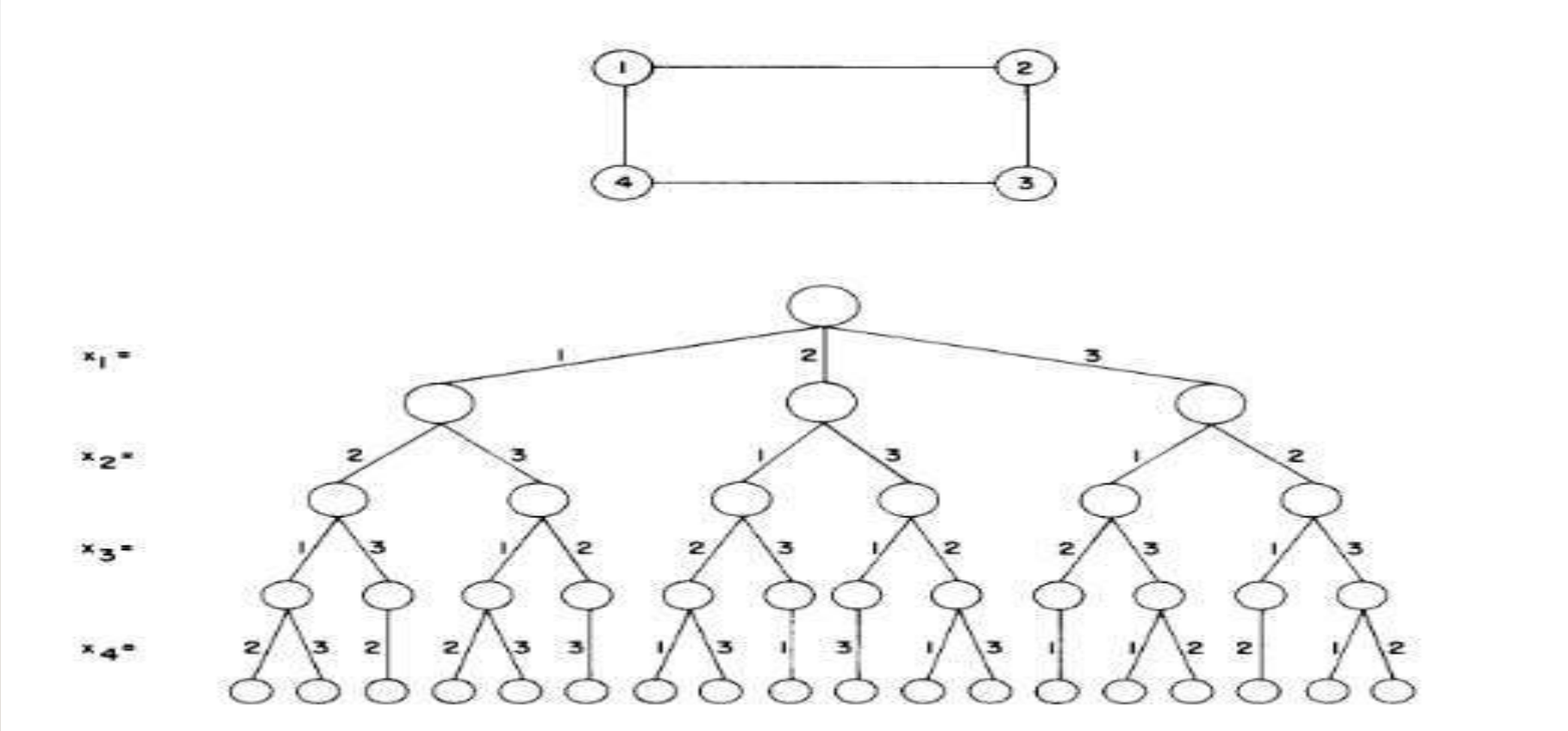
If(( Graph (k,j) == 1) and X(k) = X(j))

# GRAPH COLORING EXAMPLE

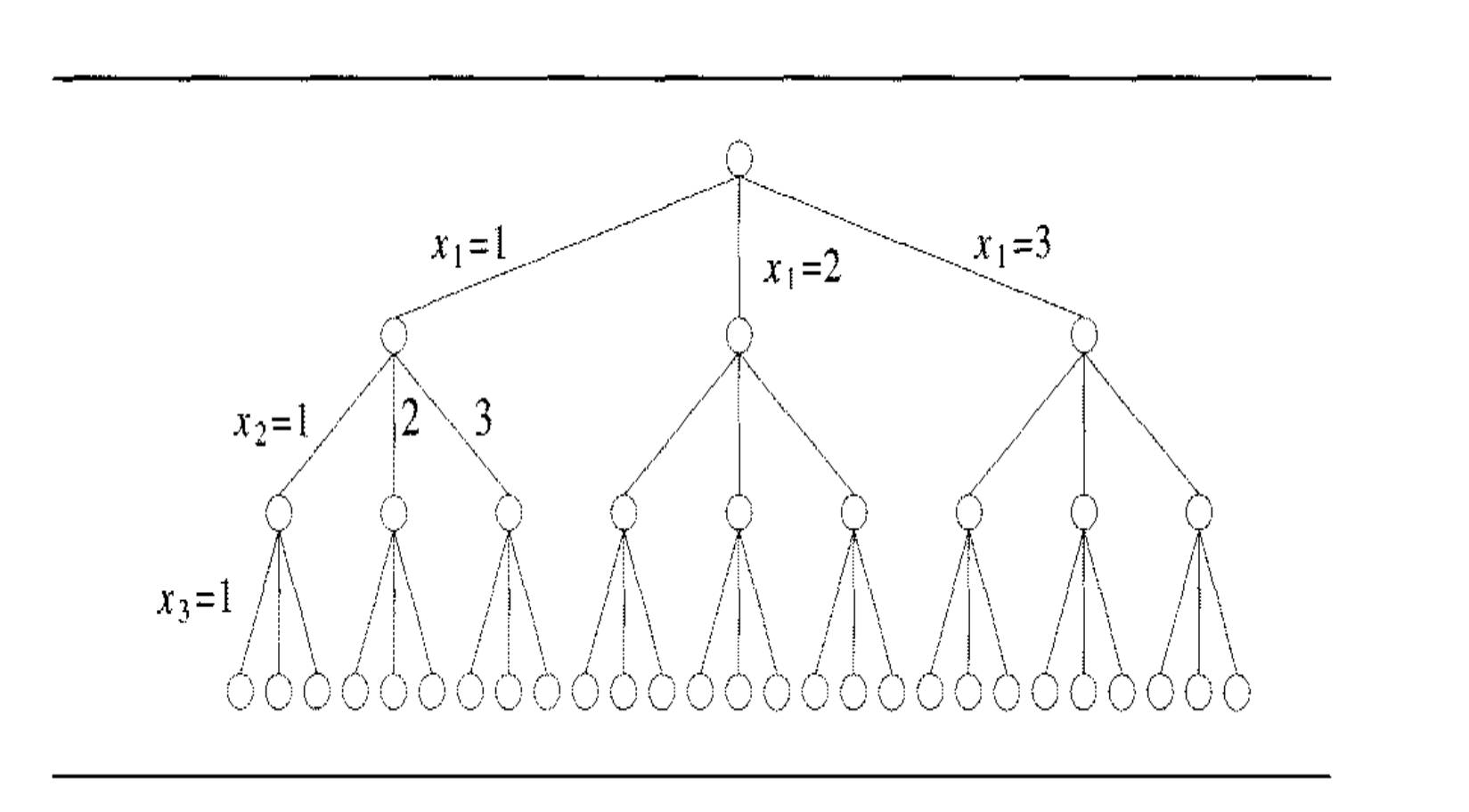




# A 4 – NODE GRAPH AND ALL POSSIBLE 3 COLORINGS



# STATE SPACE TREE FOR MCOLORING WHEN N=3 AND M=3



## Algorithm mColoring(k)

```
// 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 assignment 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 // Almost '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.
}
```

# TIME COMPLEXITY:

- An upper bound on the computing time of **mColoring** can be derived 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:

$$\sum_{i=0}^{n-1} m^{i+1} n$$

$$\sum_{i=0}^{n-1} m^i n$$

$$n(m^{n+1} - 2)/(m-1) = O(nm^n)$$

# SAMPLE QUESTIONS

- What is the Eight Queens problem? Explain its objective and constraints.
- Describe the rules and constraints that govern the placement of eight queens on an 8x8 chessboard.
- Explain the concept of backtracking
- What is goal state