

UNIT-II

BACKTRACKING

BACKTRACKING

- It is one of the most general algorithm design techniques.
- 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.
- To apply backtracking method, the desired solution must be expressible as an n-tuple $(x_1 \dots x_n)$ 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 $P(x_1 \dots x_n)$.
- The major advantage of this method is, once we know that a partial vector (x_1, \dots, x_i) will not lead to an optimal solution that (m_{i+1}, \dots, m_n) possible test vectors may be ignored entirely.
- Many problems solved using backtracking require that all the solutions satisfy a complex set of constraints.

- **These constraints are classified as:**

i) Explicit constraints.

ii) Implicit constraints.

- 1) **Explicit constraints:** Explicit constraints are rules that restrict each x_i to take values only from a given set.

Some examples are,

$x_i \geq 0$ or $S_i = \{\text{all non-negative real nos.}\}$

$x_i = 0$ or 1 or $S_i = \{0, 1\}$.

$L_i \leq x_i \leq U_i$ or $S_i = \{a: L_i \leq a \leq U_i\}$

- All tuples that satisfy the explicit constraint define a possible solution space for I .
- 2) **Implicit constraints:** The implicit constraint determines which of the tuples in the solution space of I satisfy the criterion function. The implicit constraints describe the way in which the x_i must relate to each other.

Example : In 4 – queen problem, The implicit constraints are no queens can be on the same column, same row and same diagonal.

CONTROL ABSTRACTION OF BACKTRACKING:

The control abstraction is also called as general method for backtracking is as follows. Let $(X_1, X_2, X_3, \dots, X_i)$ be a path from the root to a node in a state space tree. Let $(X_1, X_2, X_3, \dots, X_i)$ be the set of all possible values for x_{i+1} such that $(X_1, X_2, X_3, \dots, X_{i+1})$ is also a path to a problem state. We assume the existence of bounding function B_{i+1} such that if $B_{i+1}(X_1, X_2, X_3, \dots, X_{i+1})$ is false for a path $(X_1, X_2, X_3, \dots, X_{i+1})$ from the root node to problem state, then the path cannot be extended to reach an answer node.

Algorithm : Backtracking (k)

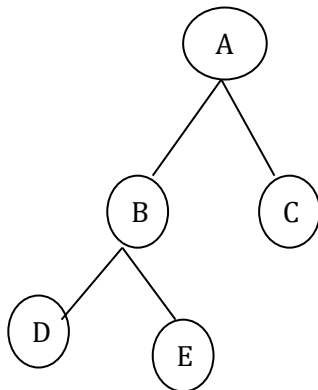
```
{
  for ( each  $x[k] \in T(x[1], \dots, x[k-1])$  ) do
  {
    if (  $B_k(x[1], x[2], \dots, x[k]) \neq 0$  ) then
    {
      if (  $x[1], x[2], \dots, x[k]$  is a path to an answer node ) then
        write (  $x[1 : k]$  );
      if (  $k < n$  ) then
        Backtrack ( k + 1 );
    }
  }
}
```

BASIC TERMINOLOGY OF BACK TRACKING :

1 : Criterion Function : it is a function $P(x_1, x_2, x_3, \dots, x_n)$ which needs to be minimized or maximized for a given problem.

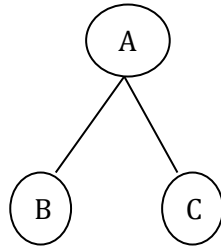
2 : Solution Space : All tuples that satisfy the explicit constraints define a possible solution space for a particular instance 'I' of the problem.

Ex :



Here, ABD, ABE, AC are the tuples in solution space.

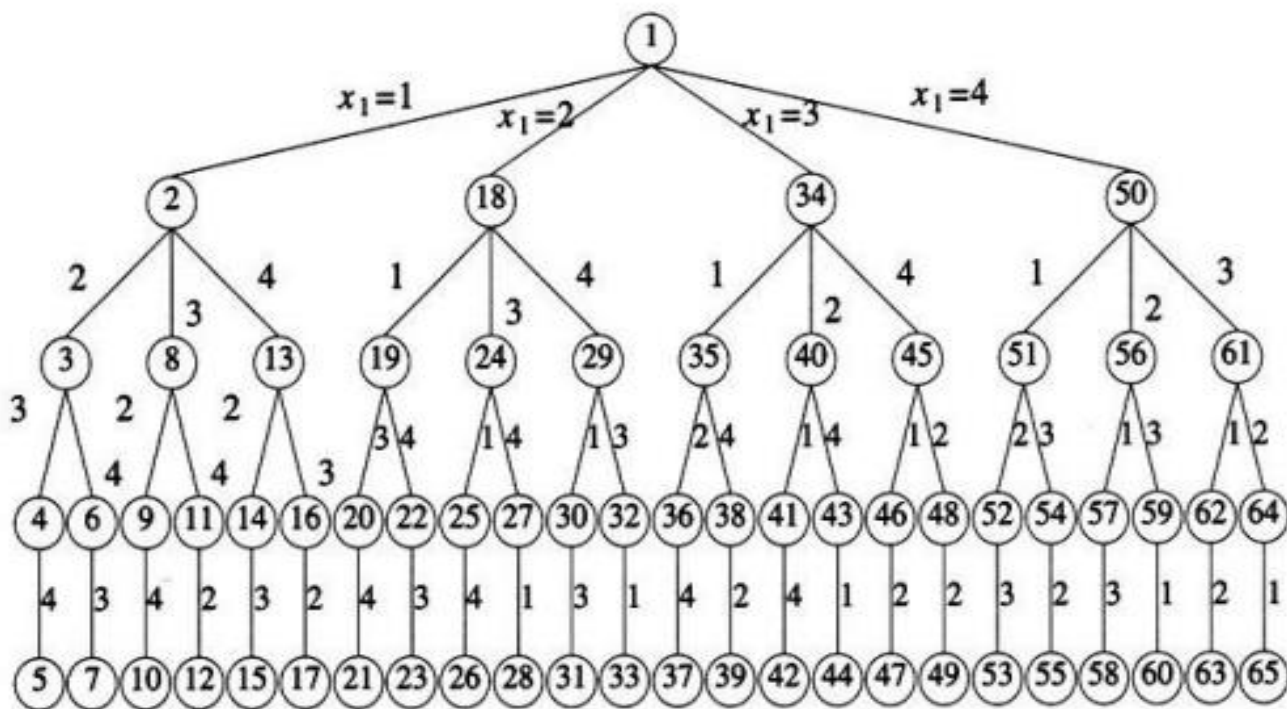
3 : Problem State : Each node in this tree defines a problem state.



So, the nodes A, B, C are problem state.

4 : State Space : State Space is the set of all paths from root node to other nodes.

Example : State space tree of a 4 – queen problem.



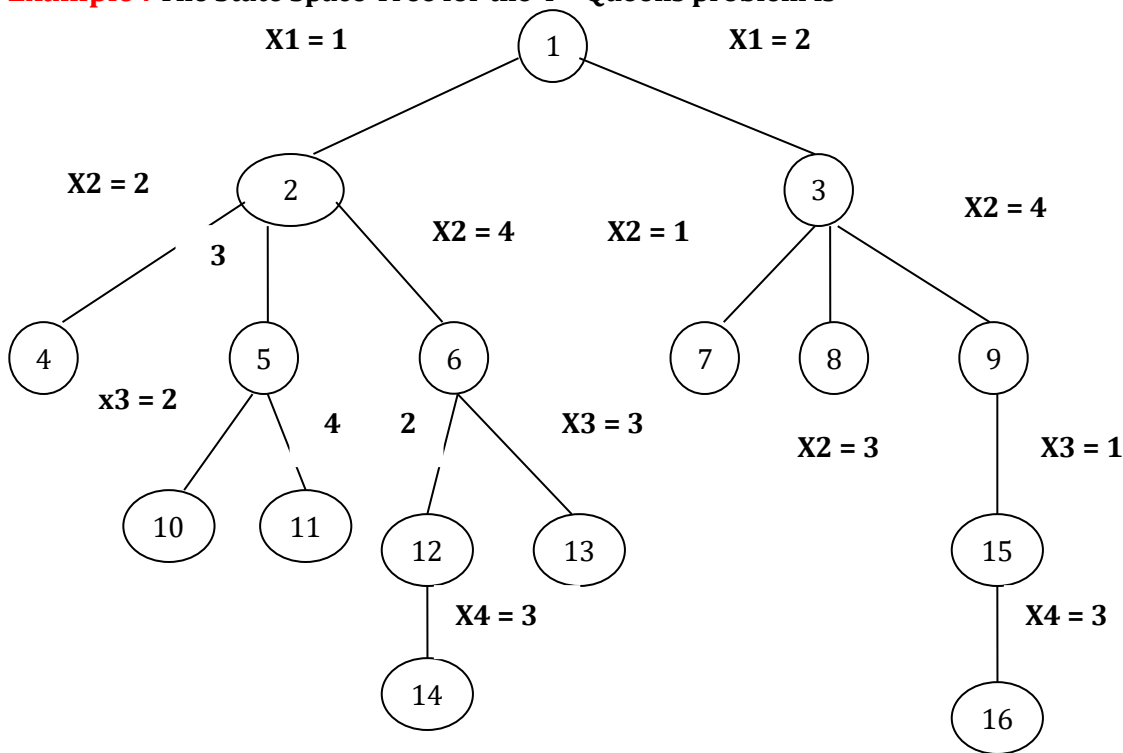
Tree organization of 4-queens solution space

5 : Solution States : Solution states are the problem states s for which the path from the root node to s defines a tuple in the solution space

- In variable tuple size formulation tree, all nodes are solution states
- In fixed tuple size formulation tree, only the leaf nodes are solution states
- Partitioned into disjoint sub-solution spaces at each internal node

6 : Answer 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 of the problem.

Example : The State Space Tree for the 4 - Queens problem is



These Solution States represented in the form of tuples i.e, (1, 3, 9, 15, 16) are the solution states and Here 16 is Answer States

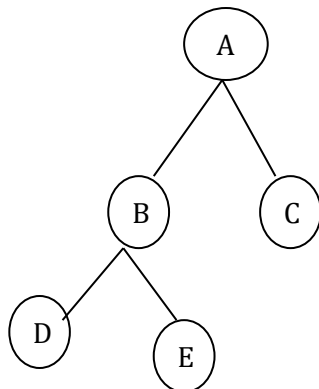
7 : Live Node : A node which has been generated and all of whose children have not yet been generated is called Live Node.

Ex1 :



Here node A is called Live node since the children of node A have not yet been generated.

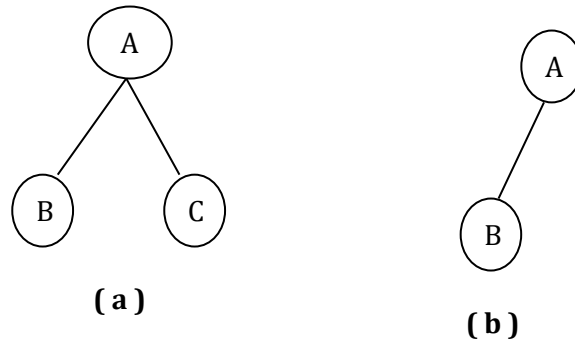
Ex 2 :



Here, nodes D, E, C are live nodes because the children of these nodes are not yet been generated.

8 : E – Node : The live node whose children are currently being generated is called E – node.

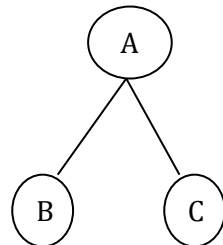
Ex :



Here Node A is E – node.

9 : Dead Node : A node which is not to be expanded further or all of whose children have been generated is called Dead Node.

Ex 1 :



Nodes A, B,C are dead nodes. Since node A is children generated and nodes B,C are not expanded.

10 : Bounding Function : is a function used to kill live nodes without generating all their children.

Applications of Backtracking :

- 1 : N – Queen Problems.
- 2 : SumOfSubset Problem.
- 3 : Graph Coloring Problems.
- 4 : Hamiltonian Graph Problem.

1 : N QUEENS PROBLEMS (4 – QUEENS AND 8 – QUEENS PROBLEM) : Consider an $n * n$ chessboard. Let there are N queens. These n – queens are to be placed on the $n * n$ chessboard so that no two queens are on the same column, same row or same diagonal.

Algorithm : NQueens(k, n)

```
{
  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 place (k, i)

```
{
  for j = 1 to  $k - 1$  do
  {
    If( $X[k] = i$ ) then
      return false;
    else
      retrn true;
  }
}
```

4 – QUEENS PROBLEM : Consider a $4 * 4$ chessboard let there are 4 – queens. These 4 – queens are to be placed on the $4 * 4$ chessboard . So that no two queens are on the same column, same row or same diagonal position. The explicit constraints are 4 – queens are to be placed on $4 * 4$ chessboard in 4 ways. The implicit constraints are no two queens are in the same row or column or diagonal position.

Let $\{ x_1, x_2, x_3, x_4 \}$ be the solution vector where x_i , column number on which the queen i is placed.

Step 1 : First queen Q_1 is placed in first row and first column.

Q1			

Step 2 : The second queen should not be placed in first row and second column. It should be placed in second row and in second, third or fourth column. If we place in second column, both will be in same diagonal, so place it in third column.

Q1			
*	*	Q2	

Step 3 : We are unable to place Q3 in third row .

Q1			
*	*	Q2	
*	*	*	*

Then go back to Q2 and place it somewhere else and then place Q3 in 3rd row, 2nd column.

Q1			
			Q2

Q1			
			Q2
	Q3		

Step 4 : The fourth queen should be placed in 4th row and 3rd column but there will be a diagonal attack from Q3. so go back, remove Q3 and place it in the next column. But it is not possible, so move back to Q2 and remove it to next column but it is not possible. So go back to Q1 and move it to next column. It can be shown as follows.

	Q1		

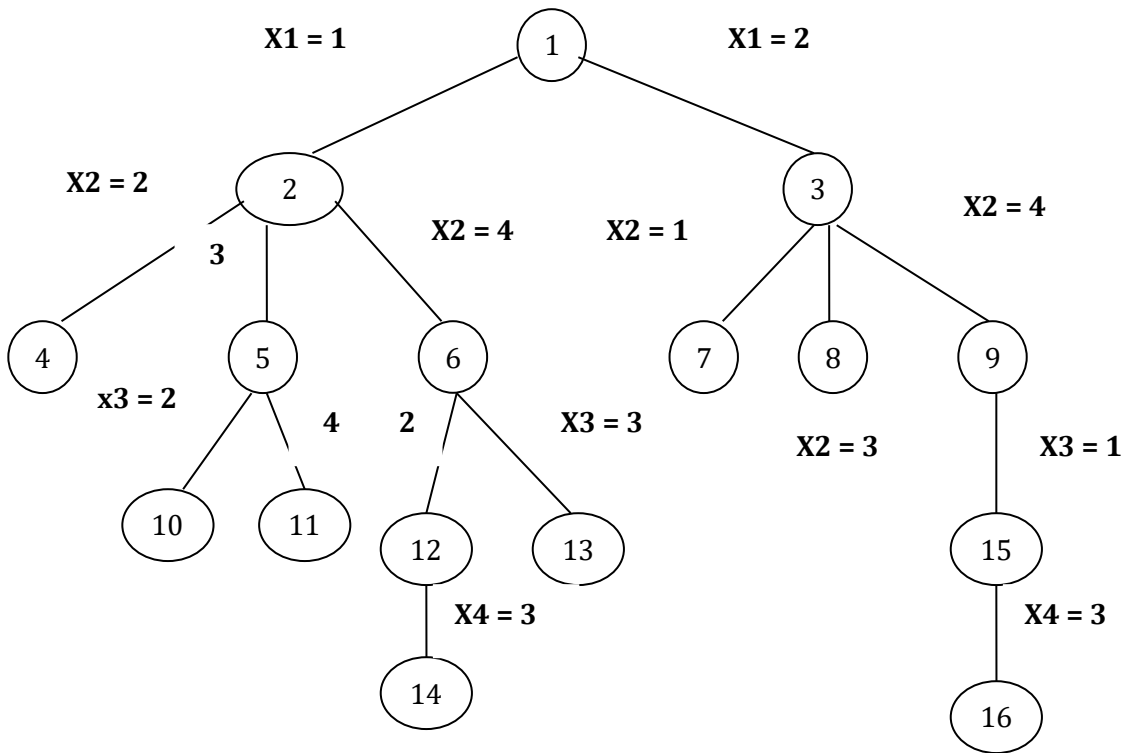
	Q1		
			Q2

	Q1		
			Q2
Q3			

	Q1		
			Q2
Q3			
		Q4	

Hence the solution to 4 – queen’s problem is $x_1 = 2$, $x_2 = 4$, $x_3 = 1$, $x_4 = 3$. i.e, first queen is placed in 2nd column, second queen is placed in 4th column and third queen is placed in first column and fourth queen is placed in third column.

The State Space Tree for the 4 – Queens problem is



2.SUM OF SUBSETS : We are given n distinct positive numbers (usually called weights) and we desire to find all combinations of these numbers whose sums are m . This is called Sum of Subsets problem.

Example 1 : if given set $S = (1, 2, 3, 4)$ and $M = 5$, then

there exists sets $S' = (3, 2)$ and $S' = (1,4)$ whose sum is equal to M

Example 2 : if a given set $S = (1, 3, 5)$ and $M = 7$, then

No subset occurs for which the sum is equal to $M = 7$.

We could formulate Sum Of Subsets Problem using either fixed or variable sized tuples. Now we consider a backtracking solution using the fixed sized tuple. In this case the element x_i of the solution vector is either zero or one depending on whether the weight w_i is included or not.

The children of any node are easily generated. For a node at level I the left child corresponds to $x_i = 1$ and the right child corresponds to $x_i = 0$.

Algorithm : SumOfSub(s, k, r)

```
{
    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]);
    if ( ( s + r - w[k] ≥ m ) and ( s + w[k+1] ≤ m ) then
    {
        x[k] = 0;
        SumOfSub( s, k+1, r - w[k]);
    }
}
```

Example1: Let $n=3$, $w = \{2, 4, 6\}$ and $m = 6$, Find all possible subsets of w that sum to m .

Solution : $n = 3$

$M=6$

$w_1 = 2, w_2=4, w_3=6$

Initially three variables

$S = 0$

$k = 1$

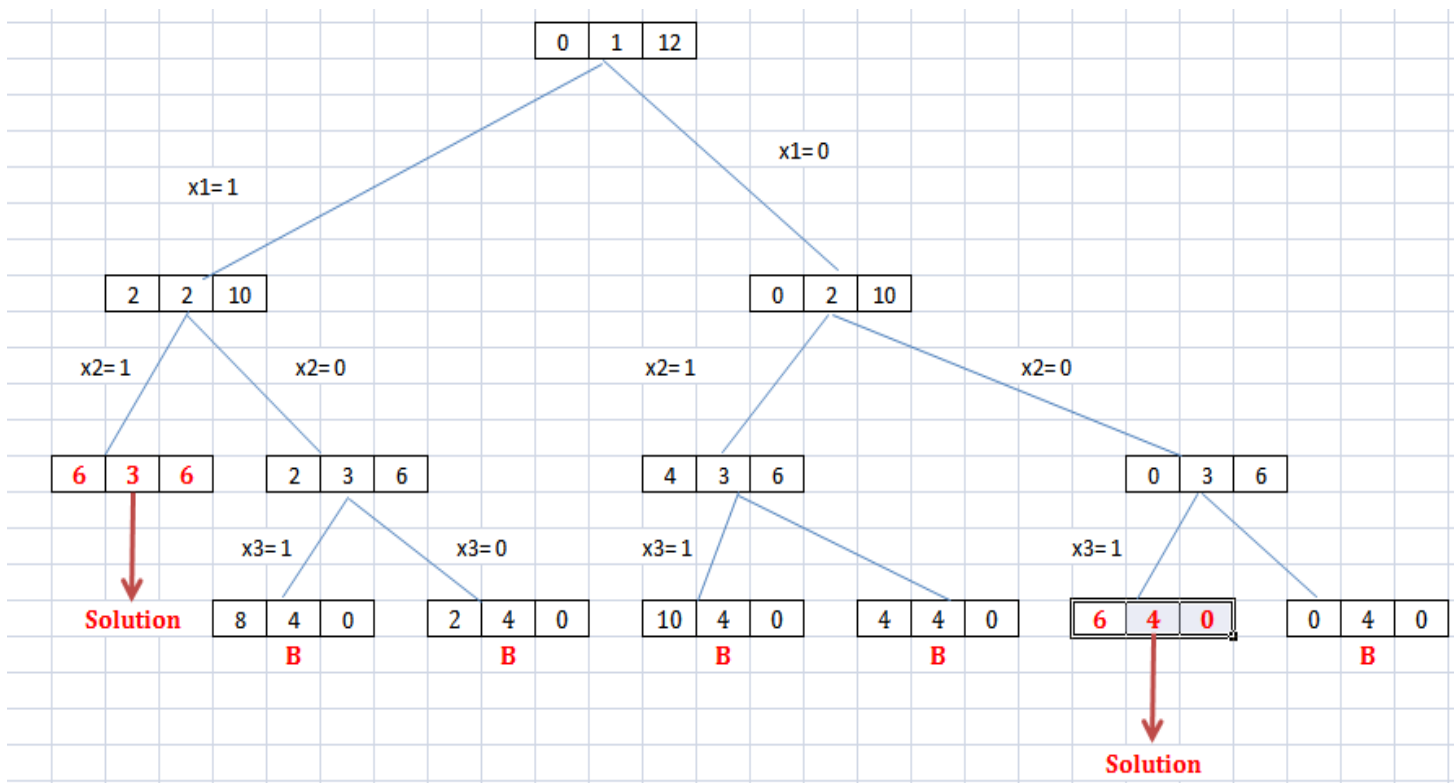
$r = (2 + 4 + 6) = 12$

To calculate Left child and Right child

$x_i = 1$ (Left Child) $= s + w[k], k+1, r-w[k]$

$x_i = 0$ (Right Child) $= s, k+1, r-w[k]$

State space tree generated by Sum of Subsets Problem



There are Two answer states in the state space tree diagram. The answer states are denoted by A,B. There are 2 solutions

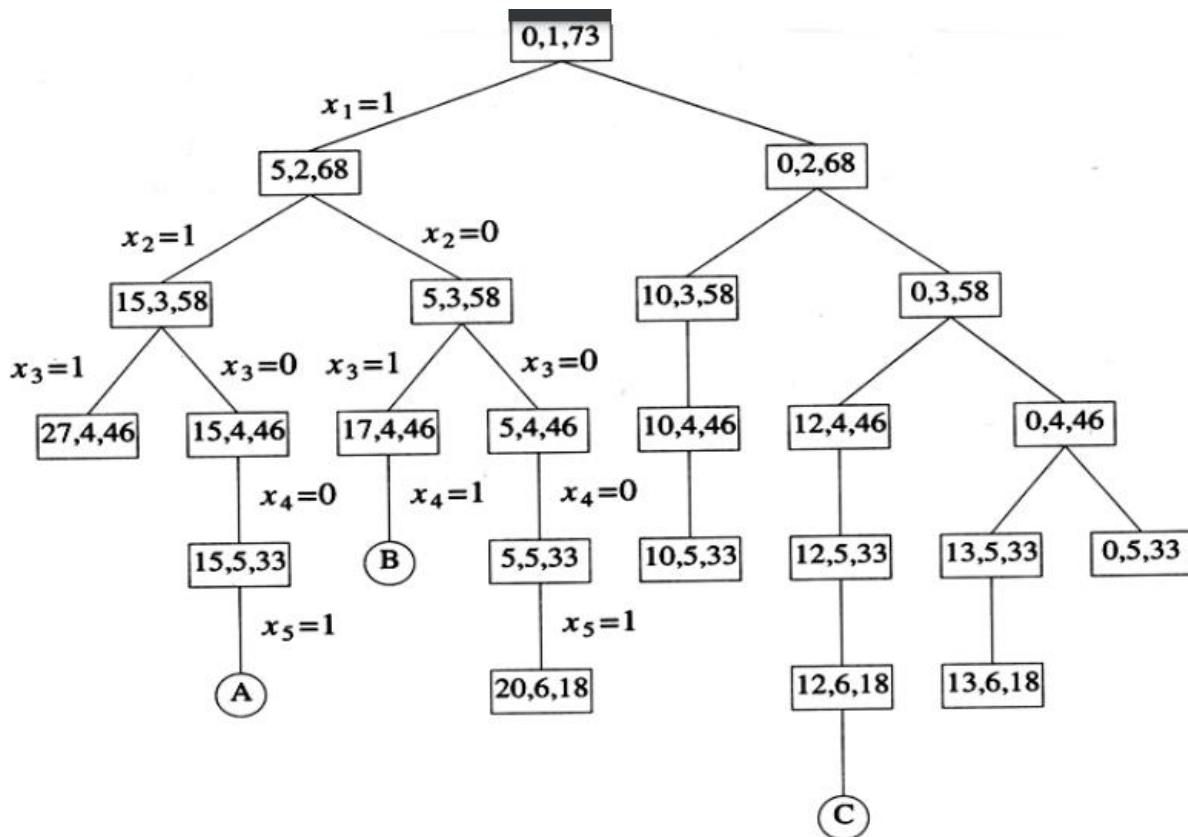
$A = \{ 1, 1, 0 \} = \{ 2 + 4 + 0 \} = 6$

$B = \{ 0, 0, 6 \} = \{ 0 + 0 + 6 \} = 6$

Example 2 : Let $n=6$, $w = \{ 5,10,12,13,15,18 \}$ and $m = 30$, Find all possible subsets of w that sum to m .

Solution : $n = 6$, $M=30$
 $w_1 = 5, w_2=10, w_3=12, w_4=13, w_5=15, w_6=18$
 Initially three variables
 $S = 0$
 $k = 1$
 $r = (5+10+12+13+15+18) = 73$
 To calculate Left child and Right child
 $x_i = 1$ (Left Child) $= s + w[k], k+1, r-w[k]$
 $x_i = 0$ (Right Child) $= s, k+1, r-w[k]$

State space tree generated by Sum of Subsets Problem



There are Three answer states in the state space tree diagram. The answer states are denoted by A,B,C. There are 3 solutions

$$A = \{ 1, 1, 0, 0, 1, 0 \} = \{ 5 + 10 + 0 + 0 + 15 + 0 \} = 30$$

$$B = \{ 1, 0, 1, 1, 0, 0 \} = \{ 5 + 0 + 12 + 13 + 0 + 0 \} = 30$$

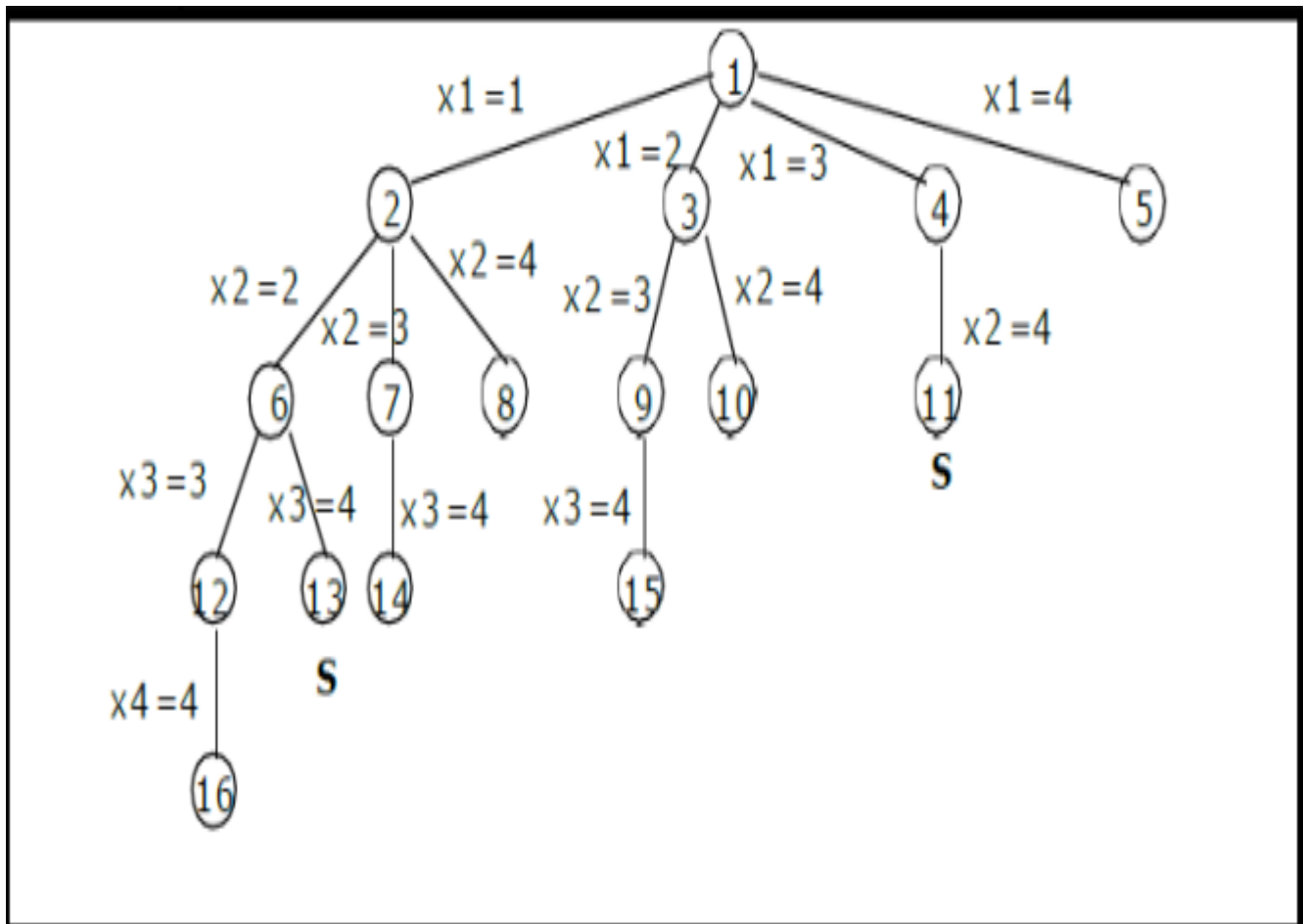
$$C = \{ 0, 0, 1, 0, 0, 1 \} = \{ 0 + 0 + 12 + 0 + 0 + 18 \} = 30$$

Variable Size Tuple : The tree corresponds to the variable tuple size formulation. The edges are labeled such that an edge from a level i node to a level $i+1$ node represents a value for x_i . At each node, the solution space is partitioned into sub - solution spaces. All paths from the root node to any node in the tree define the solution space, since any such path corresponds to a subset satisfying the explicit constraints.

The possible paths are (1), (1, 2), (1, 2, 3), (1, 2, 3, 4), (1, 2, 4), (1, 3, 4), (2), (2,3), and so on. Thus, the left most sub-tree defines all subsets containing w_1 , the next sub-tree defines all subsets containing w_2 but not w_1 , and so on.

Example: Let $n = 4$, $w = (11, 13, 24, 7)$ and $m = 31$, by using Variable Sized tuple

Solution: State space tree generated by using variable sized tuple

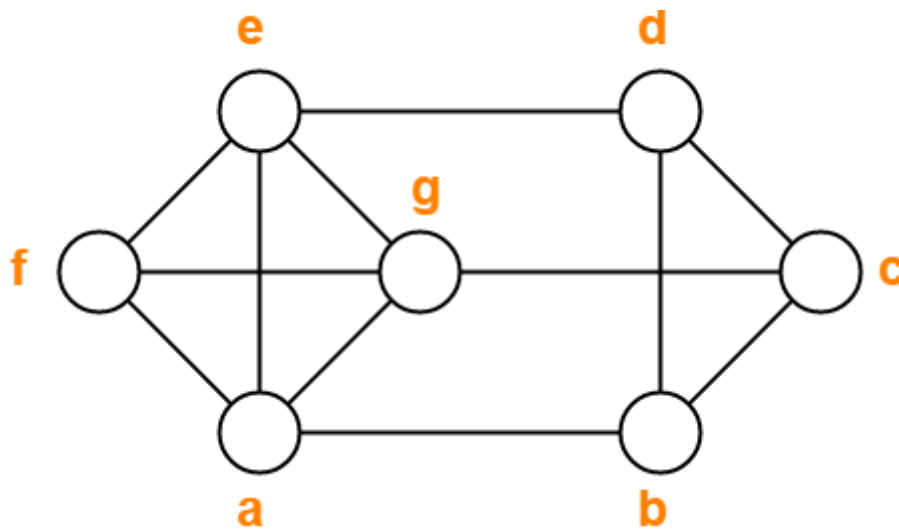


The desired subsets are (11,13, 7) and (24,7).

GRAPH COLOURING PROBLEM :

- Let G be a graph and m be a given positive integer.
- The graph coloring problem is to discover whether the nodes of G can be colored in such a way that no two adjacent nodes / vertices have the same color yet only m colors are used.
- This graph coloring problem is also known as m -colorability decision problem
- The m -Coloring 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.
- **Example**, the graph can be colored with 4 colors. Hence the chromatic number of the graph is 4.
- Graph coloring problem is a NP Complete problem.
- If d is the degree of the given graph, then it can be colored with $d+1$ colors.

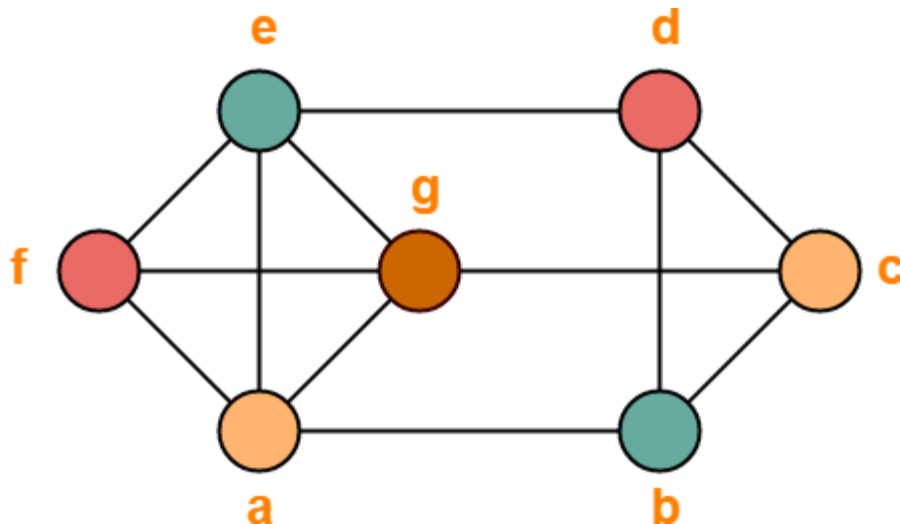
Example : Consider the following graph



From here,

- Minimum number of colors used to color the given graph are 4.
- Therefore, Chromatic Number of the given graph = 4.

The given graph may be properly colored using 4 colors as shown below-



Algorithm for Graph Coloring Problem

Algorithm mColoring(k)

```
{
    repeat
    {
        NextValue(k);
        if(x[k]=0) then
            return;
        if(k=n) then
            write(x[1:n]);
        else
            mColoring(k+1);
    }
    until false;
}
```

Algorithm NextValue(k)

```
{
    repeat
    {
        x[k] = (x[k]+1)mod(m+1);
        if(x[k]=0) then
            return;
        for j =1 to n do
        {
            if((G[k,j] ≠ 0) and (x[k] = x[j])) then
                break;
        }
        if(j=n+1) then
            return;
    }
    until(false);
}
```

Graph Coloring Applications-

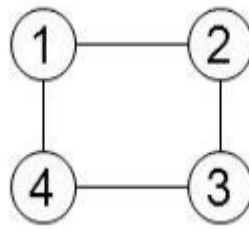
Some important applications of graph coloring are as follows-

- Map Coloring
- Scheduling the tasks
- Preparing Time Table
- Assignment
- Conflict Resolution
- Sudoku

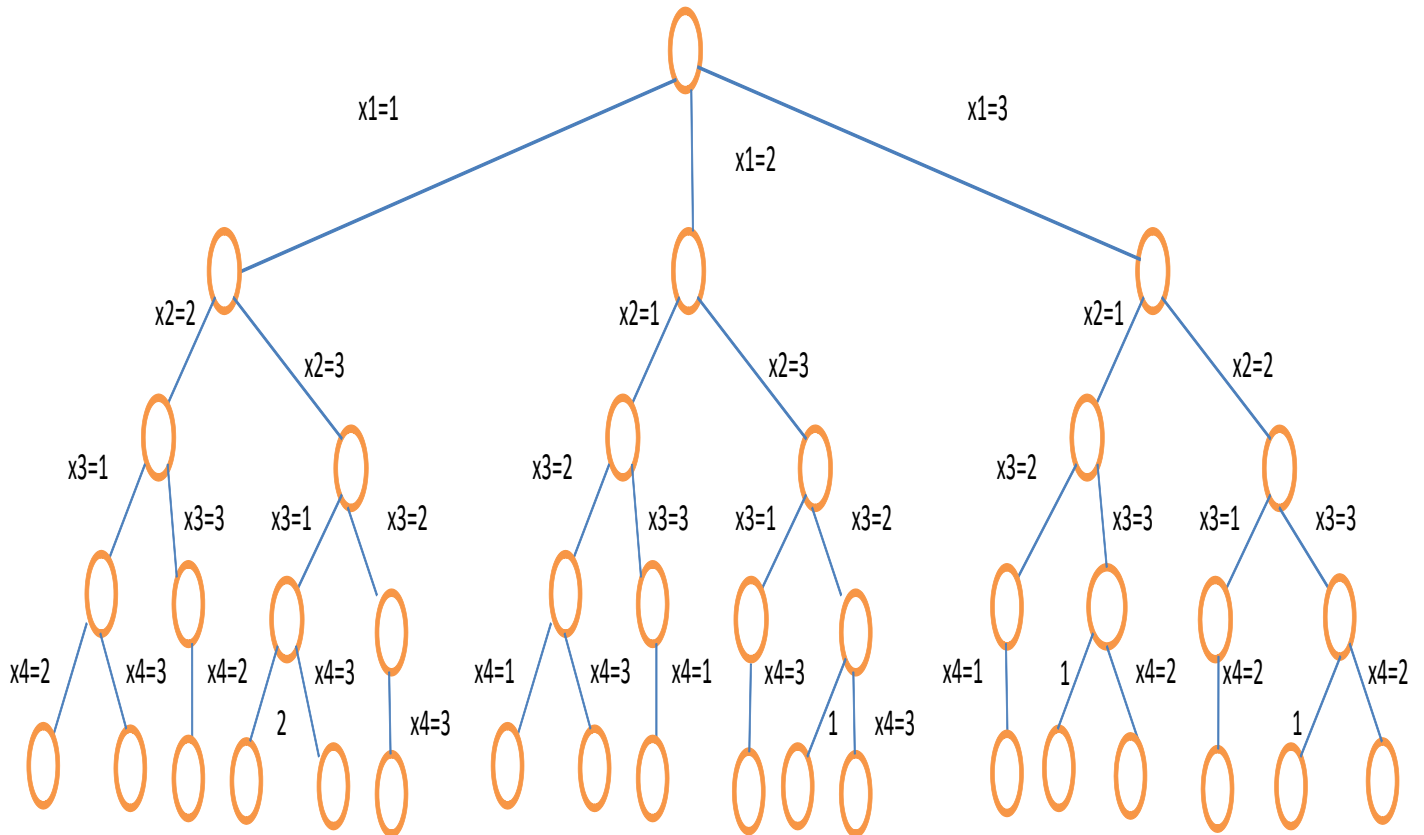
Time Complexity: $O(mV)$.

Space Complexity: $O(V)$

Example : A 4-node graph and all possible 3-colorings



Solution : State Space tree for graph coloring problem (n=4, m=3)



Here The Solutions are

- | | | |
|----------------|----------------|----------------|
| (1, 2, 1, 3) | (2, 1, 2, 3) | (3, 1, 2, 1) |
| (1, 2, 3, 2) | (2, 1, 3, 1) | (3, 1, 3, 2) |
| (1, 3, 1, 2) | (2, 3, 1, 3) | (3, 2, 1, 2) |
| (1, 3, 2, 3) | (2, 3, 2, 1) | (3, 2, 3, 1) |

HAMILTONIAN GRAPH

Definition : A Hamiltonian graph is the directed or undirected graph containing a Hamiltonian cycle. The Hamiltonian cycle is the cycle that traverses all the vertices of the given graph G exactly once and then ends at the starting vertex.

- Given a graph $G = (V, E)$ we have to find the Hamiltonian Circuit using Backtracking approach.
- We start our search from any arbitrary vertex say 'a.' This vertex 'a' becomes the root of our implicit tree.
- The first element of our partial solution is the first intermediate vertex of the Hamiltonian Cycle that is to be constructed.
- The next adjacent vertex is selected by alphabetical order. If at any stage any arbitrary vertex makes a cycle with any vertex other than vertex 'a' then we say that dead end is reached.
- In this case, we backtrack one step, and again the search begins by selecting another vertex and backtrack the element from the partial solution must be removed.
- The search using backtracking is successful if a Hamiltonian Cycle is obtained

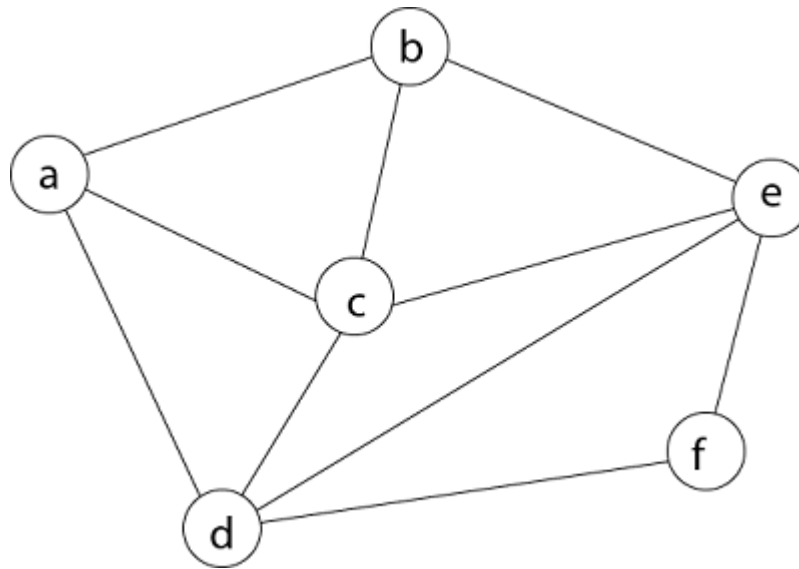
Algorithm Hamiltonian(k)

```
{
    repeat
    {
        NextValue(k);
        if(x[k]=0) then
            return;
        if(k=n) then
            write(x[1:n]);
        else
            Hamiltonian (k+1);
    }
    until false;
```

}
Algorithm NextValue(k)

```
{
    repeat
    {
        x[k] = (x[k]+1)mod(n+1);
        if(x[k]=0) then return;
        if(G[x[k-1],x[k]] ≠ 0) then
        {
            for j =1 to k-1 do
                if(x[j] = x[k])) then
                    break;
            if(j=k) then
                if((k<n) or ((k=n) and G[x[n], x[1]] ≠ 0)) then
                    return;
        }
    }
    until(false);
}
```

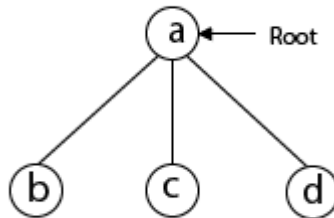
Example : The following graph illustrates the presence of the Hamiltonian cycle in a graph G.



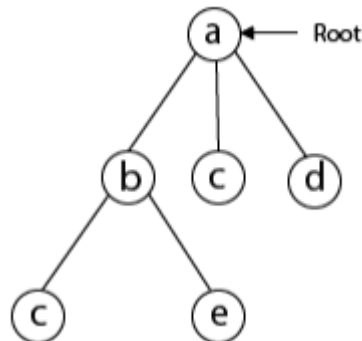
Step 1: Firstly, we start our search with vertex 'a.' this vertex 'a' becomes the root of our implicit tree.



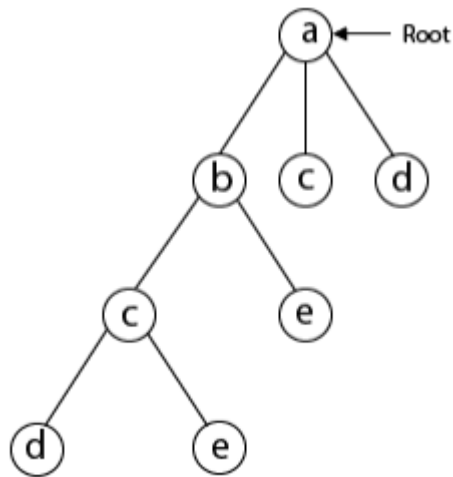
Step 2: Next, The vertex adjacent to 'a' is b, c and d. it comes first in lexicographical order (b, c, d).



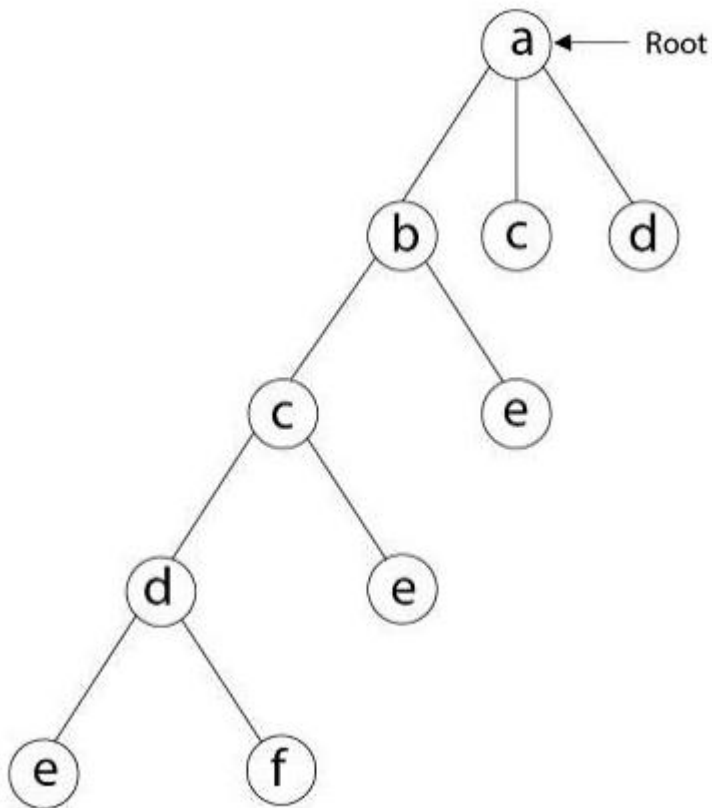
Step 3: Next, The vertex adjacent to 'b' is a, c and e. but 'a' is already visited.



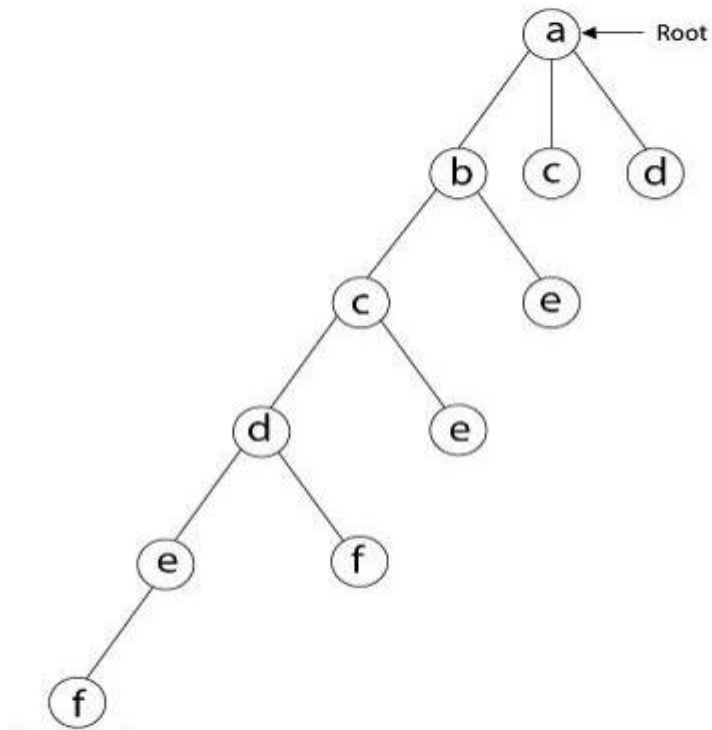
Step 4 : Next, The vertex adjacent to 'c' is a, b, d and e. but 'a' and 'b' are already visited.



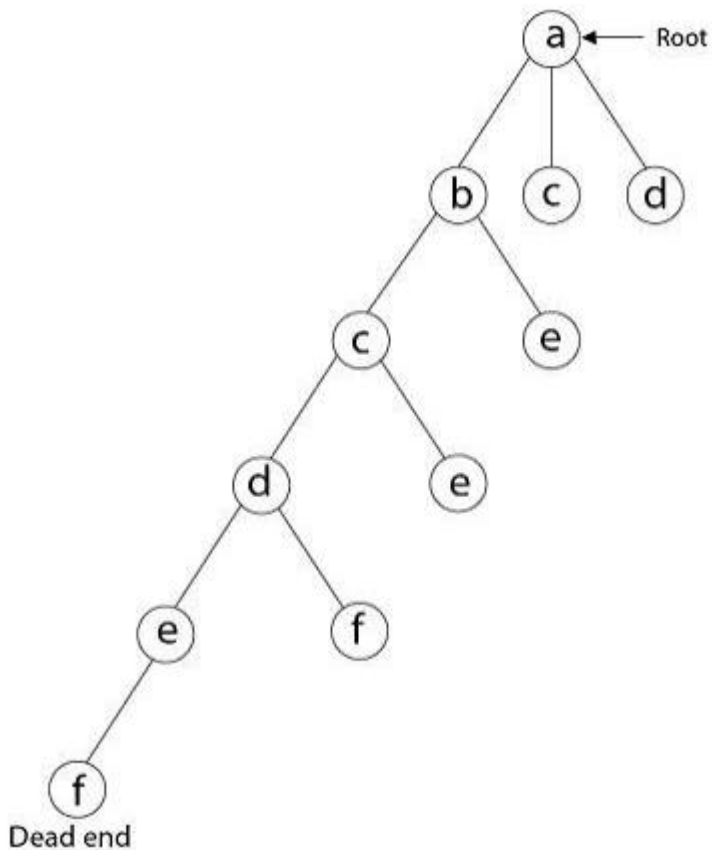
Step 5: Next, The vertex adjacent to 'd' is a, c, e and f. but 'a' and 'c' are already visited.



Step 6 : Next, The vertex adjacent to 'e' is b, c and f. but ' b' and 'c' are already visited.



Step 7 : Next, The vertex adjacent to 'f' is d and e, but 'd' and 'e' are already visited. Thus, we get the dead end, and we backtrack one step

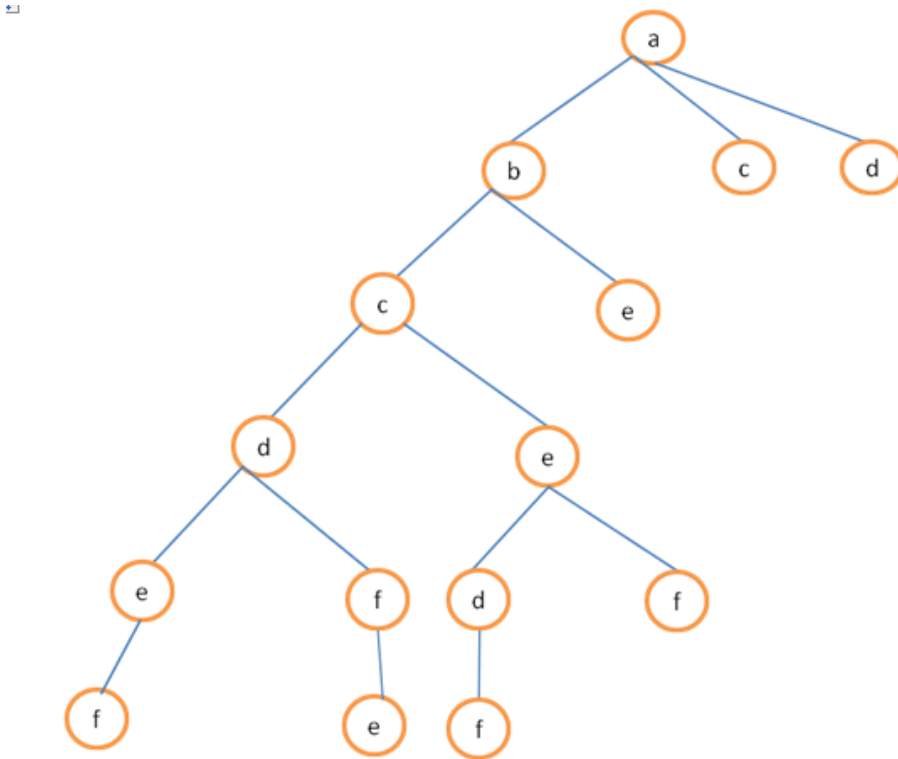




visited.



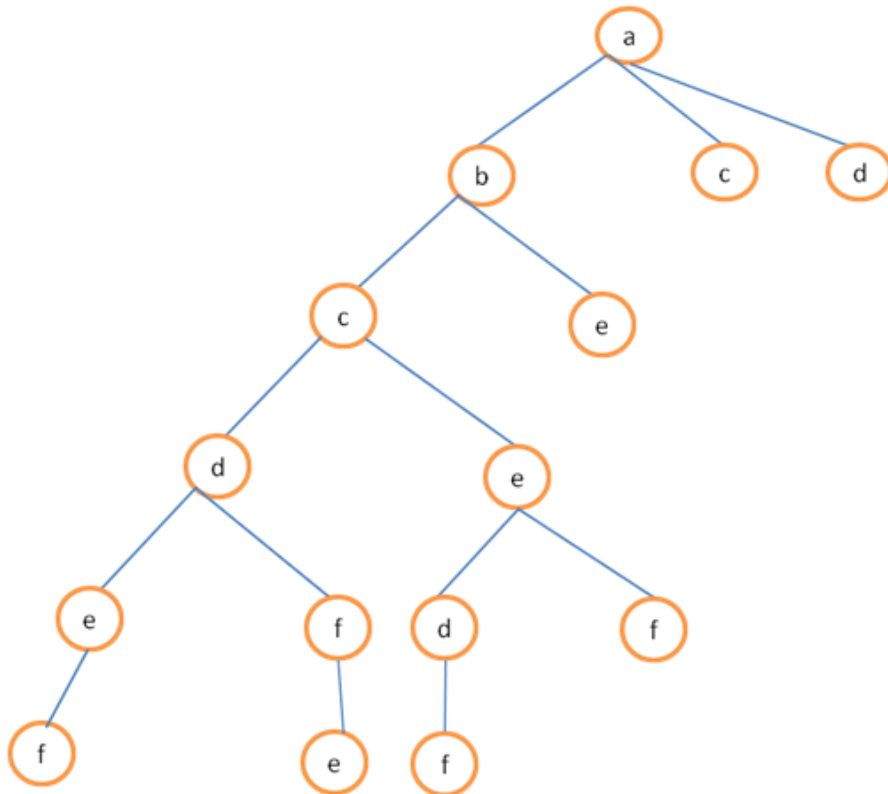
Step 9 : Next, The vertex adjacent to 'd' is a,c,e and f, but 'a','c' and 'e' are already visited.



Dead end

Dead end

Step10 : Next, The vertex adjacent to 'f' is d,e but 'd',and 'e' are already visited. Thus, we get the dead end

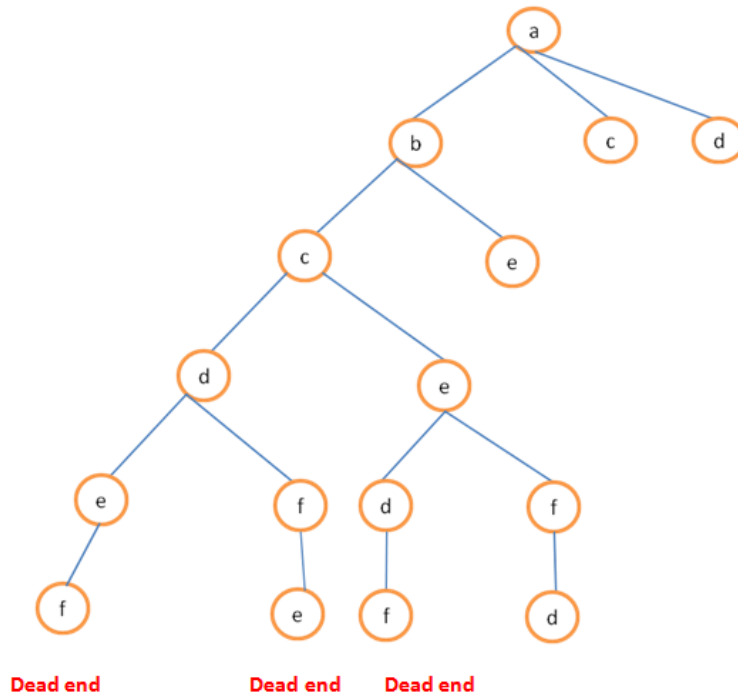


Dead end

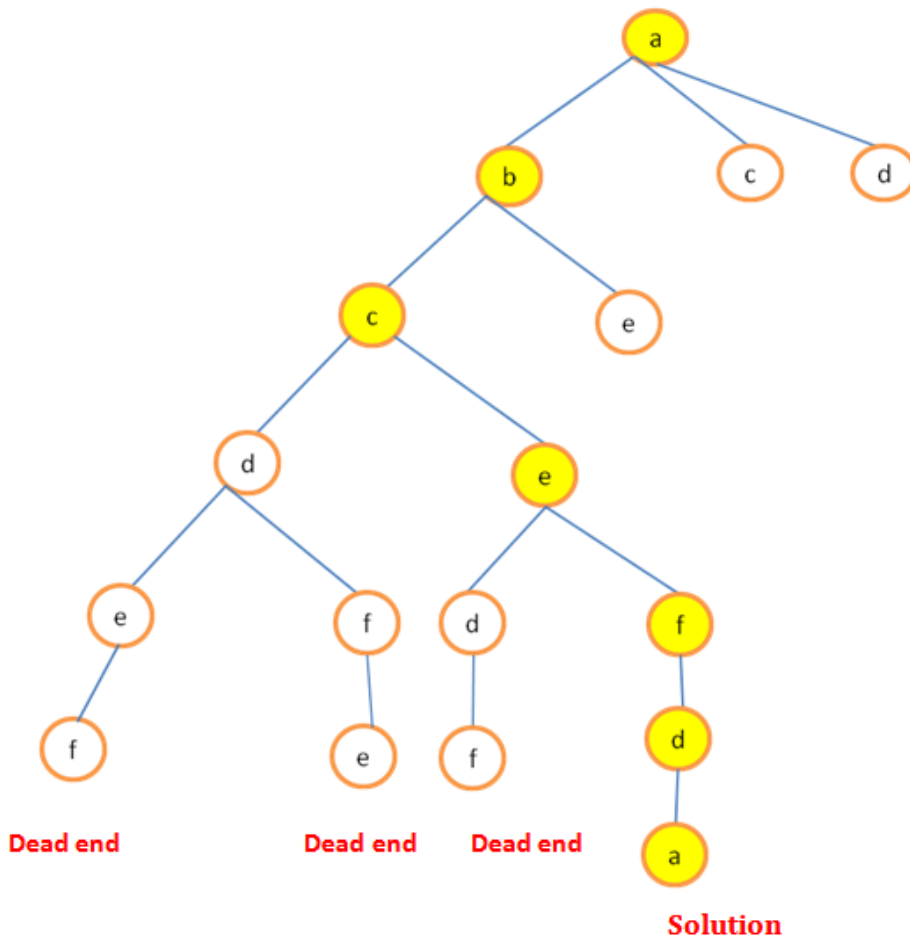
Dead end

Dead end

Step11 : Next, The vertex adjacent to 'f' is d,e but 'e' are already visited.



Step12 : Next, The vertex adjacent to 'd' is d,e but 'e' are already visited.



The Solution is : a - b - c - e - f - d - a