Advanced Data Structures and Algorithms

BACKTRACKING

Dr G.Kalyani

Department of Information Technology

Velagapudi Ramakrishna Siddhartha Engineering College

Topics

- General Method
- N-Queens Problem
- Sum of Subsets
- Graph Coloring
- Hamiltonian Cycles.

Backtracking

- 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 :**

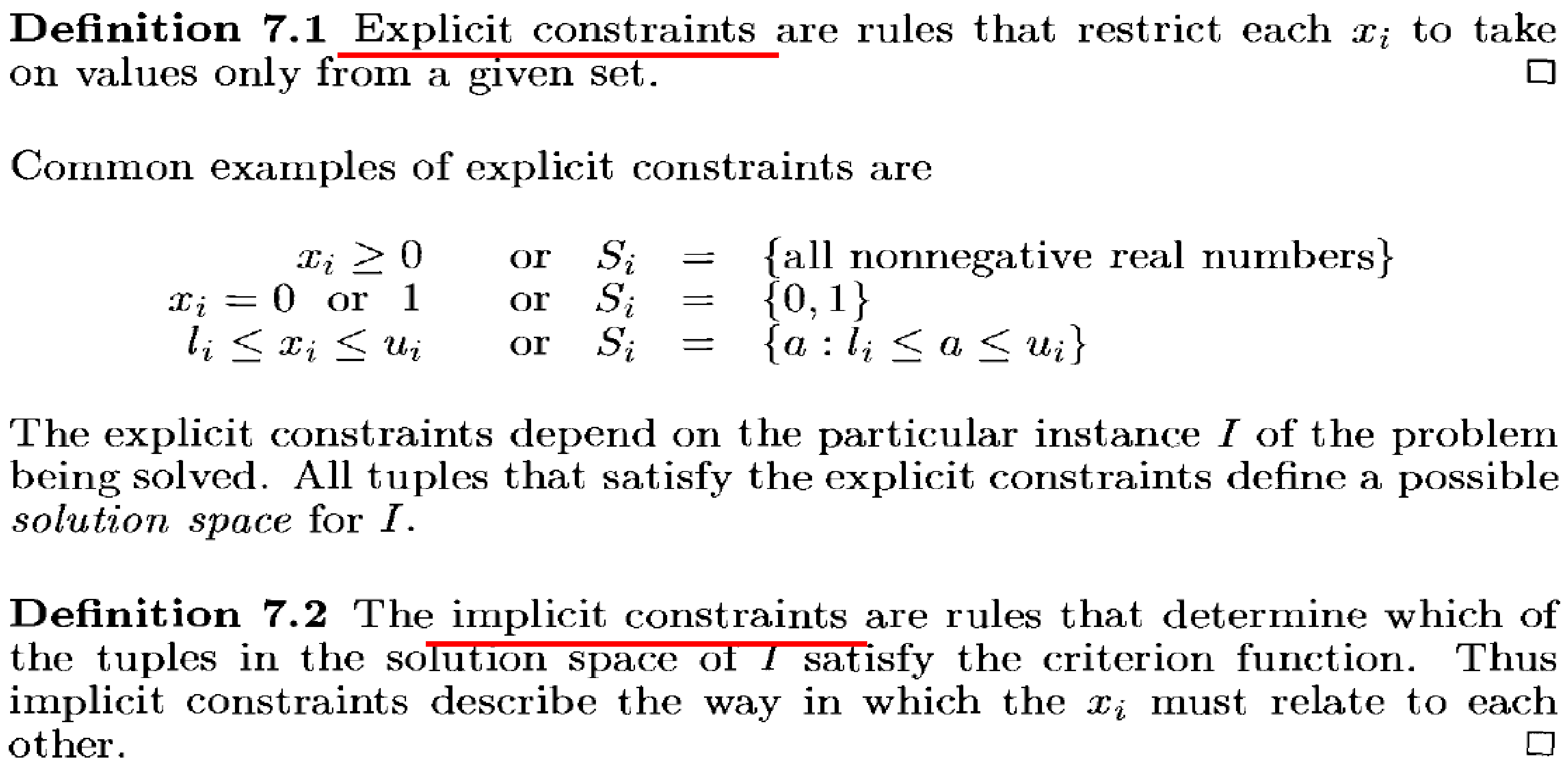
  $(x_1 \ldots\ldots\ldots\ldots\ldots x_i \ldots\ldots\ldots\ldots\ldots\ldots x_n)$

  where x_i is chosen from some finite set S_i.
- Often the problem to be solved calls for finding one vector that maximizes or minimizes or satisfies a criterion function $P(x_1 \ldots\ldots\ldots\ldots\ldots x_i \ldots\ldots\ldots\ldots x_n)$

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

Backtracking

- Many of the problems we solve using backtracking require that all the solutions satisfy a complex set of constraints.
- For any problem these constraints can be divided into two categories: **explicit and implicit.**



Definition 7.1 Explicit constraints are rules that restrict each x_i to take on values only from a given set. □

Common examples of explicit constraints are

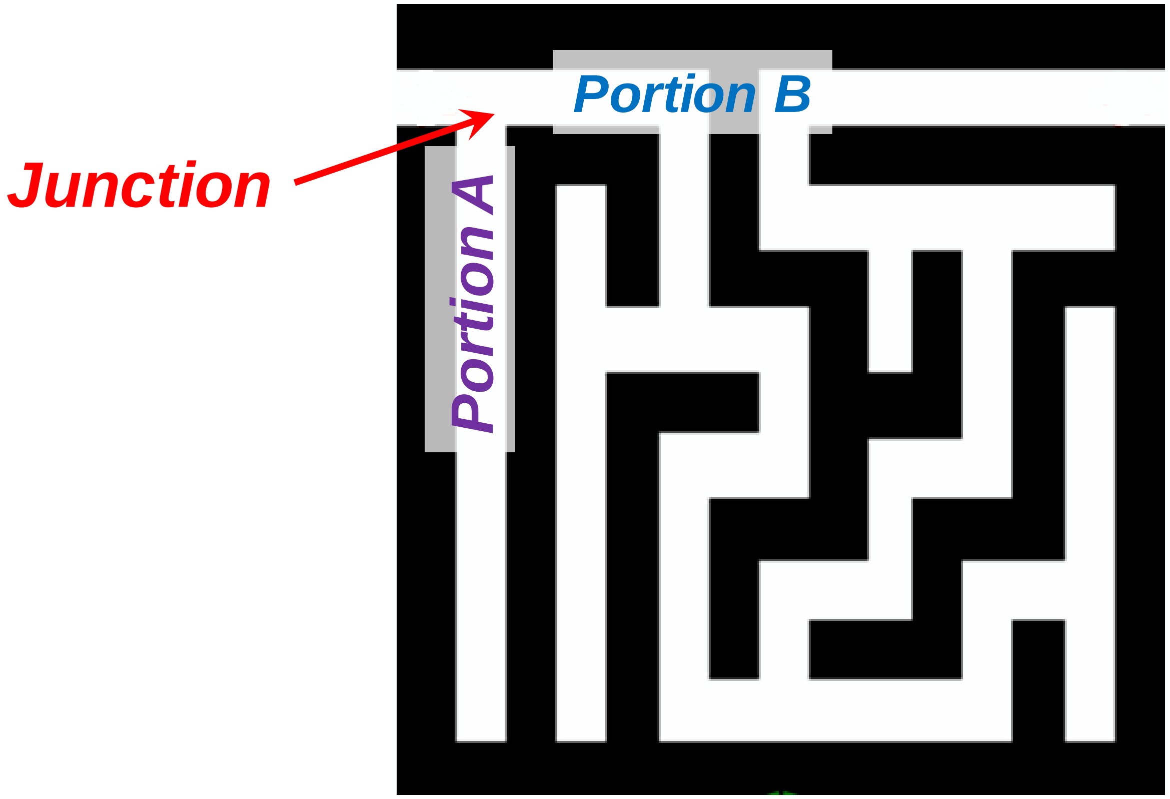
$$\begin{array}{rcrcl} x_i \geq 0 & \text{or} & S_i & = & \{\text{all nonnegative real numbers}\} \\ x_i = 0 \ \text{ or } \ 1 & \text{or} & S_i & = & \{0, 1\} \\ l_i \leq x_i \leq u_i & \text{or} & S_i & = & \{a : l_i \leq a \leq u_i\} \end{array}$$

The explicit constraints depend on the particular instance I of the problem being solved. All tuples that satisfy the explicit constraints define a possible *solution space* for I.

Definition 7.2 The implicit constraints are rules that determine which of the tuples in the solution space of I satisfy the criterion function. Thus implicit constraints describe the way in which the x_i must relate to each other. □

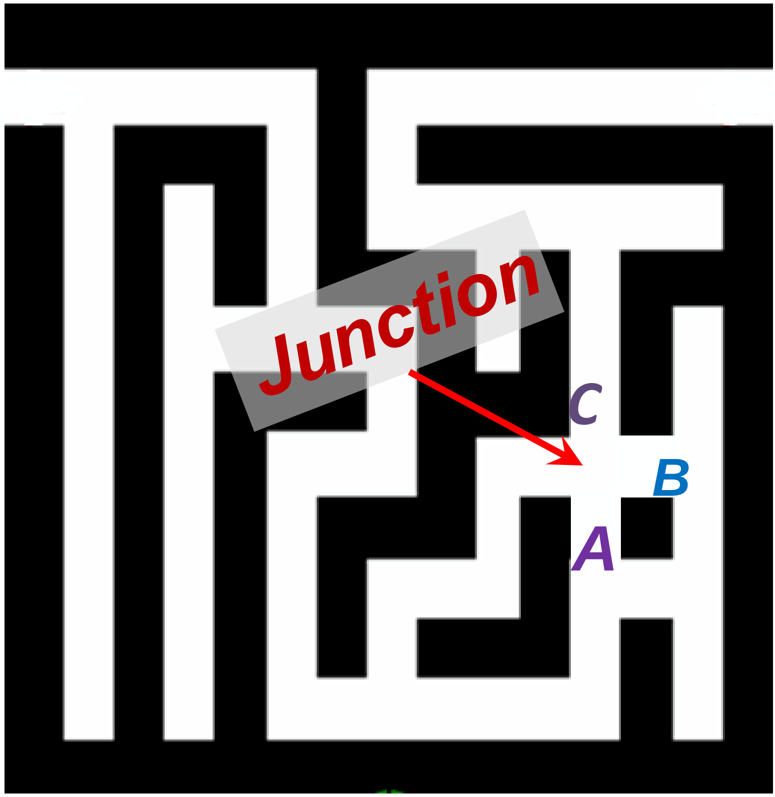
Backtracking: Idea

- Backtracking is a technique used to solve problems with a large search space, by systematically trying and eliminating possibilities.
- A standard example of backtracking would be going through a maze.
  - At some point, you might have two options of which direction to go:

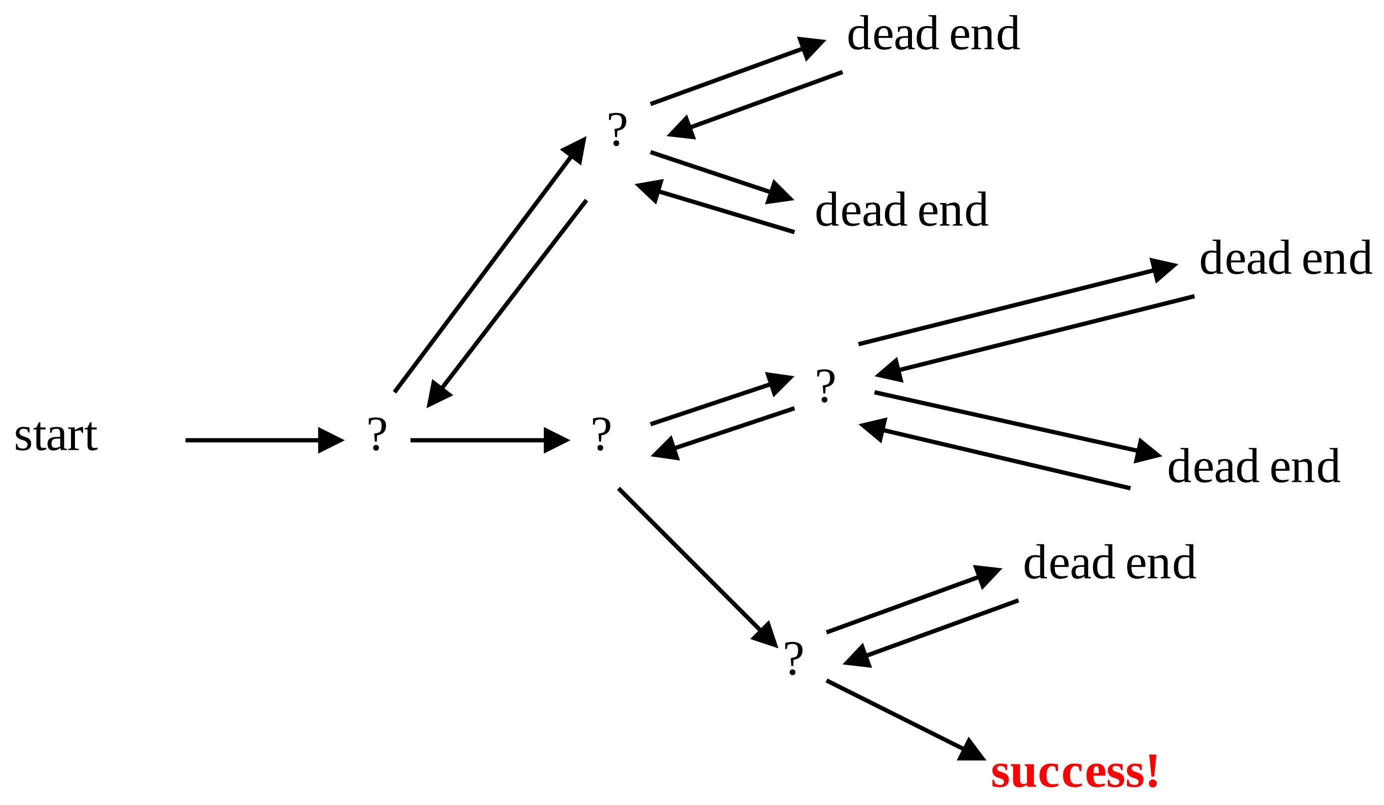


Backtracking

- Clearly, at a single junction you could have even more than 2 choices.
- The backtracking strategy says to try each choice, one after the other,
  - if you ever get stuck, *"backtrack"* to the junction and try the next choice.
- If you try all choices and never found a way out, then there is no solution to the problem.



Backtracking (animation)



Backtracking

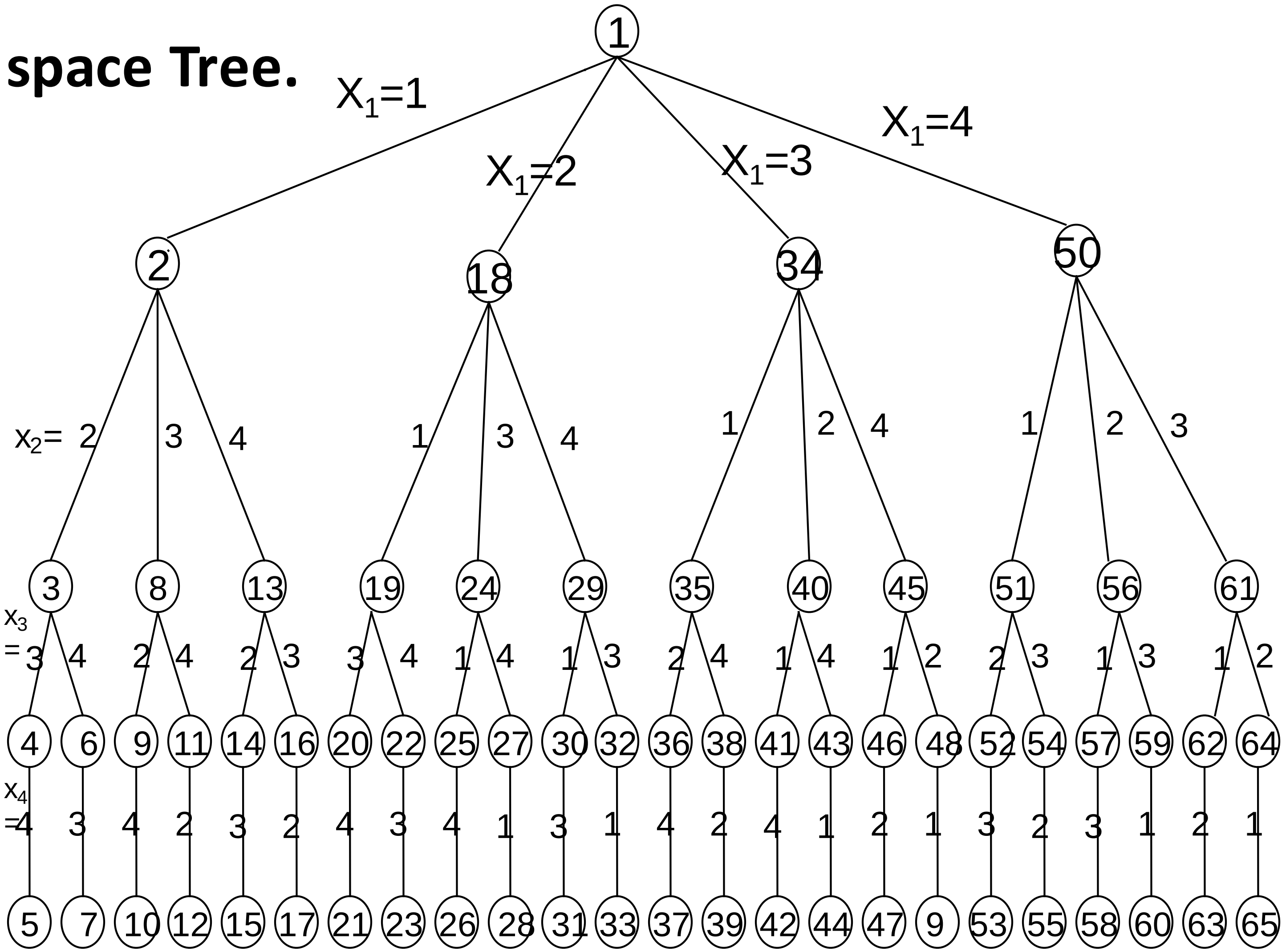
- Backtracking algorithms determine problem solutions by systematically searching the solution space for the give problem instance.
- This **search is facilitated by using a tree organization** for the solution space.
- For a given solution space many tree organizations may be possible.

Terminology regarding tree organizations of solution spaces

- Each node in this tree defines a **problem state**.
- All paths from the root to other nodes define the **state space** of the problem.
- **Solution states** are those problem states s for which the path from the root to a node defines a tuple in the solution space. only leaf nodes are solution states.
- **Answer states** are those solution states s for which the path from the root to s defines a tuple that is a member of the set of solutions(i.e., it satisfies the implicit constraints)of the problem.
- The tree organization of the solution space is referred to as the **state space tree**.

An Example State Space Tree

State space Tree.



Terminology regarding tree organizations of solution spaces

- State space trees can be of two ways:

- **Static trees** are ones for which tree organizations are independent of the problem instance being solved
  - Fixed tuple size formulation. Tree organization is independent of the problem instance being solved.

- **Dynamic trees** are ones for which organization is dependent on problem instance
  - After conceiving state space tree for any problem, the problem can be solved by systematically generating problem states, checking which of them are solution states, and checking which solution states are answer states

Terminology regarding tree organizations of solution spaces

- There are **two different ways to generate** the problem states.
- Both of these begin with the root node and generate other nodes.
- A node which has been generated and all of whose children have not yet been generated is called a **live node**.
- The live node whose children are currently being generated is called the **E-node** (node being expanded).
- A **dead node** is a generated node which is not to be expanded further or all of whose children have been generated.

Terminology regarding tree organizations of solution spaces

- **In the first** of these two methods as soon as a new child C of the current E-node R is generated, this child will become the new E-node.
- Then R will become the E-node again when the subtree C has been fully explored.
- This corresponds to a **depth first generation of the problem states.**

- **In the second** state generation method, the E-node remains the E-node until it is dead.
- It corresponds to **breadth first generation of the problem states.**
- In both methods, bounding functions are used to kill live nodes without generating all their children

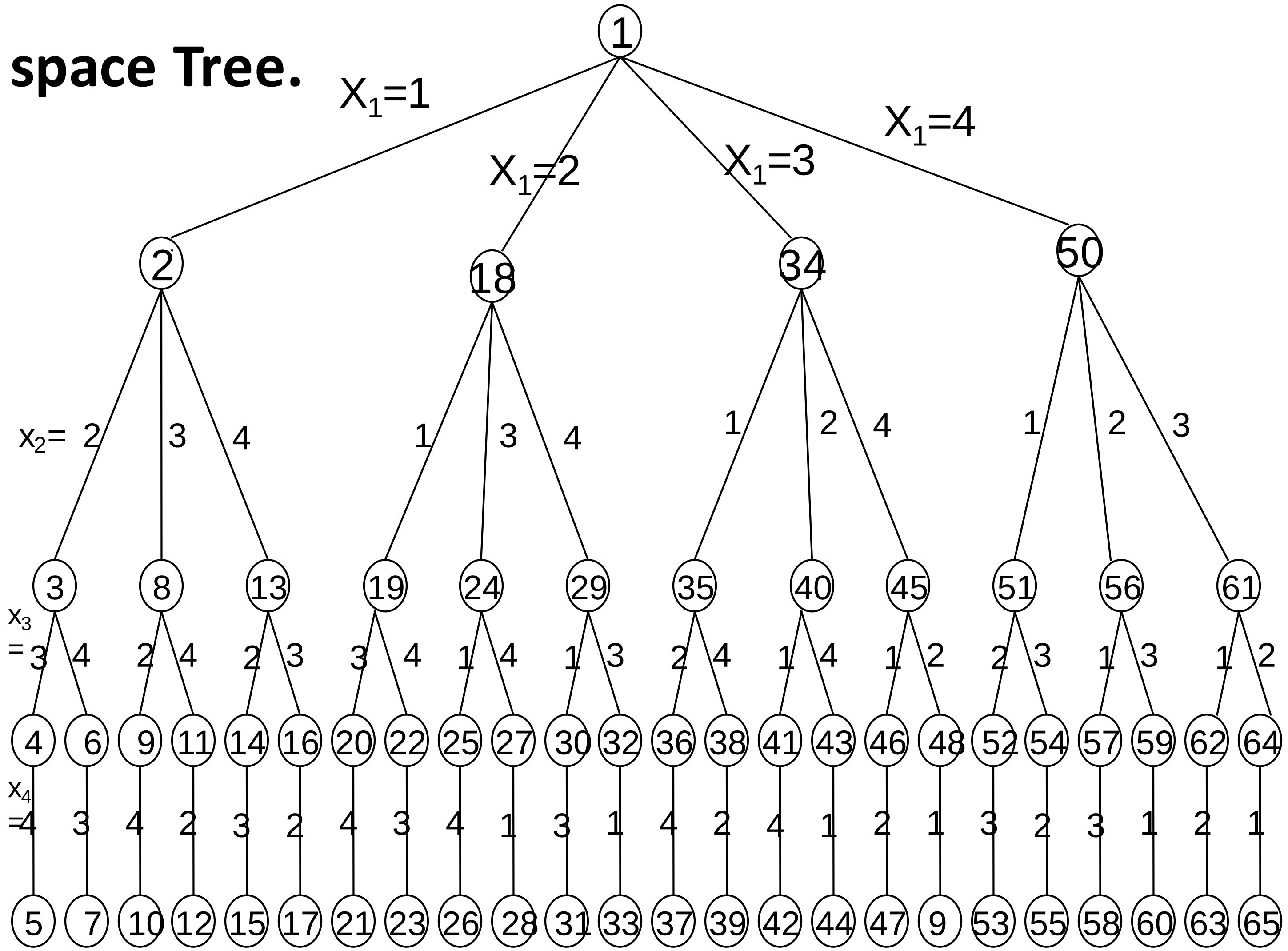
Backtracking Vs Branch-and-Bound

- **Depth first generation of nodes with bounding functions is called backtracking.**

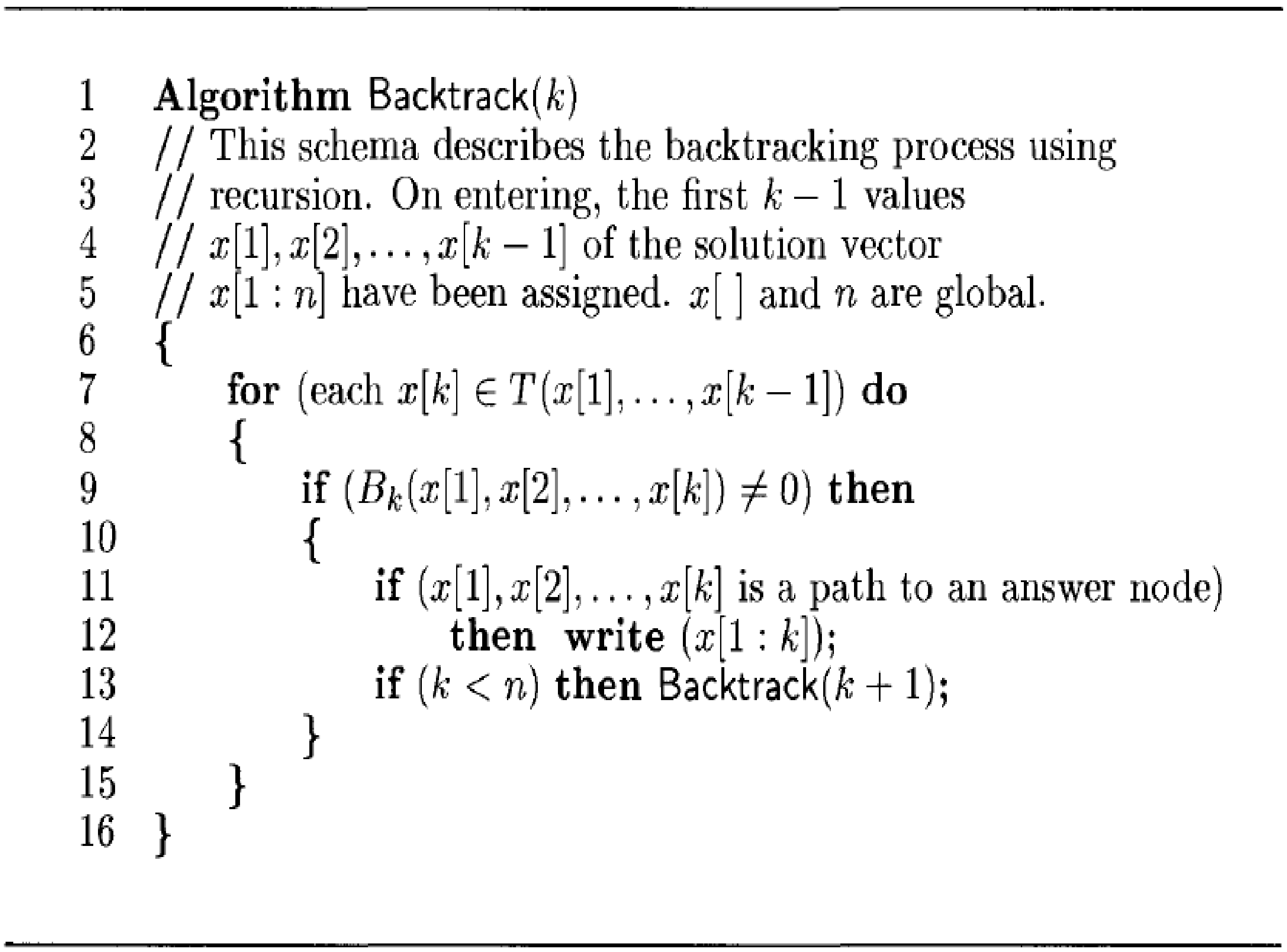
- **Breadth first generation of nodes with bounding functions lead to branch-and-bound methods**

An Example State Space Tree

State space Tree.

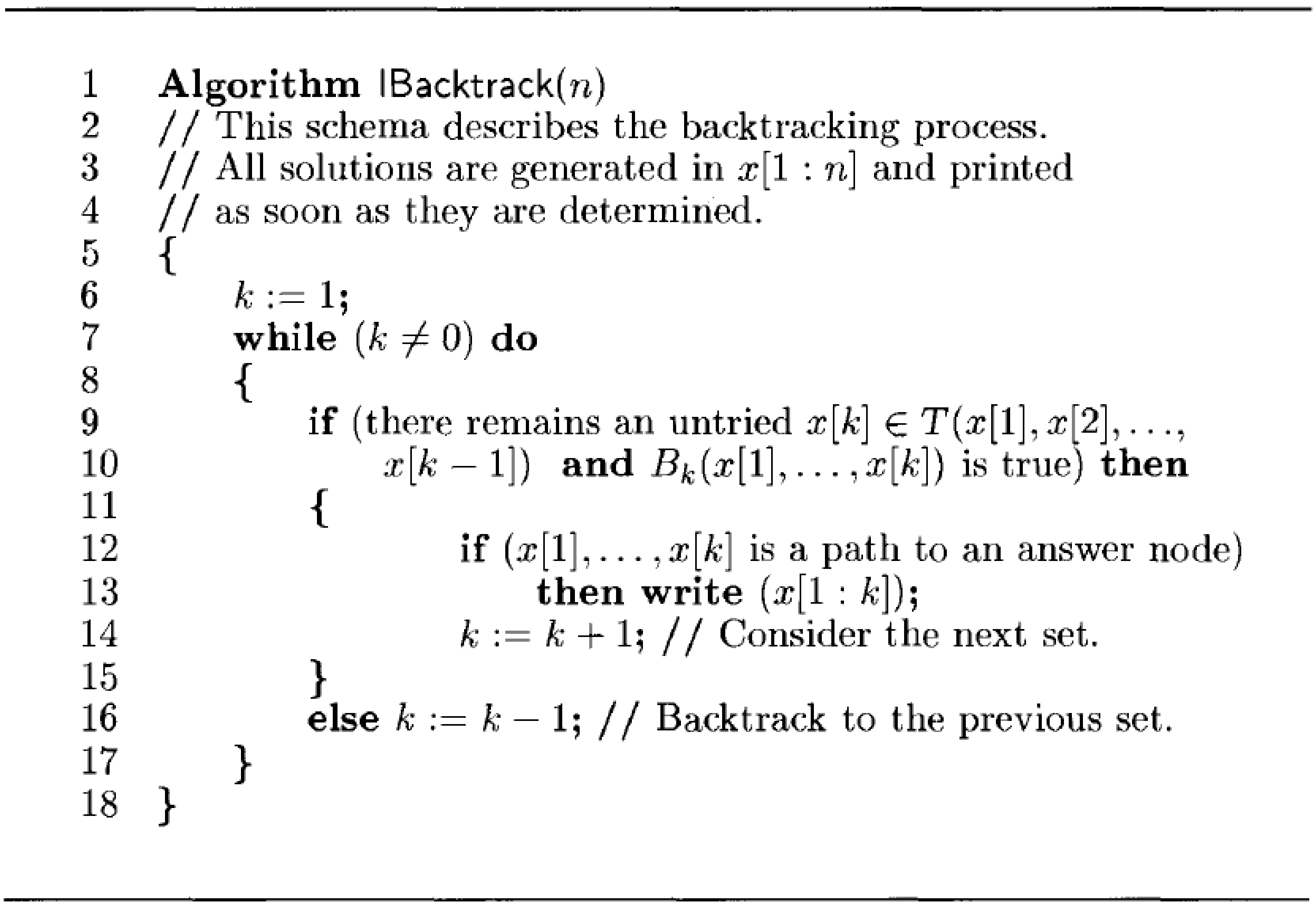


Recursive Algorithm for Backtracking-General Method



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

Iterative Algorithm for Backtracking-General Method



```
Algorithm IBacktrack($n$)
// 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 ($k \neq 0$) do
    {
        if (there remains an untried $x[k] \in T(x[1], x[2], \ldots,$
            $x[k-1])$ and $B_k(x[1], \ldots, x[k])$ is true) then
        {
            if ($x[1], \ldots, x[k]$ is a path to an answer node)
                then write ($x[1:k]$);
            $k := k + 1$; // Consider the next set.
        }
        else $k := k - 1$; // Backtrack to the previous set.
    }
}
```

Types of problems can be solved with Backtracking

- **Enumeration problems:**
  - Problem will have number of solutions and all solutions are to be identified.

- **Decision Problems:**
  - Solution to the problem is in terms of Yes/No or True/False

- **Optimization Problems:**
  - Problem will have number of solutions and best of all to be identified.

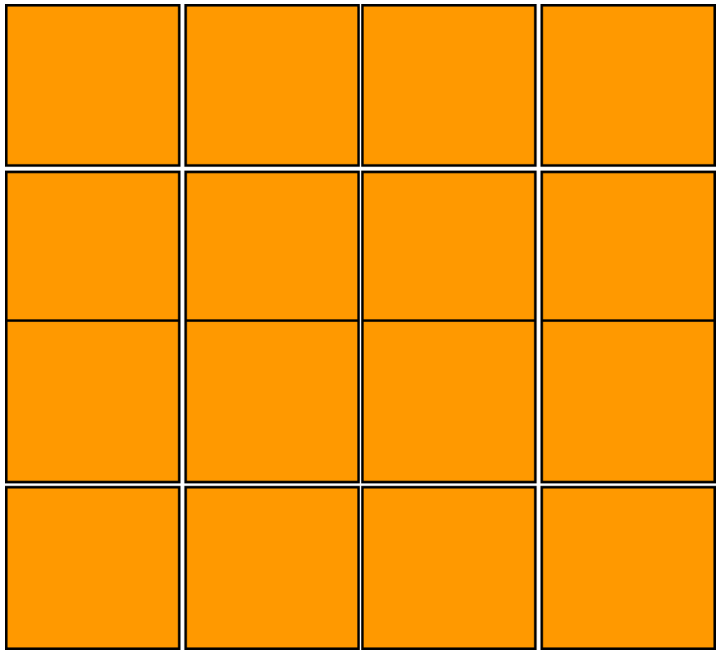
Topics

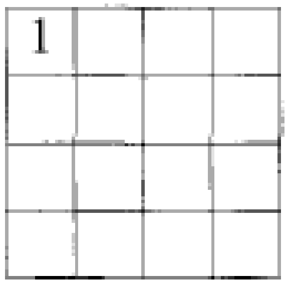
- **General Method**
- **N-Queens Problem**
- **Sum of Subsets**
- **Graph Coloring**
- **Hamiltonian Cycles.**

N-Queens Problem

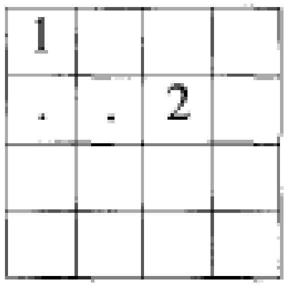
- The problem is to place n queens on an n x n chessboard so that no two "attack" that is no two queens on the same row, same column, or same diagonal.
- Assume rows and columns of chessboard are numbered 1 through n. Queens also be numbered 1 through n.
- To ensure the condition No two queens on same Row, **ith queen can be placed in ith row only.**
- **Explicit Constraints:** in ith row queen can be placed in any one of the n columns. Xi ∈ $\{1,2,,\ldots,n\}$
- **Implicit Constraints (Condition to ensure):**
  - **No two queens on same Column**:
  - **No two queens on same Diagonal**

4-Queens Problem

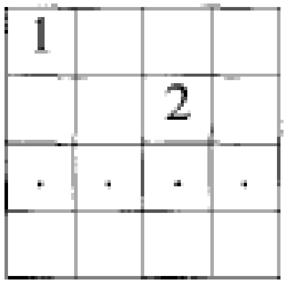




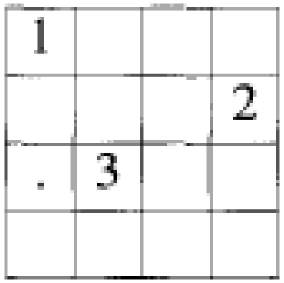
(a)



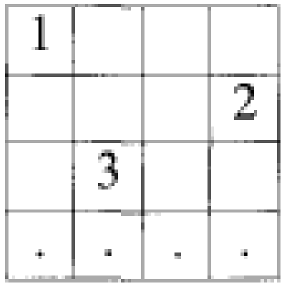
(b)



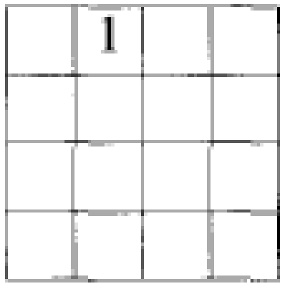
(c)



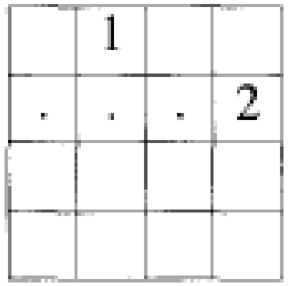
(d)



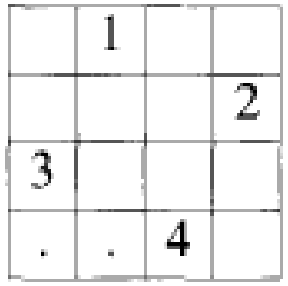
(e)



(f)

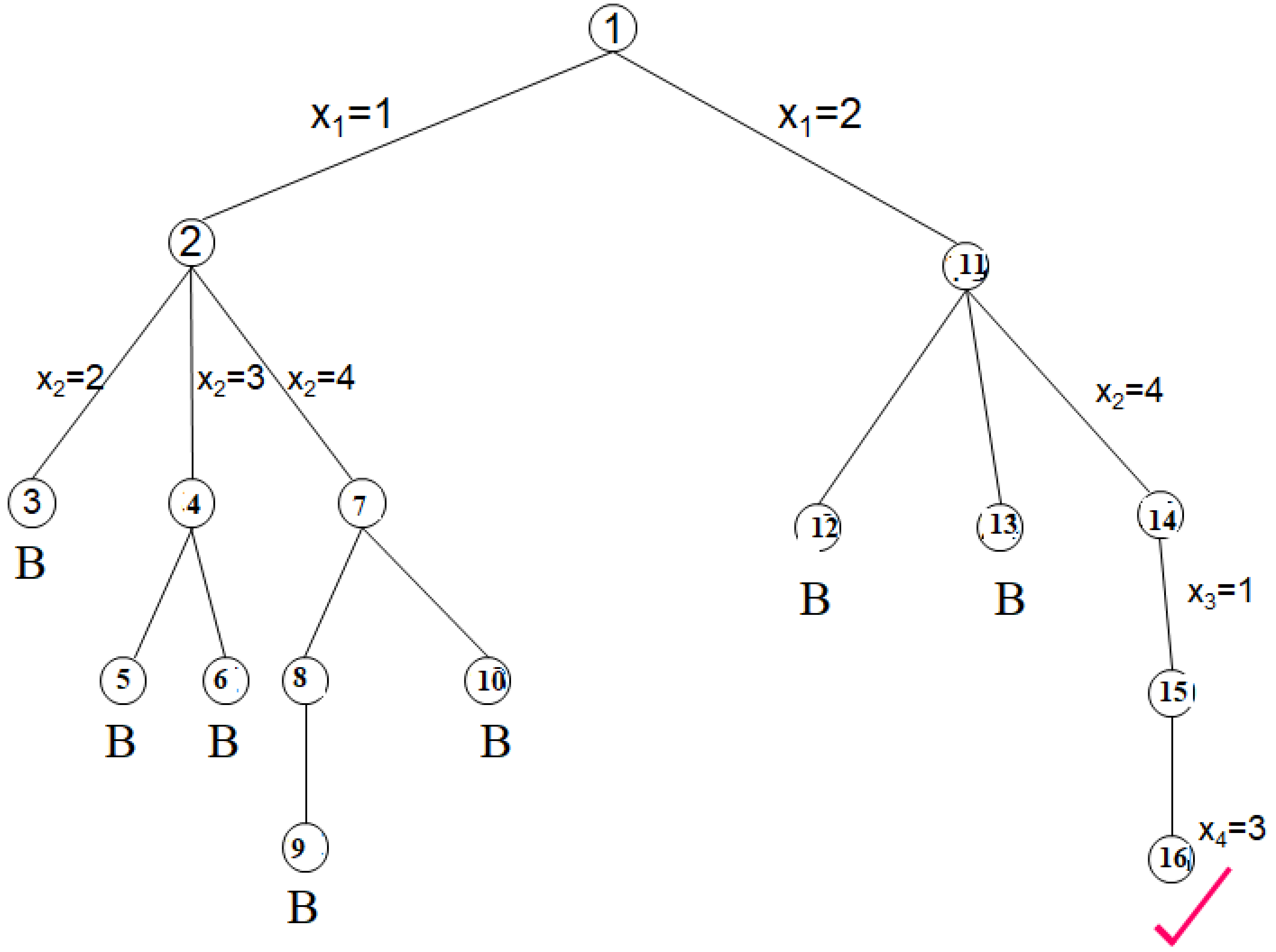


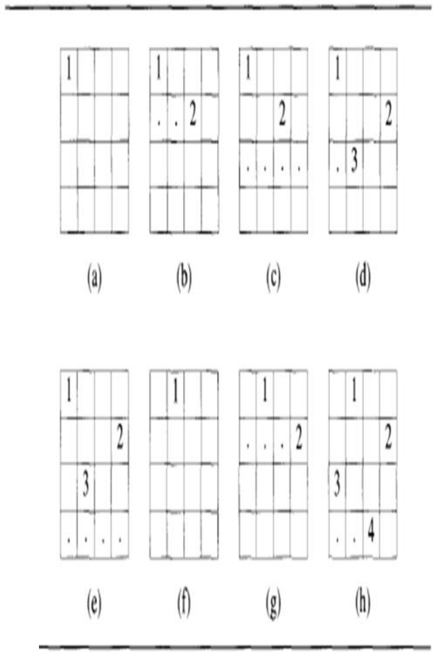
(g)



(h)

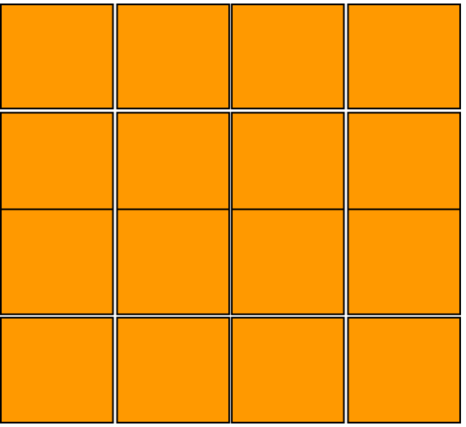
State Space Tree for 4-queens problem





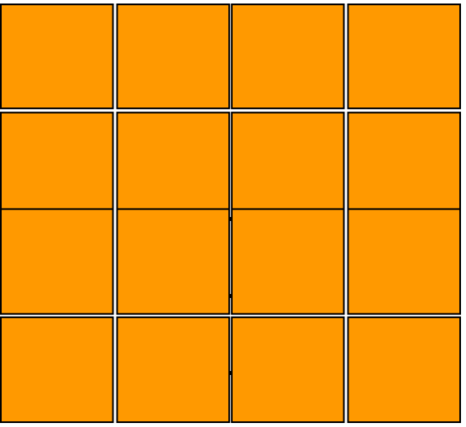
N-Queens Problem

- **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 ($x_1,x_2,\ldots\ldots x_n$), 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**



N-Queens Problem

- **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**



(i,j)	(k,l) are diagonal
(2,3)	(1,2), (3,4)
	(1,4), (3,2), (4,1)
(1,2)	(2,3), (3,4)
	(2,1)

N-Queens Problem

- 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.

   $i - j = k - l$ -----(1)

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

   $i + j = k + l$ -----(2)

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

  Second equation implies : $j - l = k - l \rightarrow -(i - k)$

- Therefore, two queens lie on the same diagonal if and only if

$$\boxed{|j - l| = |i - k|}$$

Algorithm for N-Queens Problem

```
Algorithm NQueens($k, n$)
// Using backtracking, this procedure prints all
// possible placements of $n$ queens on an $n \times n$
// chessboard so that they are nonattacking.
{
    for $i := 1$ to $n$ do
    {
        if Place($k, i$) then
        {
            $x[k] := i$;
            if ($k = n$) then write ($x[1 : n]$);
            else NQueens($k + 1, n$);
        }
    }
}
```

Algorithm for N-Queens Problem

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

Problem for Practice

Draw the State Space tree for 8-Queens problem at least for one solution.

Topics

- **General Method**
- **N-Queens Problem**
- **Sum of Subsets**
- **Graph Coloring**
- **Hamiltonian Cycles.**

Sum of Subsets or Subset Sum Problem

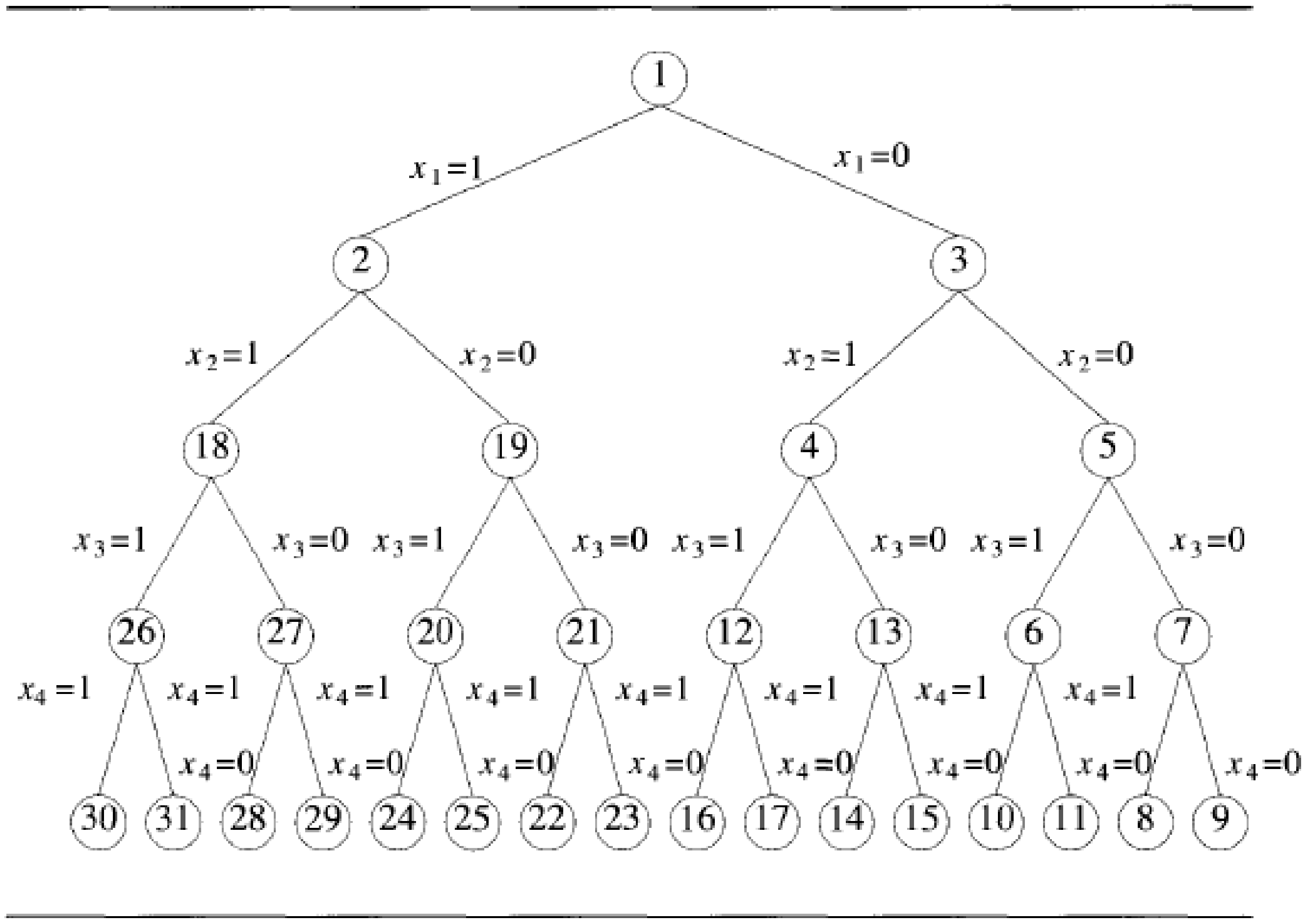
- The Problem is to find a subsets " s' " of the given set S = (v_1 v_2 v_3...v_n) where the elements of the set S are n positive integers in such a manner that s'∈ S and sum of the elements of subsets " s' " is equal to the given positive integer 'M.'

Sum of Subsets Problem

- **Explicit Constraints:**
  - Possibilities are either the number considered or not considered into the subset.
  - Hence $X_i \in \{0,1\}$

- **Implicit Constraints:**
  - Sum of the numbers in the subset must be equal to m

Static Space tree for n=4 in Sum of Subsets Problem



Sum of Subsets Problem

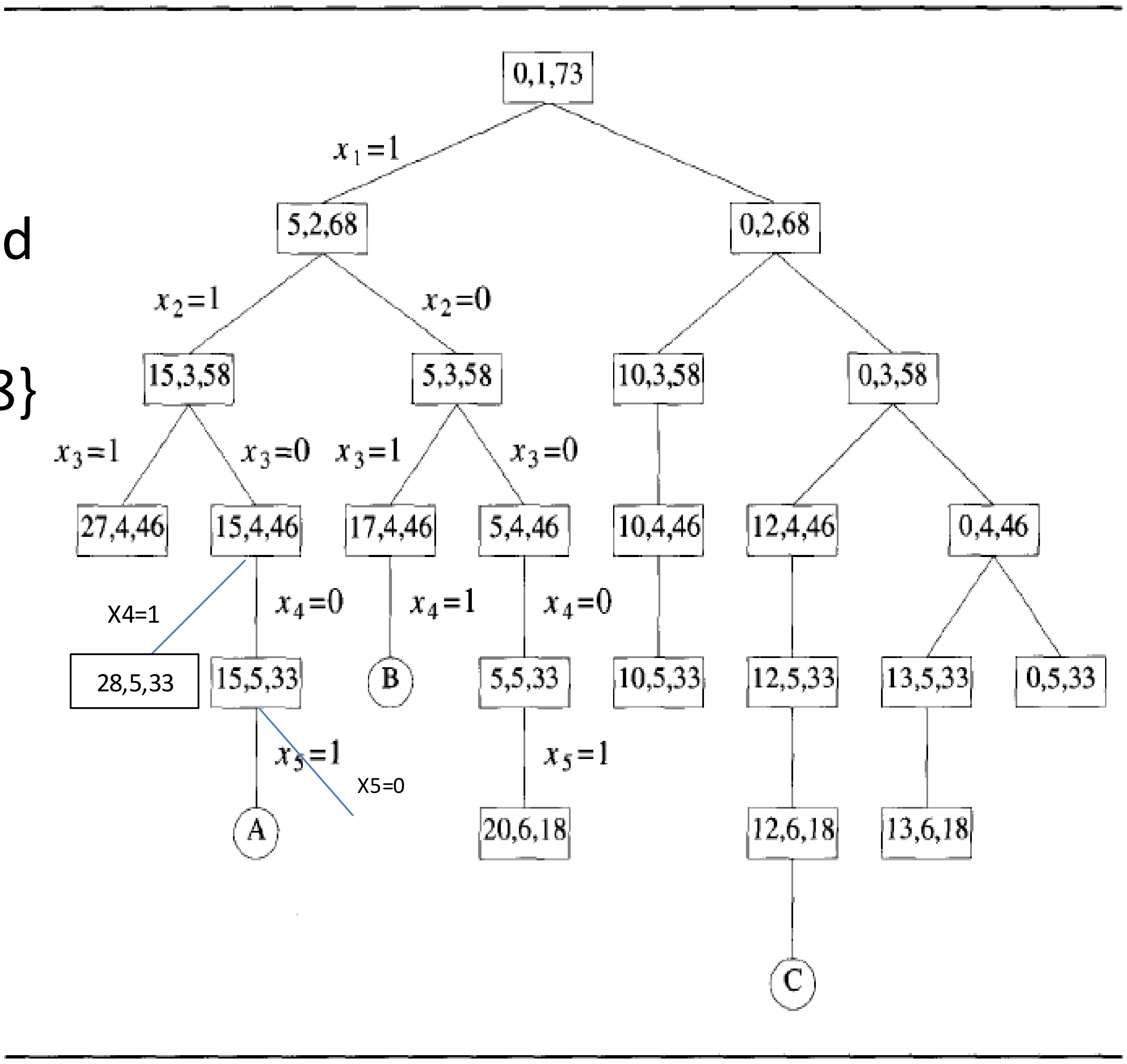
- **Ex: n = 6, M = 30, and w[1 : 6] = {5,10,12,13,15,18}**

State Space tree for the Example

Ex:

n = 6, M = 30, and

w[1 : 6] = {5,10,12,13,15,18}



Algorithm for sum of subsets Problem

```
Algorithm SumofSub (s, k, r)
        //   $s = \sum_{j=1}^{k-1} w[j] * x[j]$
        //   k is the position of the number for which decision is to be taken
        //   r = $\sum_{j=k}^{n} w[j]$
{
                    // Generate left child.
    x[k]:=1;
    if (s+w[k]==m) then write (x[1: k]);
    else if (s+w[k]+w[k+1] <= m) then
            SumofSub (s+w[k], k+1, r-w[k]);
                    // Generate right child
    if ((s+r-w[k] >= m) and (s+w[k+1] <= m)) then
    {
        x[k]:=0 ;
        SumofSub (s, k+1, r-w[k]);
    }
}
```

Problem for Practice

- Let w = {5,7,10,12,15,18,20} and M= 35.Find all possible subsets of w that sum to M. Draw the state space tree that is generated.

Topics

- General Method
- N-Queens Problem
- Sum of Subsets
- Graph Coloring
- Hamiltonian Cycles.

Graph Coloring Problem

- **Let ‘G’ be a graph and ‘m’ be a given positive integer (no of colors).**

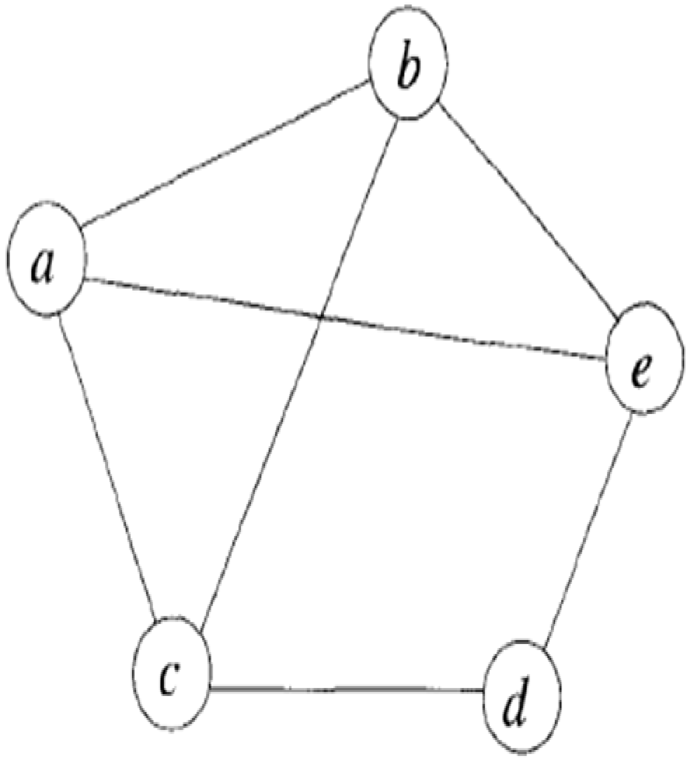
- Discovering whether the nodes of G can be colored in such a way that **no two adjacent nodes have the same color** yet only m colors are used is termed as **m-colorability decision problem**.

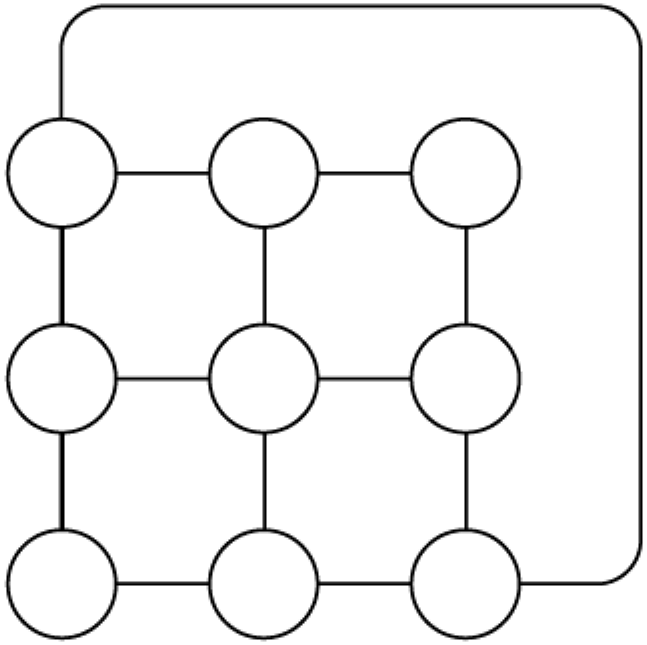
- Finding the number of ways a graph can be colored with given m colors is the **m-colorability enumeration problem**

- 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**.

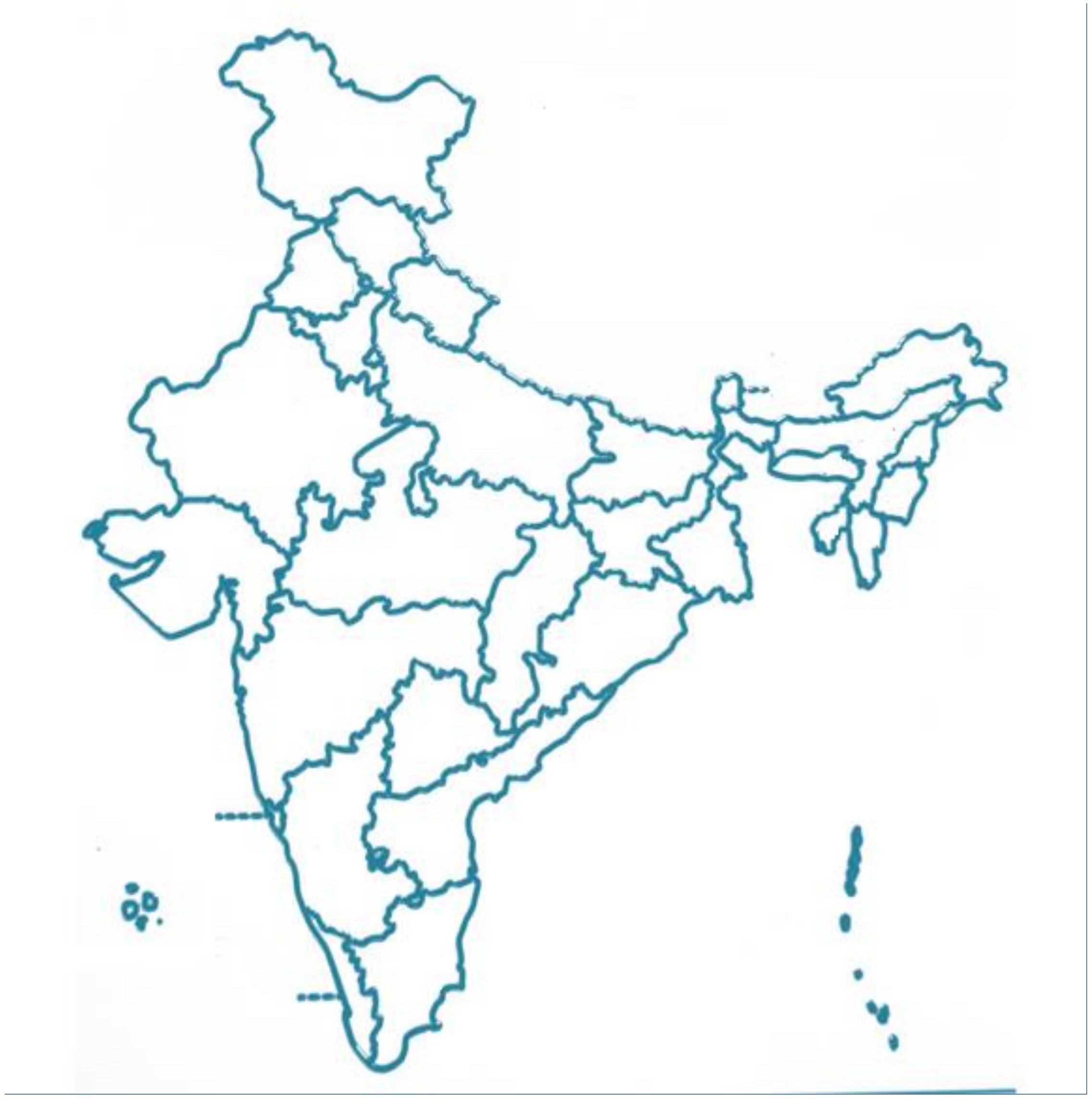
Graph Coloring Problem





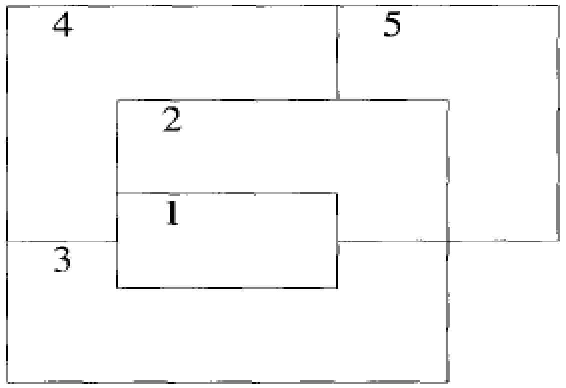
Find the chromatic number of the above graphs

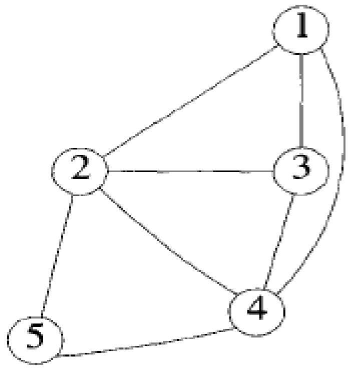
Application of Graph coloring



Application of Graph coloring

- **Question:** given any map, can the regions be colored in such a way that no two adjacent regions have the same color yet only m 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.



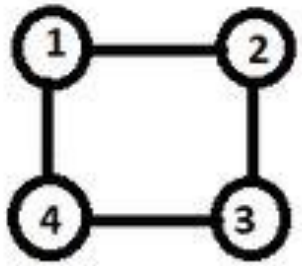


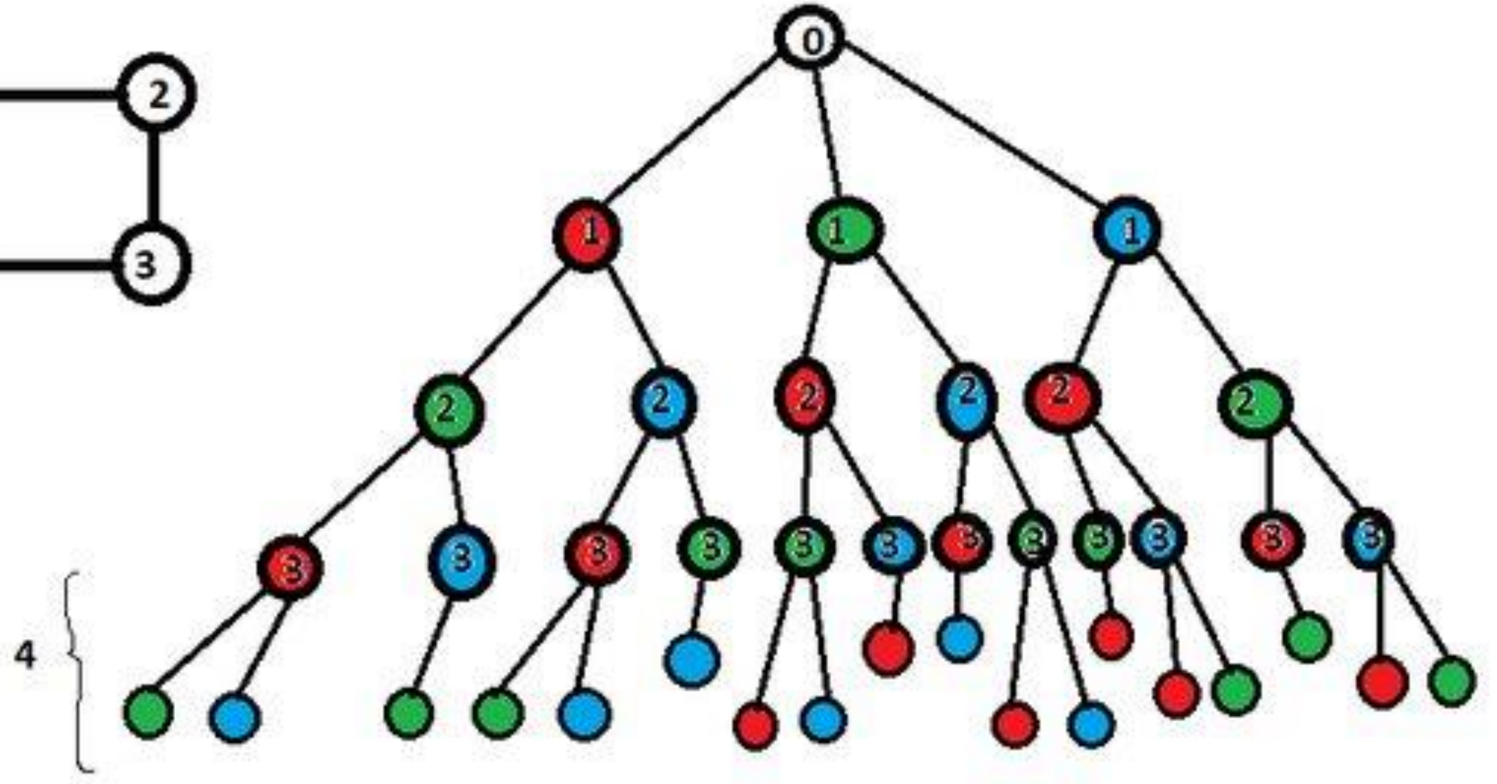
Graph Coloring Problem

- **Explicit Constraints:**
  - Solution X_i is the color number used for ith vertex.
  - Hence $X_i \in \{1,2, \dots , m\}$

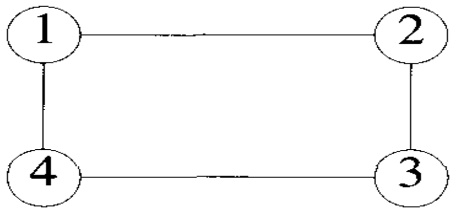
- **Implicit Constraints:**
  - No two adjacent vertices have same color

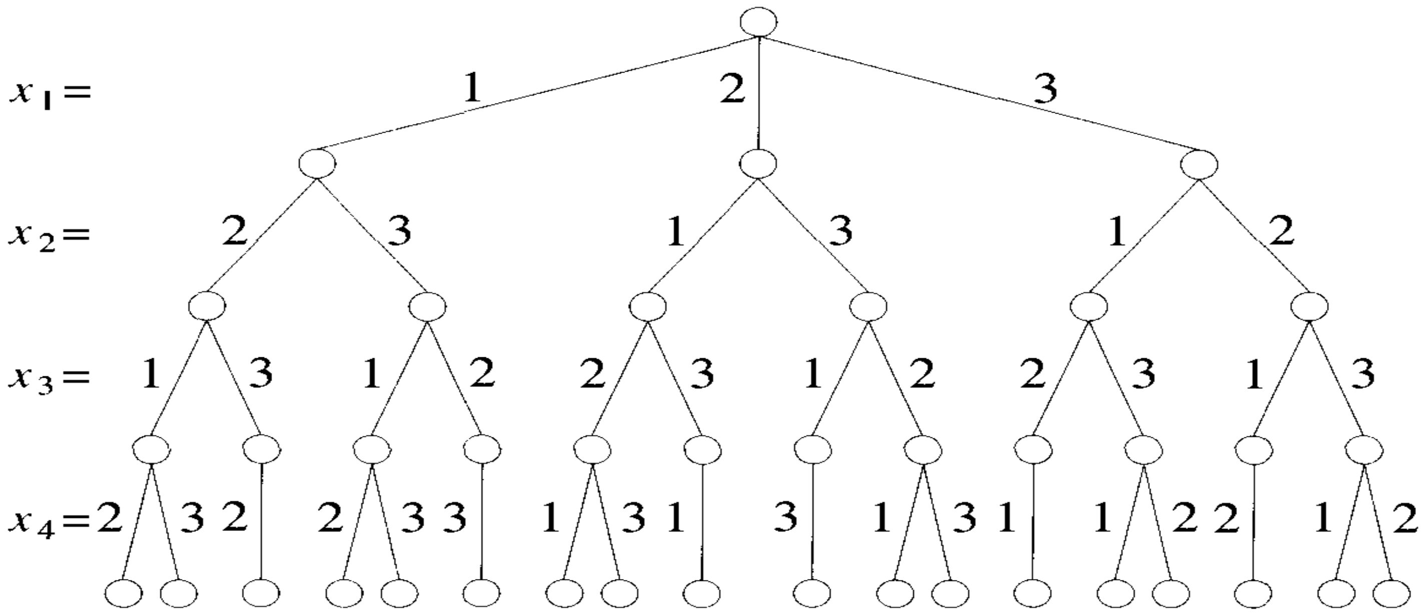
Example for Graph Coloring





Example for Graph Coloring





Steps to color the Graph

- Consider the adjacency matrix 'C' of the graph(1:n,1:n). 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), where X(index) is the color and index is the node.
- Here formula is used to set the color is,

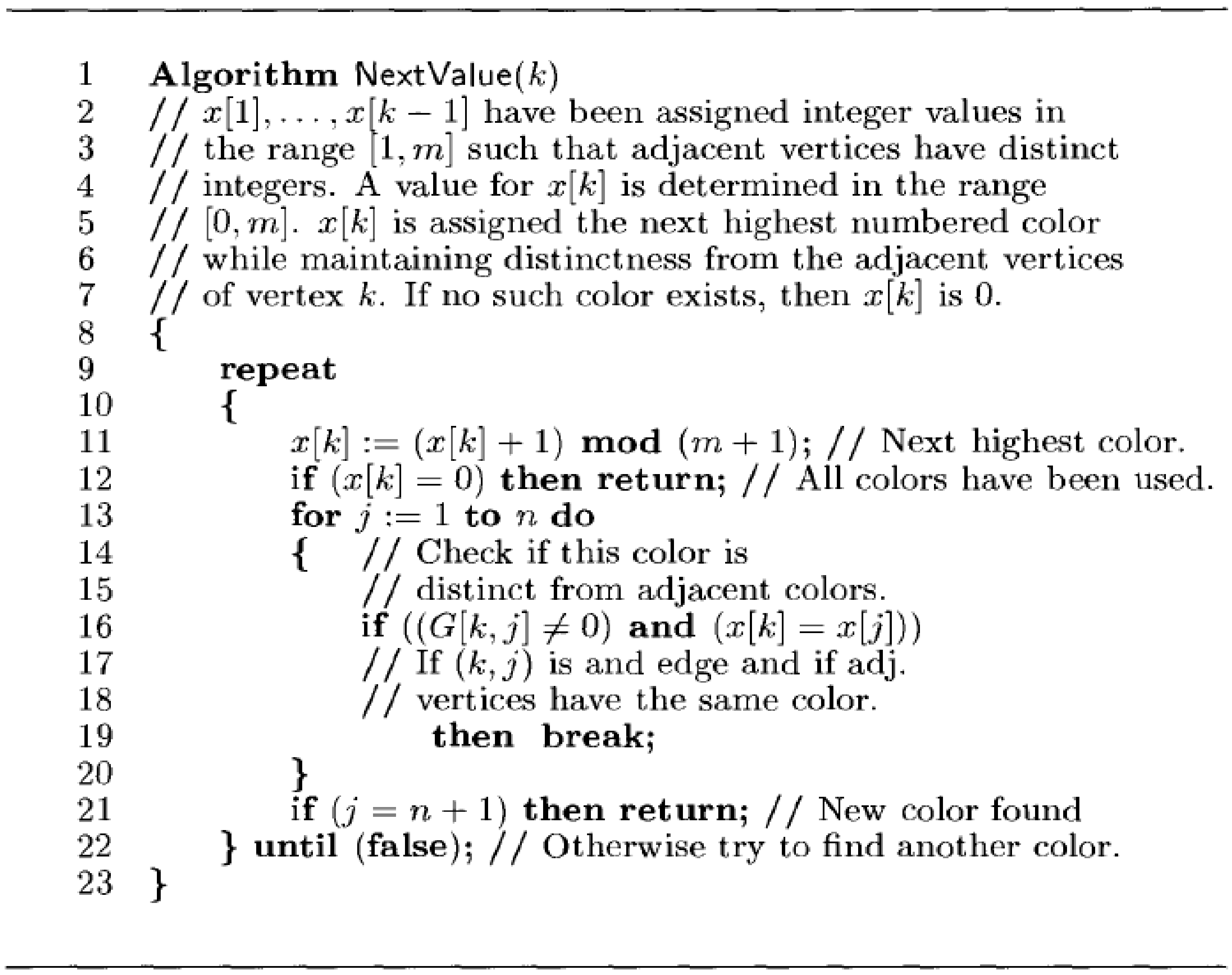
  X(k) = (X(k)+1) % (m+1)
- we have to check whether the adjacent nodes has got the same values, if so then we have to assign the next color value.
- The function which is used to check the adjacent nodes of k and same color is,

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

Algorithm for Graph Coloring

```
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);
}
```

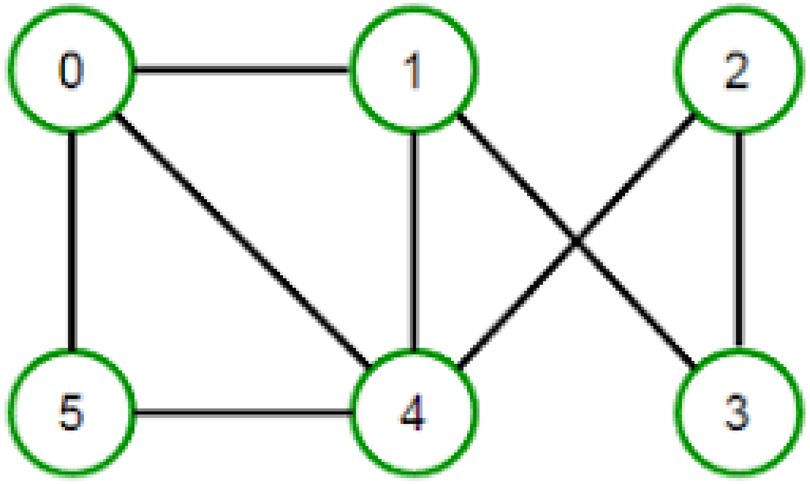
Algorithm for Graph Coloring

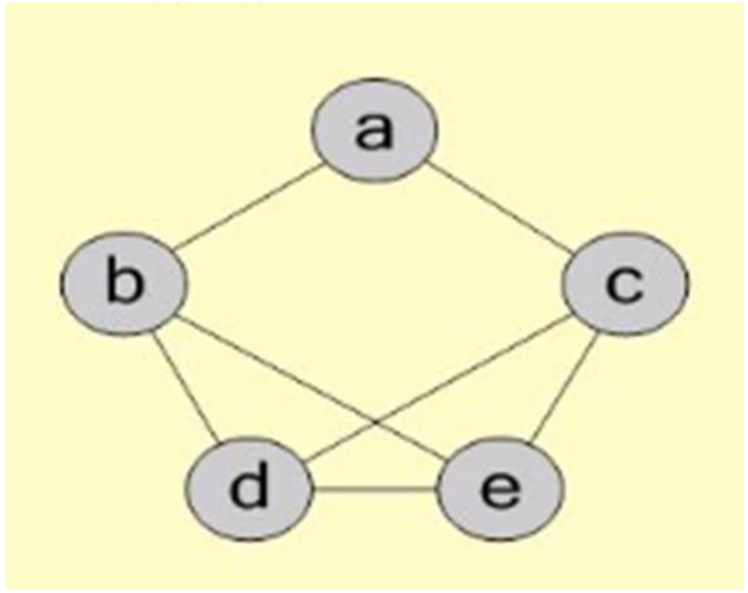


```
Algorithm NextValue($k$)
// $x[1], \ldots, x[k-1]$ have been assigned integer values in
// the range $[1, m]$ such that adjacent vertices have distinct
// integers. A value for $x[k]$ is determined in the range
// $[0, m]$. $x[k]$ is assigned the next highest numbered color
// while maintaining distinctness 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 $((G[k, j] \neq 0)$ 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.
}
```

Problem for Practice

- Find the number of ways to color the following graphs with 3 colors {C1, C2, C3}. Draw the state space tree.





Topics

- **General Method**
- **N-Queens Problem**
- **Sum of Subsets**
- **Graph Coloring**
- **Hamiltonian Cycles**

Hamiltonian Cycle Problem

- Let G = (V, E) be a connected graph with n vertices.
- A Hamiltonian cycle (suggested by Sir William Hamilton)is a round-trip path along n edges of G that visits every vertex once and returns to its starting position.
- In other words if a Hamiltonian cycle begins at some vertex v∈ G and the vertices of G are visited in the some sequence V1,……, Vn+1.
- Then for any pair (Vj, Vj+i) the edge ∈ E, and the all visited vertices should be distinct except for v1 and Vn+1 which are equal.

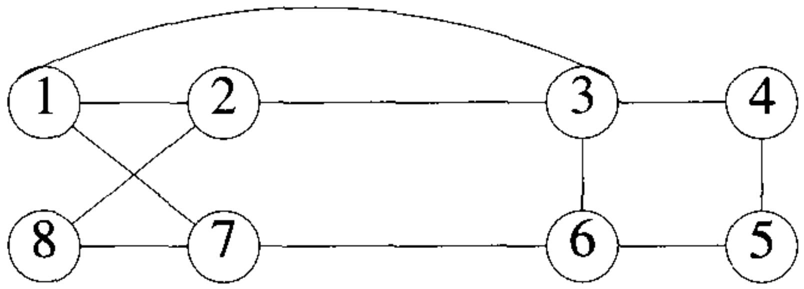
Hamiltonian Cycles Problem

- **Explicit Constraints:**
  - Possibilities are which vertex should be visited in ith position.
  - Hence Xi $\in \{1,2,\ldots,n\}$

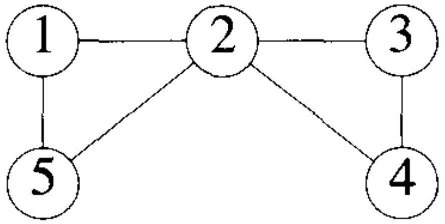
- **Implicit Constraints:**
  - Vertex has to be visited only once
  - Vertex at ith position should have an edge with vertex at i-1 position

Example for Hamiltonian Cycles Problem

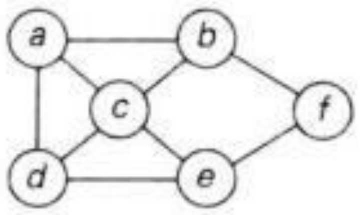
G1:



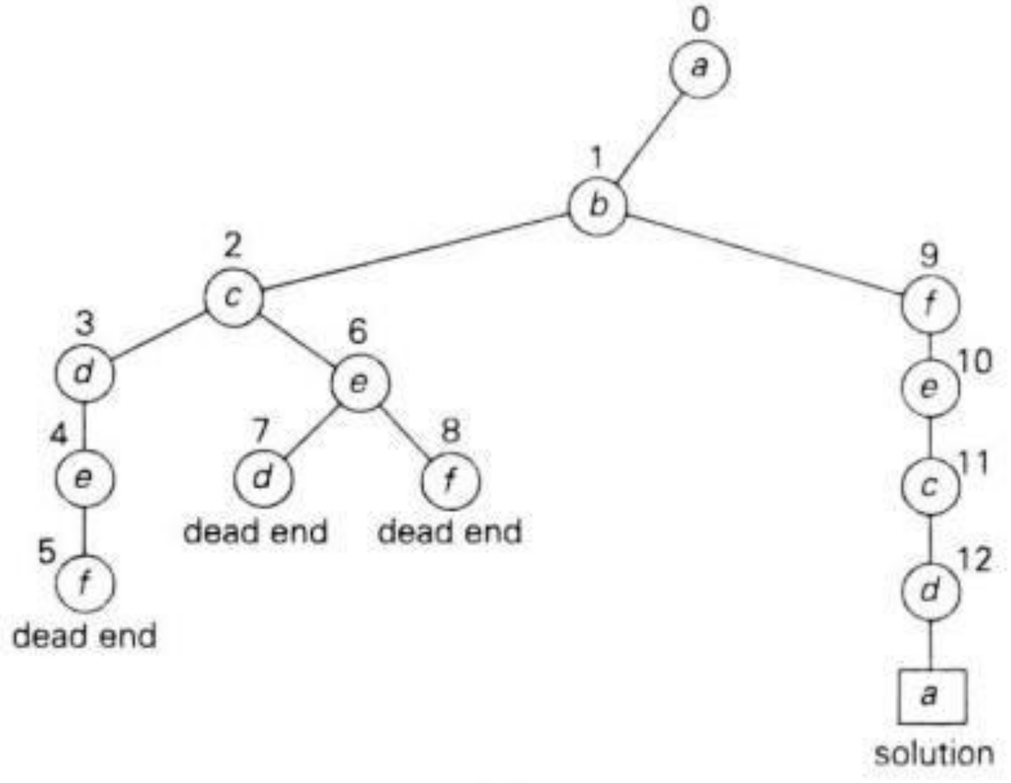
G2:



Example for Hamiltonian Cycles Problem



(a)



(b)

Algorithm for Hamiltonian Cycles Problem

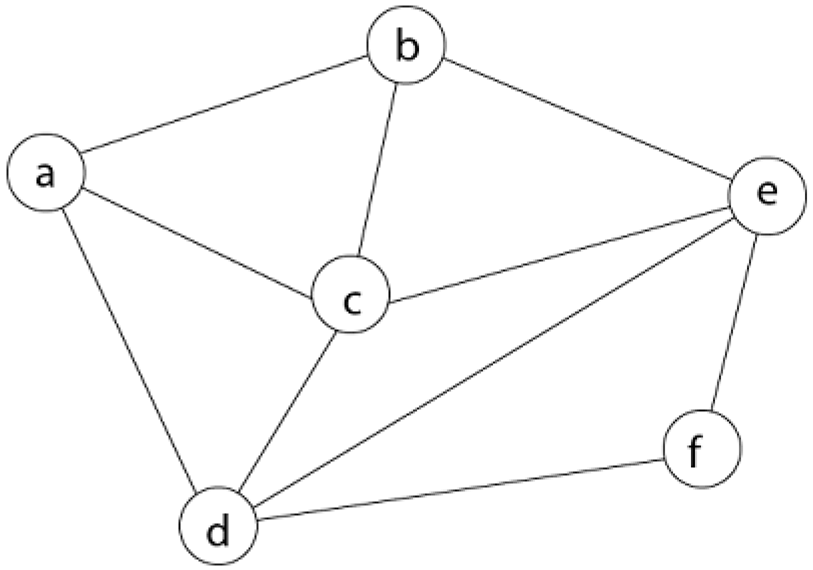
```
Algorithm Hamiltonian($k$)
// This algorithm uses the recursive formulation of
// backtracking to find all the Hamiltonian cycles
// of a graph. The graph is stored as an adjacency
// matrix $G[1:n, 1:n]$. All cycles begin at node 1.
{
    repeat
    { // Generate values for $x[k]$.
        NextValue($k$); // Assign a legal next value to $x[k]$.
        if ($x[k] = 0$) then return;
        if ($k = n$) then write ($x[1:n]$);
        else Hamiltonian($k+1$);
    } until (false);
}
```

Algorithm for Hamiltonian Cycles Problem

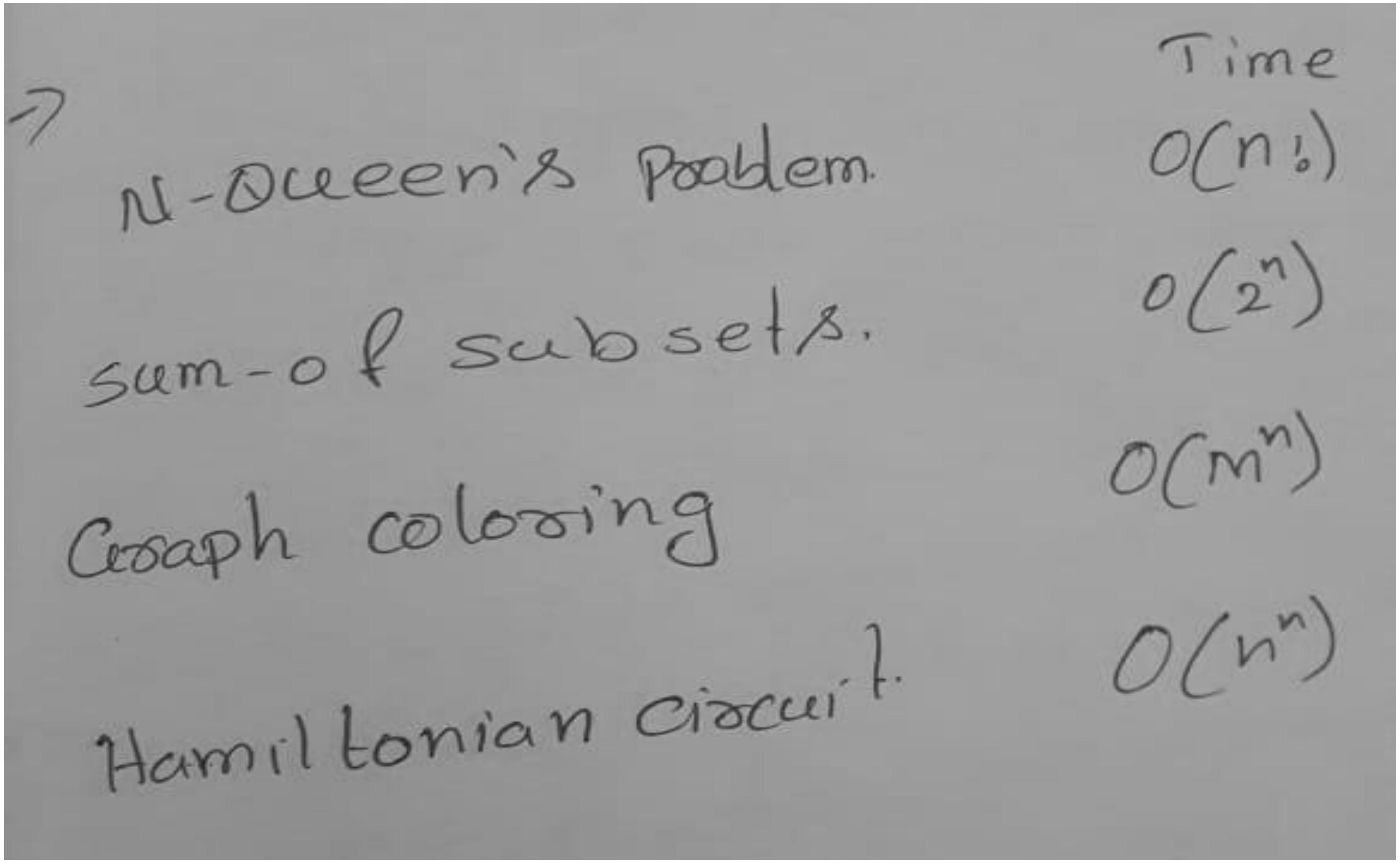
```
Algorithm NextValue($k$)
// $x[1:k-1]$ is a path of $k-1$ distinct vertices. If $x[k]=0$, then
// no vertex has as yet been assigned to $x[k]$. After execution,
// $x[k]$ is assigned to the next highest numbered vertex which
// does not already appear in $x[1:k-1]$ and is connected by
// an edge to $x[k-1]$. Otherwise $x[k]=0$. If $k=n$, then
// in addition $x[k]$ is connected to $x[1]$.
{
    repeat
    {
        $x[k] := (x[k]+1)$ mod $(n+1)$; // Next vertex.
        if $(x[k]=0)$ then return;
        if $(G[x[k-1], x[k]] \neq 0)$ then
        { // Is there an edge?
            for $j := 1$ to $k-1$ do if $(x[j]=x[k])$ then break;
                                   // Check for distinctness.
            if $(j=k)$ then // If true, then the vertex is distinct.
                if $((k<n)$ or $((k=n)$ and $G[x[n], x[1]] \neq 0))$
                    then  return;
        }
    } until (false);
}
```

Problem for Practice

- Find all possible Hamiltonian cycles for the following graph. Draw the state space tree at least for 3 solutions.



Time Complexity of Backtracking Problems



	Time
N-Queen's Problem.	$O(n!)$
sum-of subsets.	$O(2^n)$
Graph coloring	$O(m^n)$
Hamiltonian circuit.	$O(n^n)$

Topics

- General Method
- N-Queens Problem
- Sum of Subsets
- Graph Coloring
- Hamiltonian Cycles