DESIGN AND ANALYSIS OF ALGORITHMS (DAA)

Unit-2

Dr. G. S. Naveen Kumar,
Dean, Quality
Associate Professor, CSE

DAA Syllabus

UNIT- II

Disjoint set operations, Union and find algorithms, AND/OR graphs, Connected Components and Spanning trees, Bi-connected components

Backtracking-General method, applications- The 8-queen problem, sum of subsets problem, graph coloring, Hamiltonian cycles.

Disjoint Set Operations

Set:

A set is a collection of distinct elements. The Set can be represented, for examples, as S1={1,2,5,10}.

Disjoint Sets:

The disjoints sets are those do not have any common element. For example S1={1,7,8,9} and S2={2,5,10}, then we can say that S1 and S2 are two disjoint sets.

Disjoint Set Operations:

The disjoint set operations are

1. Union
2. Find

In this representation each set is represented as a tree. Nodes are linked from the child to parent rather than usual method of linking from parent to child

Disjoint Set Operations

Disjoint set Union:

If Si and Sj are tow disjoint sets, then their union Si U Sj consists of all the elements x such that x is in Si or Sj.

Example:

S1={1,7,8,9} S2={2,5,10}

S1 U S2={1,2,5,7,8,9,10}

Find:

Given the element I, find the set containing i.

Example:

S1={1,7,8,9} S2={2,5,10} s3={3,4,6}

Then,

Find(4)= S3 Find(5)=S2 Find97)=S1

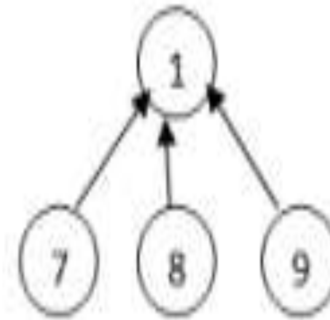
Set Representation

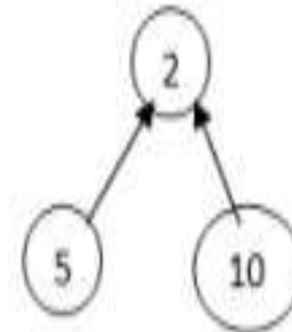
- The set will be represented as the tree structure where all children will store the address of parent / root node.
- The root node will store null at the place of parent address.
- In the given set of elements any element can be selected as the root node, generally we select the first node as the root node.

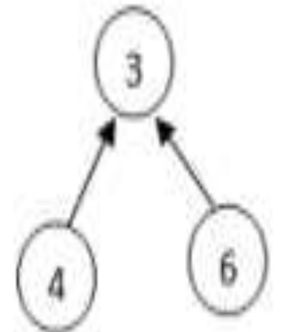
Example:

S1={1,7,8,9}   S2={2,5,10}   s3={3,4,6}

Then these sets can be represented as



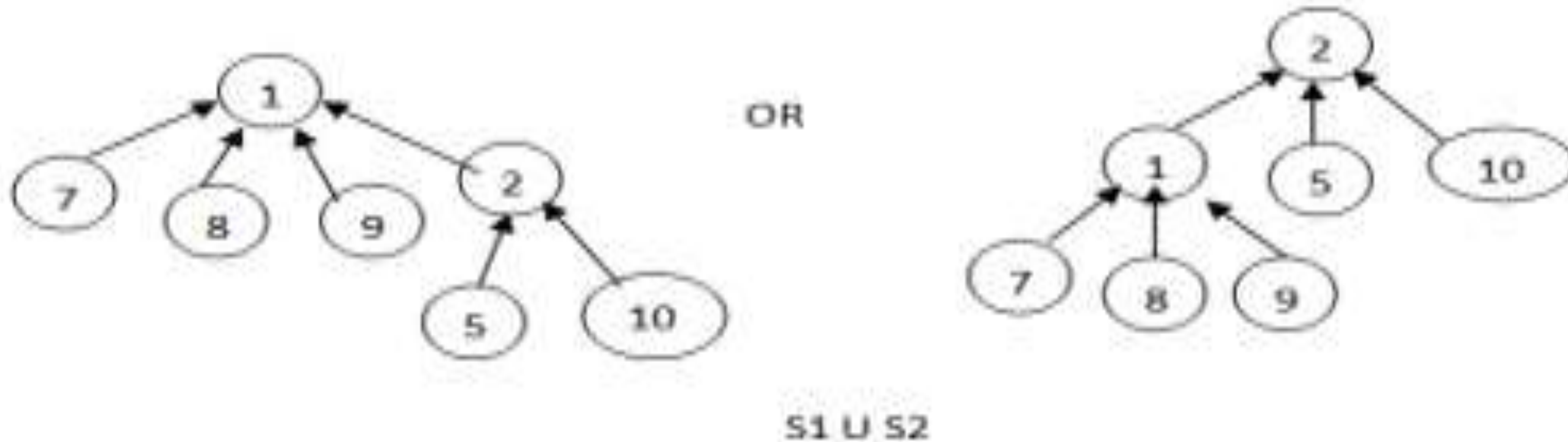




S1   S2   S3

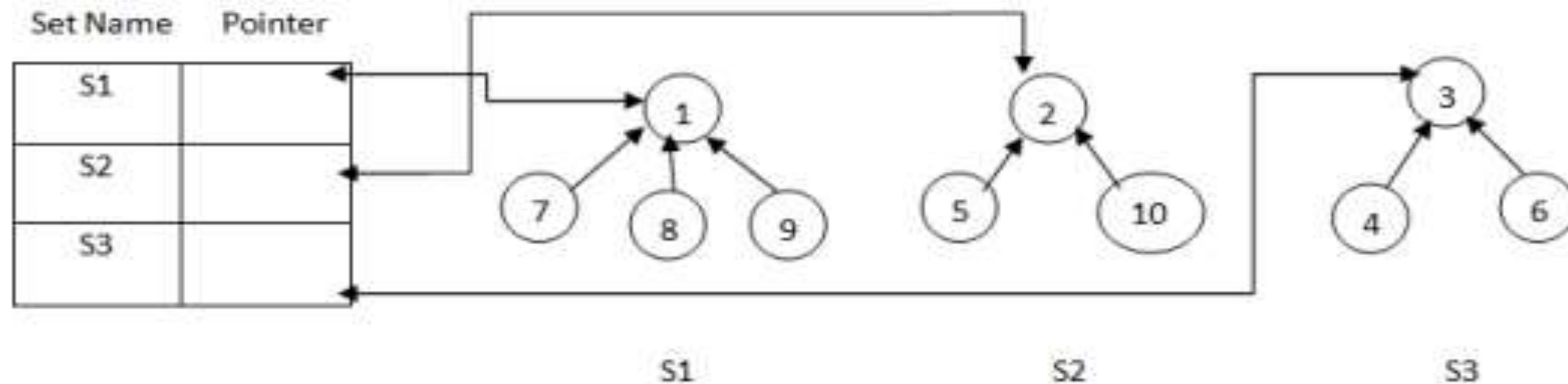
Disjoint Union:

- To perform disjoint set union between two sets Si and Sj can take any one root and make it sub-tree of the other.
- Consider the example sets S1 and S2 then the union of S1 and S2 can be represented as any one of the following.



Find:

- To perform find operation, along with the tree structure we need to maintain the name of each set.
- we require one more data structure to store the set names.
- The data structure contains two fields.
- One is the set name and the other one is the pointer to root.



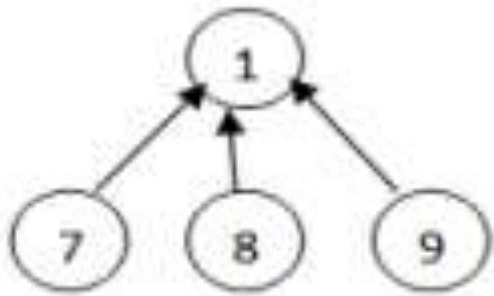
Union and Find Algorithms:

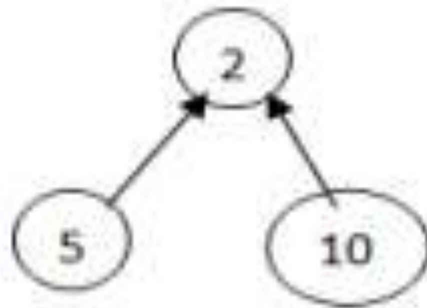
- In presenting Union and Find algorithms, we ignore the set names and identify sets just by the roots of trees representing them.
- To represent the sets, we use an array of 1 to n elements where n is the maximum value among the elements of all sets.
- The index values represent the nodes (elements of set)
- The entries represent the parent node. For the root value the entry will be ‘-1’.

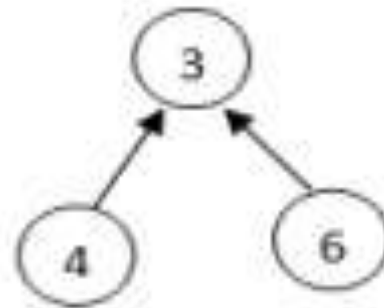
Union and Find Algorithms:

Example:

For the following sets the array representation is as shown below.







i	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
p	-1	-1	-1	3	2	3	1	1	1	2

Algorithm for Union operation:

To perform union the **SimpleUnion(i,j)** function takes the inputs as the set roots i and j . And make the parent of i as j i.e, make the second root as the parent of first root.

```
Algorithm SimpleUnion(i,j)
{
        P[i]:=j;
}
```

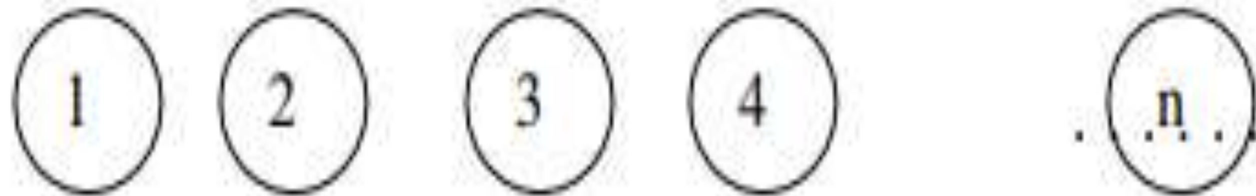
Algorithm for find operation:

The SimpleFind(i) algorithm takes the element i and finds the root node of i. It starts at I until it reaches a node with parent value -1.

```
Algorithms SimpleFind(i)
{
        while( P[i]≥0)
        do i:=P[i];
        return i;
}
```

Analysis of SimpleUnion(i,j) and SimpleFind(i):

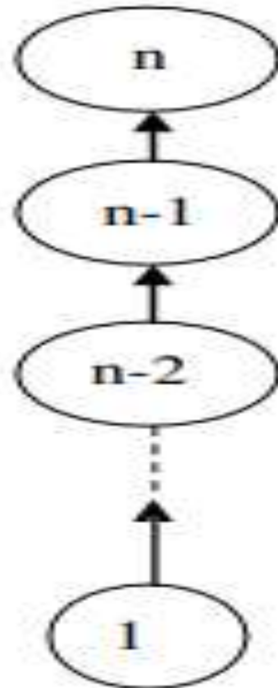
Although the SimpleUnion(i,j) and SimpleFind(i) algorithms are easy to state, their performance characteristics are not very good. For example, consider the sets



Process the following sequence of union operations

Union(1,2),Union(2,3) Union(n-1,n)

The sequence of Union operations results the degenerate tree as below.

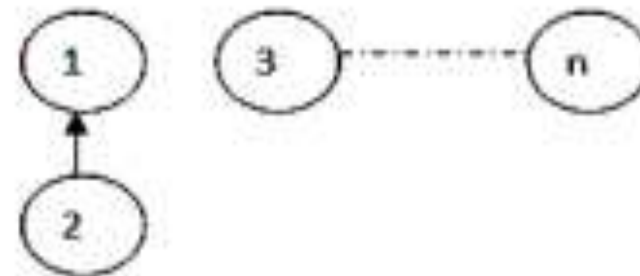


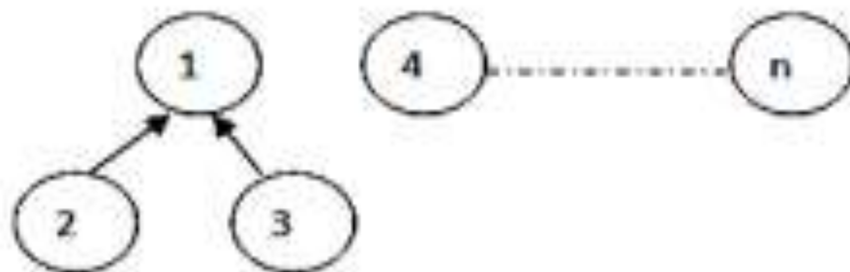
Since, the time taken for a Union is constant, the n-1 sequence of union can be processed in time O(n). And for the sequence of Find operations it will take time

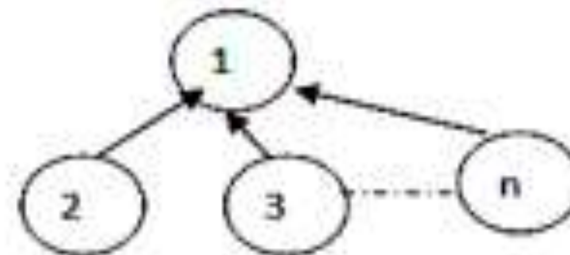
complexity of $O\left(\sum_{i=1}^{n} i\right) = O(n^2)$.

Weighting rule for Union:

If the number of nodes in the tree with root I is less than the number in the tree with the root j, then make 'j' the parent of i; otherwise make 'i' the parent of j.





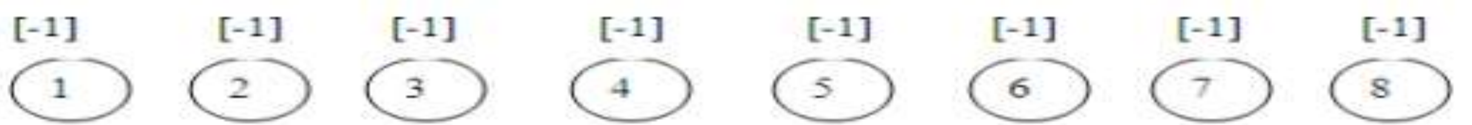


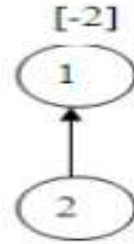
Algorithm Weighted Union(i,j)

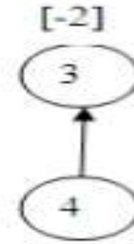
```
//Union sets with roots i and j , i≠j using the weighted rule
// P[i]=-count[i] and p[j]=-count[j]
{
temp:= P[i]+P[j];
if (P[i]>P[j]) then
{// i as fewer nodes P[i]:=j;
P[j]:=temp;
}
else
{// j has fewer nodes P[j]:=i;
P[i]:=temp;
}
}
```

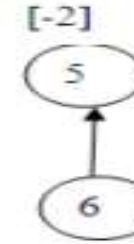
Collapsing rule for find:

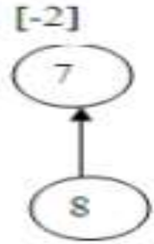
- If j is a node on the path from i to its root and p[i]≠root[i], then set P[j] to root[i].
- Consider the tree created by Weighted Union() on the sequence of 1≤i≤8. Union(1,2), Union(3,4), Union(5,6) and Union(7,8)





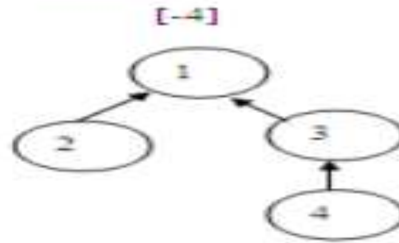




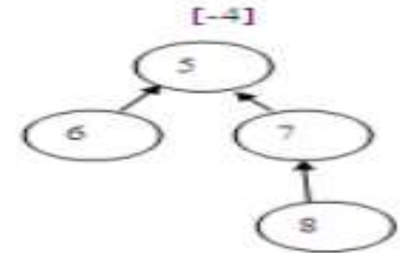


Union(1,2) Union(3,4) Union(5,6) Union(7,8)

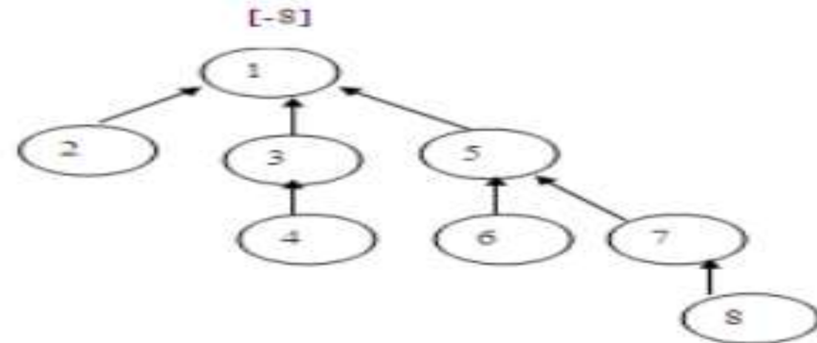
Union(1,3)



Union(5,7)



Union(1,5)



Algorithm CollapsingFind(i)

```
// Find the root of the tree containing element i
// use the collapsing rule to collapse all nodes from i to root.
{
r:=i;
while(P[r]>0) do
r:=P[r]; //Find root
while(i≠r)
{
//reset the parent node from element i
to the root s:=P[i];
P[i]:=r;
i:=s;
}
}
```

AND/OR Graph

Introduction

AND/OR Graph useful for representing the solution of problems that can be solved by decomposing them into a set of smaller problems, all of which must than be solved.

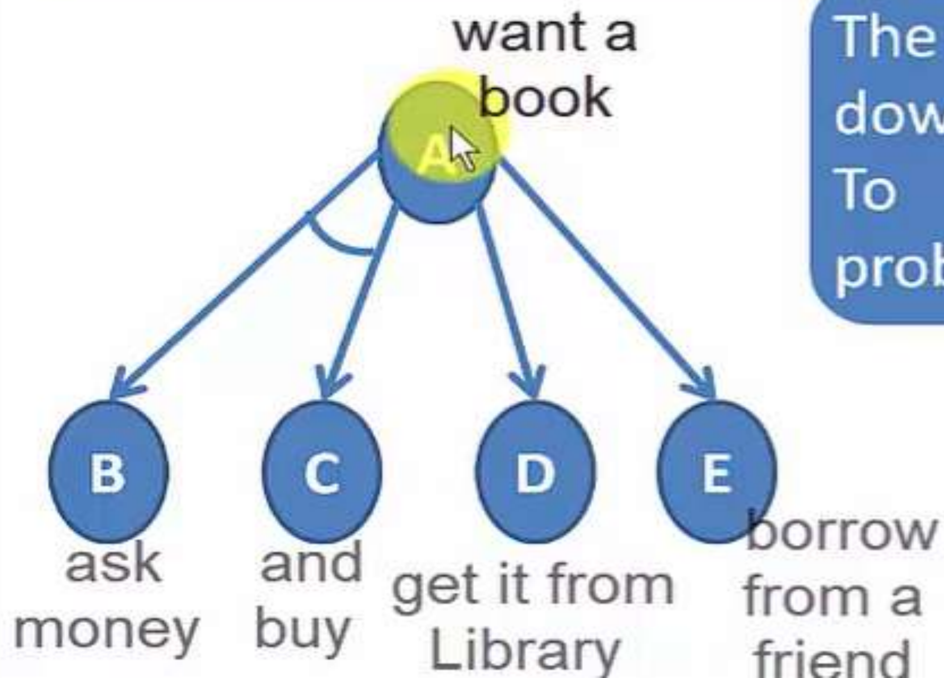
These problems are best represented as some sub-problems, some of which are solved simultaneously and independently (AND)

An **AND/OR tree** is a graphical representation of the reduction of problems (or goals) to conjunctions and disjunctions of sub-problems (or subgoals).

AND/OR Graph

Introduction

Suppose we want to solve problem A, let us represent this as the root of a tree.

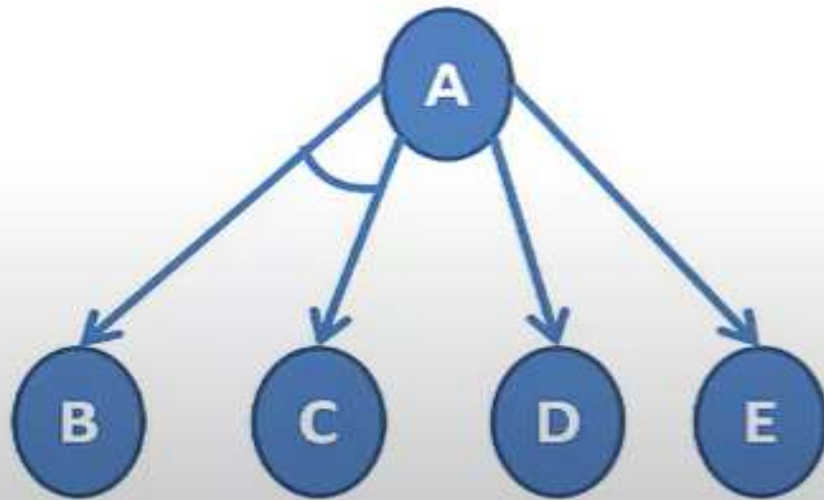


The problem A is solved by breaking down it into sub-problems B, C, D, E.
To solve problem A, solve both the problems B and C or solve D or solve E.

AND/OR Graph

The and-or tree represents the search space for solving a problem P.

The goal-reduction method is used for solving problem with AND/OR tree.



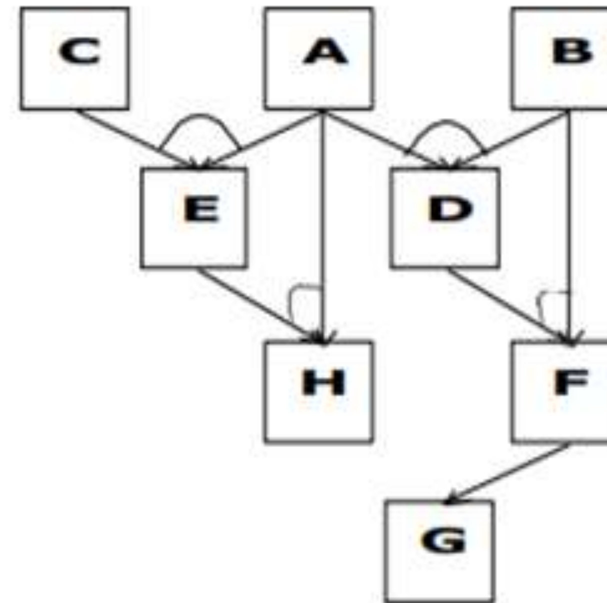
A if B and C
A if D
A if E

AND/OR Graph

Example 1:

Draw an AND/OR graph for the following prepositions:

1. A
2. B
3. C
4. A ^ B -> D
5. A ^ C -> E
6. B ^ D -> F
7. F -> G
8. A ^ E -> H



Example : Consider a boy who collect stamps (S).

He has for the purpose of exchange a winning match (M), a bat (B) and a toy (T). In his class there are friends who are also keen collectors of different items and will make the following exchanges:

1. 1 winning match (M) for a comic (C) and a pencil (P).
2. 1 winning match (M) for a bat (B) and a stamp (S).
3. 1 bat (B) for two stamps (S, S).
4. 1 toy (T) for two bats (B, B) and a stamp (S).

Objective: How to carry out the exchanges so that all his exchangeable items are converted into stamps (S).

1. 1 winning match (M) for a comic (C) and a pencil (P).

2. 1 winning match (M) for a bat (B) and a stamp (S).

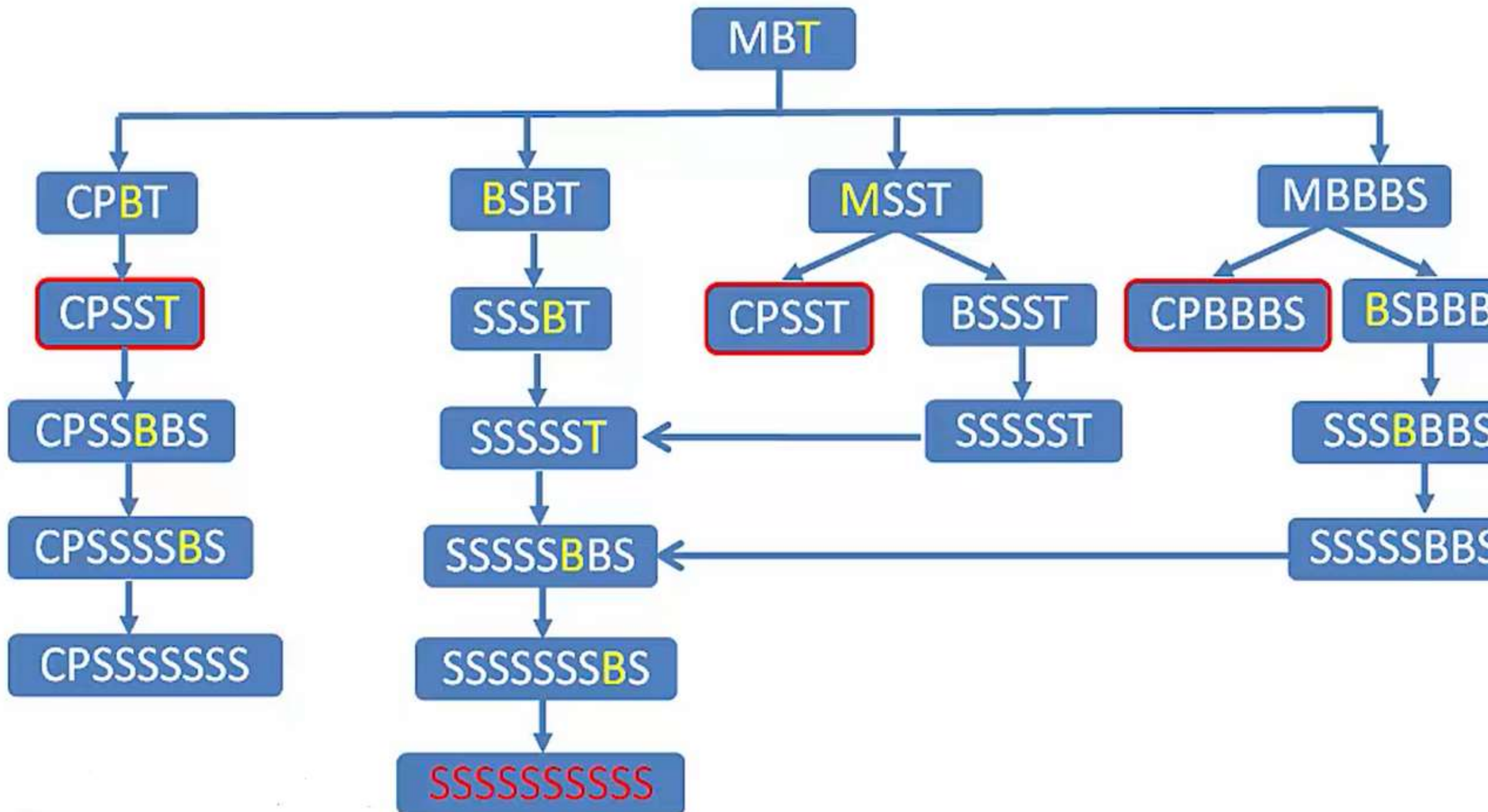
3. 1 bat (B) for two stamps (S, S).

4. 1 toy (T) for two bats (B, B) and a stamp (S).

1. Initial state = (M, B, T)
2. Transformation rules:

a. If M then (C, P)
b. If M then (B, S)
c. If B then (S, S)
d. If T then (B, B, S)

3. The goal state is to left with only stamps (S, ... , S)



MBT
CPBT
BSBT
MSST
MBBBS
CPSST
SSSBT
CPSST
BSSST
CPBBBS
BSBBB
CPSSBBS
SSSSST
SSSSST
SSSBBBS
CPSSSSBS
SSSSSBBS
SSSSSBBS
CPSSSSSSS
SSSSSSSBS
SSSSSSSSSSS

Connected Components and Spanning trees

Graph

A graph can be defined as a group of vertices and edges to connect these vertices. The types of graphs are given as follows -

- **Undirected graph:** An undirected graph is a graph in which all the edges do not point to any particular direction, i.e., they are not unidirectional; they are bidirectional. It can also be defined as a graph with a set of V vertices and a set of E edges, each edge connecting two different vertices.
- **Connected graph:** A connected graph is a graph in which a path always exists from a vertex to any other vertex. A graph is connected if we can reach any vertex from any other vertex by following edges in either direction.
- **Directed graph:** Directed graphs are also known as digraphs. A graph is a directed graph (or digraph) if all the edges present between any vertices or nodes of the graph are directed or have a defined direction.

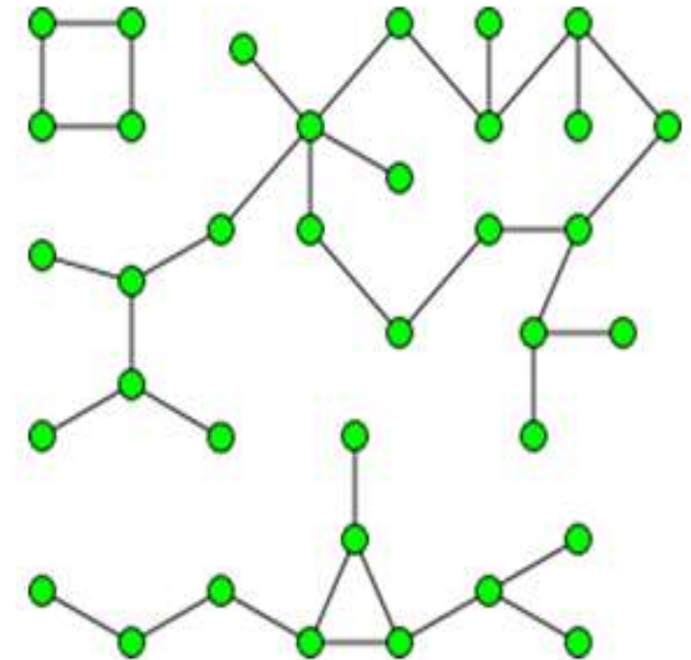
Connected components

In graph theory, a connected component (or just component) of an undirected graph is a subgraph in which any two vertices are connected to each other by paths.

It is connected to no additional vertices in the super graph.

For example,

- The graph shown in the illustration has three connected components.
- A vertex with no incident edges is itself a connected component.
- A graph that is itself connected has exactly one connected component, consisting of the whole graph.



A graph with three connected components.

What is a spanning tree?

Definition :

A spanning tree can be defined as the subgraph of an undirected connected graph.
It includes all the vertices along with the least possible number of edges.
If any vertex is missed, it is not a spanning tree.
A spanning tree is a subset of the graph that does not have cycles, and it also cannot be disconnected.
A spanning tree consists of (n-1) edges, where 'n' is the number of vertices (or nodes).

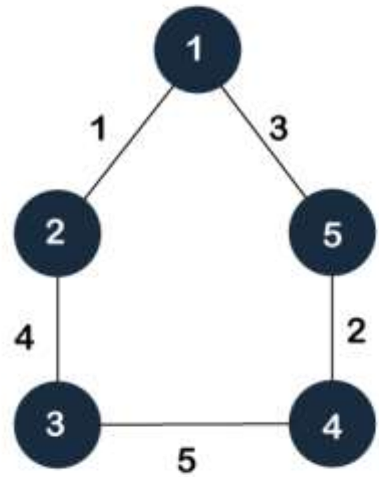
Edges of the spanning tree may or may not have weights assigned to them.
All the possible spanning trees created from the given graph G would have the same number of vertices, but the number of edges in the spanning tree would be equal to the number of vertices in the given graph minus 1.

Applications of the spanning tree

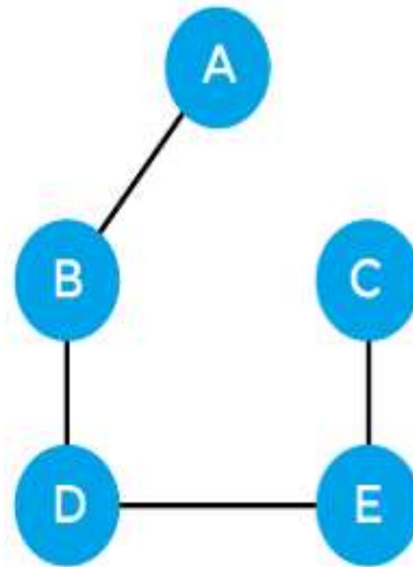
Basically, a spanning tree is used to find a minimum path to connect all nodes of the graph. Some of the common applications of the spanning tree are listed as follows -

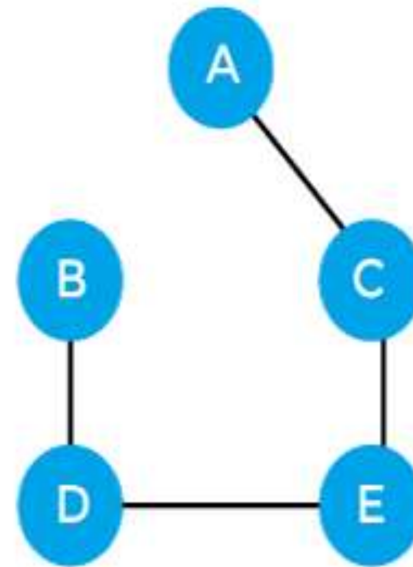
- Cluster Analysis
- Civil network planning
- Computer network routing protocol

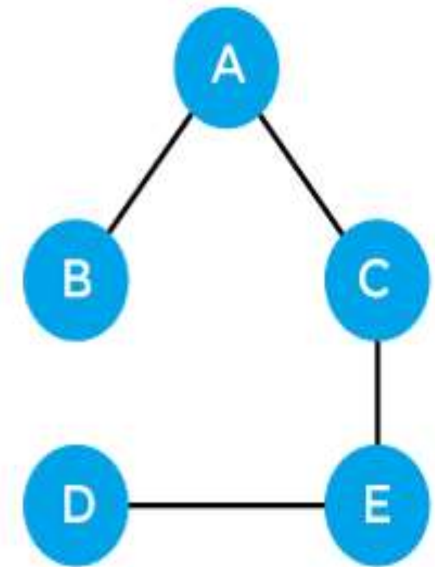
Graph Example



Spanning Tree







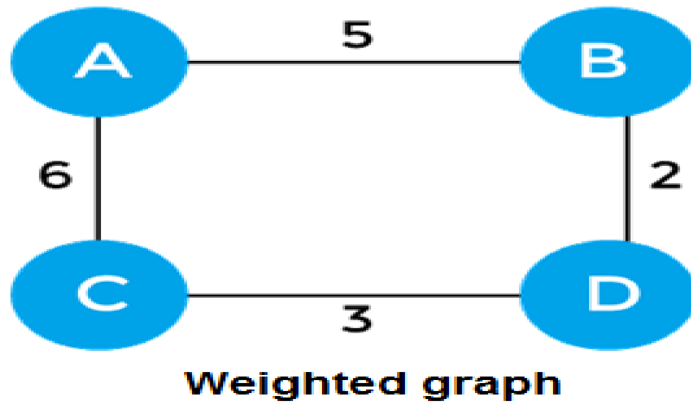
Properties of spanning-tree

- There can be more than one spanning tree of a connected graph G.
- A spanning tree does not have any cycles or loop.
- A spanning tree is **minimally connected,** so removing one edge from the tree will make the graph disconnected.
- A spanning tree is **maximally acyclic,** so adding one edge to the tree will create a loop.
- There can be a maximum n^{n-2} number of spanning trees that can be created from a complete graph.
- A spanning tree has **n-1** edges, where 'n' is the number of nodes.
- If the graph is a complete graph, then the spanning tree can be constructed by removing maximum (e-n+1) edges, where 'e' is the number of edges and 'n' is the number of vertices.

General Method of Minimum cost spanning tree.

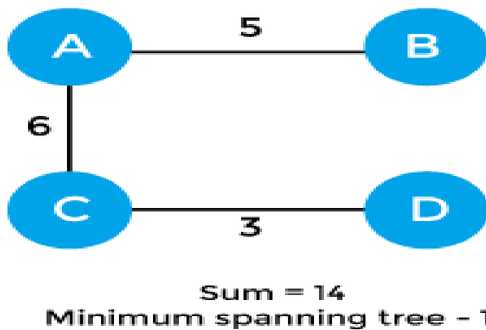
Example of minimum spanning tree

Let's understand the minimum spanning tree with the help of an example.

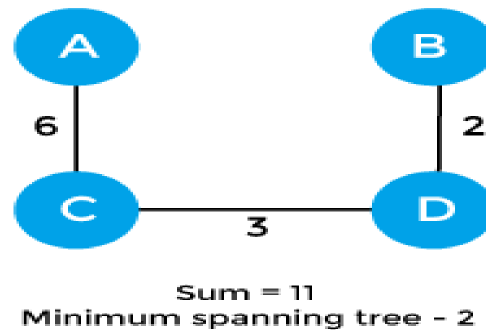


Weighted graph

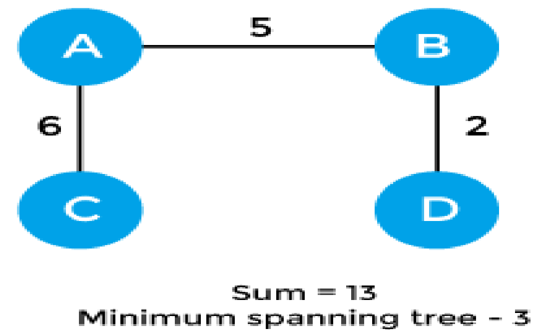
The sum of the edges of the above graph is 16. Now, some of the possible spanning trees created from the above graph are -



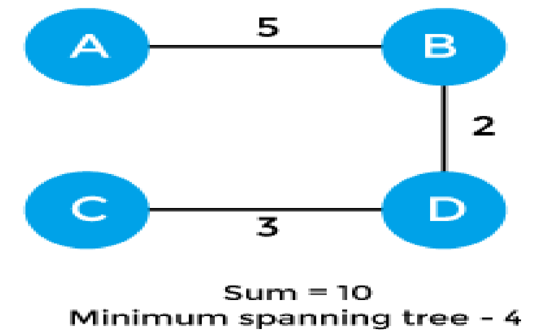
Sum = 14
Minimum spanning tree - 1



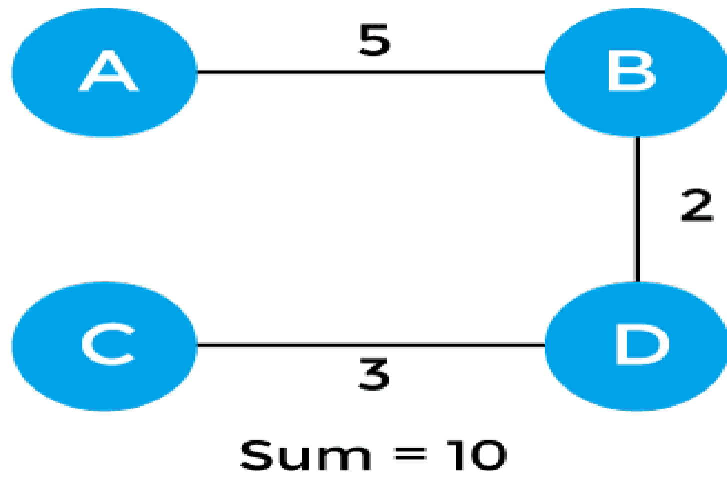
Sum = 11
Minimum spanning tree - 2



Sum = 13
Minimum spanning tree - 3



Sum = 10
Minimum spanning tree - 4



Applications of minimum spanning tree

The applications of the minimum spanning tree are given as follows -

- Minimum spanning tree can be used to design water-supply networks, telecommunication networks, and electrical grids
- It can be used to find paths in the map.

Algorithms for Minimum spanning tree

A minimum spanning tree can be found from a weighted graph by using the algorithms given below -

-
-

Prim's and Kruskal's Algorithms

Biconnected Components

Let G = (V, E) be a connected undirected graph.

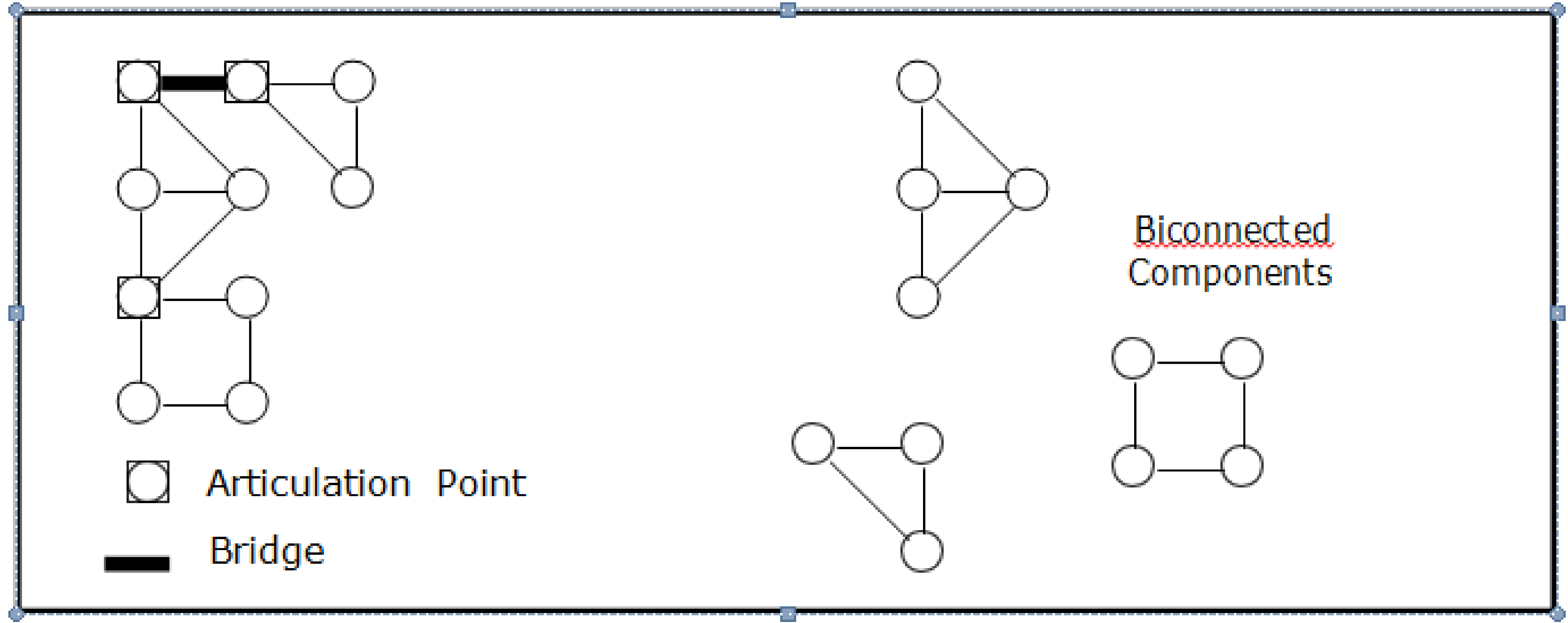
Definitions:

Articulation Point (or Cut Vertex): An articulation point in a connected graph is a vertex (together with the removal of any incident edges) that, if deleted, would break the graph into two or more pieces..

Bridge: Is an edge whose removal results in a disconnected graph.

Biconnected: A graph is biconnected if it contains no articulation points. In a biconnected graph, two distinct paths connect each pair of vertices. A graph that is not biconnected divides into biconnected components.

Biconnected Components



Articulation Points and Bridges

Biconnected Components

Let us consider the typical case of vertex v,

- v is not a leaf
- v is not the root.
- Let w1, w2, ... w_k be the children of v.
- For each child there is a subtree of the DFS tree rooted at this child.
- If for some child, there is no back edge going to a proper ancestor of v.
- if we remove v, this subtree becomes disconnected from the rest of the graph,
- Hence v is an articulation point.

L (u) = min {DFN (u), min {L (w) w is a child of u}, min {DFN(w) (u, w) is a back edge}}.

L (u) is the lowest depth first number

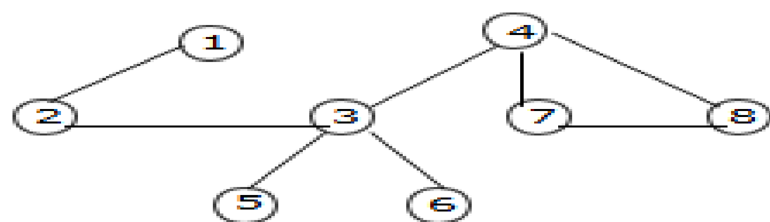
It can be reached from 'u' using a path of descendents followed by at most one back edge.

If 'u' is not the root then 'u' is an articulation point if 'u' has a child 'w' such that:

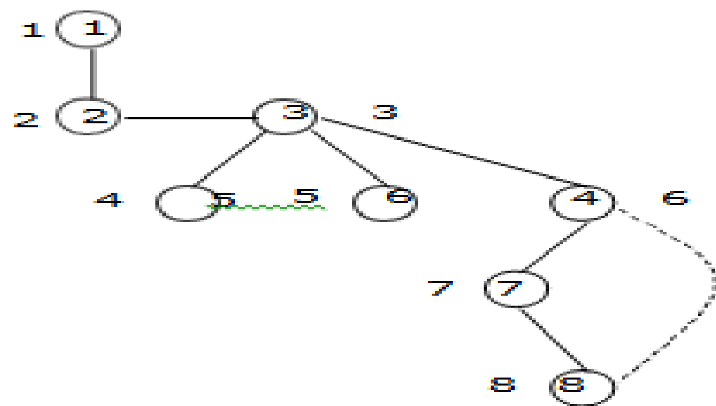
$$L(w) \geq DFN(u)$$

Example:

For the following graph identify the articulation points and Biconnected components:



Graph



DFS spanning Tree

L (u) = min {DFN (u), min {L (w) | w is a child of u}, min {DFN (w) | w is a vertex to which there is back edge from u}}

L (1) = min {DFN (1), min {L (2)}} = min {1, L (2)} = min {1, 2} = 1

L (2) = min {DFN (2), min {L (3)}} = min {2, L (3)} = min {2, 3} = 2

L (3) = min {DFN (3), min {L (4), L (5), L (6)}} = min {3, min {6, 4, 5}} = 3

L (4) = min {DFN (4), min {L (7)} = min {6, L (7)} = min {6, 6} = 6

L (5) = min {DFN (5)} = 4

L (6) = min {DFN (6)} = 5

L (7) = min {DFN (7), min {L (8)}} = min {7, 6} = 6

L (8) = min {DFN (8), min {DFN (4)}} = min {8, 6} = 6

Therefore, L (1: 8) = {1, 2, 3, 6, 4, 5, 6, 6}

Finding the Articulation Points:

Check for the condition if L (w) $\geq$ DFN (u) is true, where w is any

child of u. Vertex 1: Vertex 1 is not an articulation point.
It is a root node. Root is an articulation point if it has two or more child nodes.

Vertex 2: is an articulation point as L (3) = 3 and DFN (2) = 2.
So, the condition is true

Vertex 3: is an articulation Point as:

I. L (5) = 4 and DFN (3) = 3

II. L (6) = 5 and DFN (3) = 3 and
III. L (4) = 6 and

DFN (3) = 3 So, the

condition true in above

cases

Vertex 4: is an articulation point as L (7) = 6 and DFN (4) = 6.
So, the condition is true

Vertex 7: is not an articulation point as L (8) = 6 and DFN (7) = 7.
So, the condition is False

Vertex 5, Vertex 6 and Vertex 8 are leaf

nodes. Therefore, the articulation points

are {2, 3, 4}.

BACKTRACKING

Definition :

- Backtracking is a technique based on algorithm to solve problem.
- It uses recursive calling to find the solution by building a solution step by step increasing values with time.
- It removes the solutions that doesn't give rise to the solution of the problem based on the constraints given to solve the problem.

BACKTRACKING

use the backtracking algorithm:

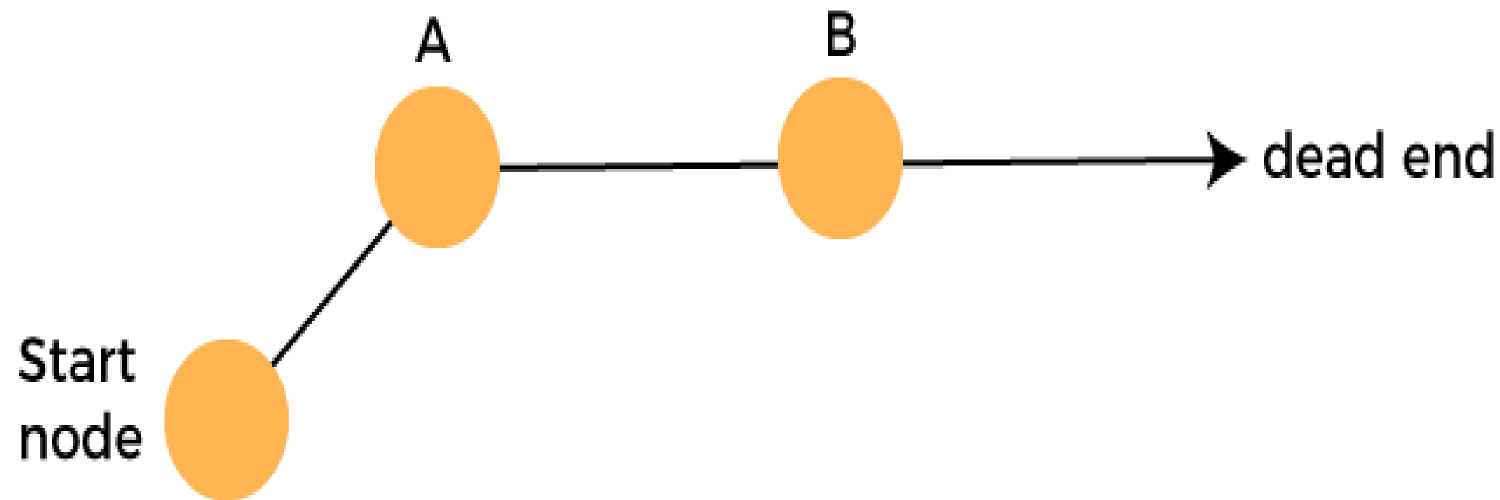
- A piece of sufficient information is not available to make the best choice, so we use the backtracking strategy to try out all the possible solutions.
- Each decision leads to a new set of choices.
- we backtrack to make new decisions. In this case, we need to use the backtracking strategy.

BACKTRACKING

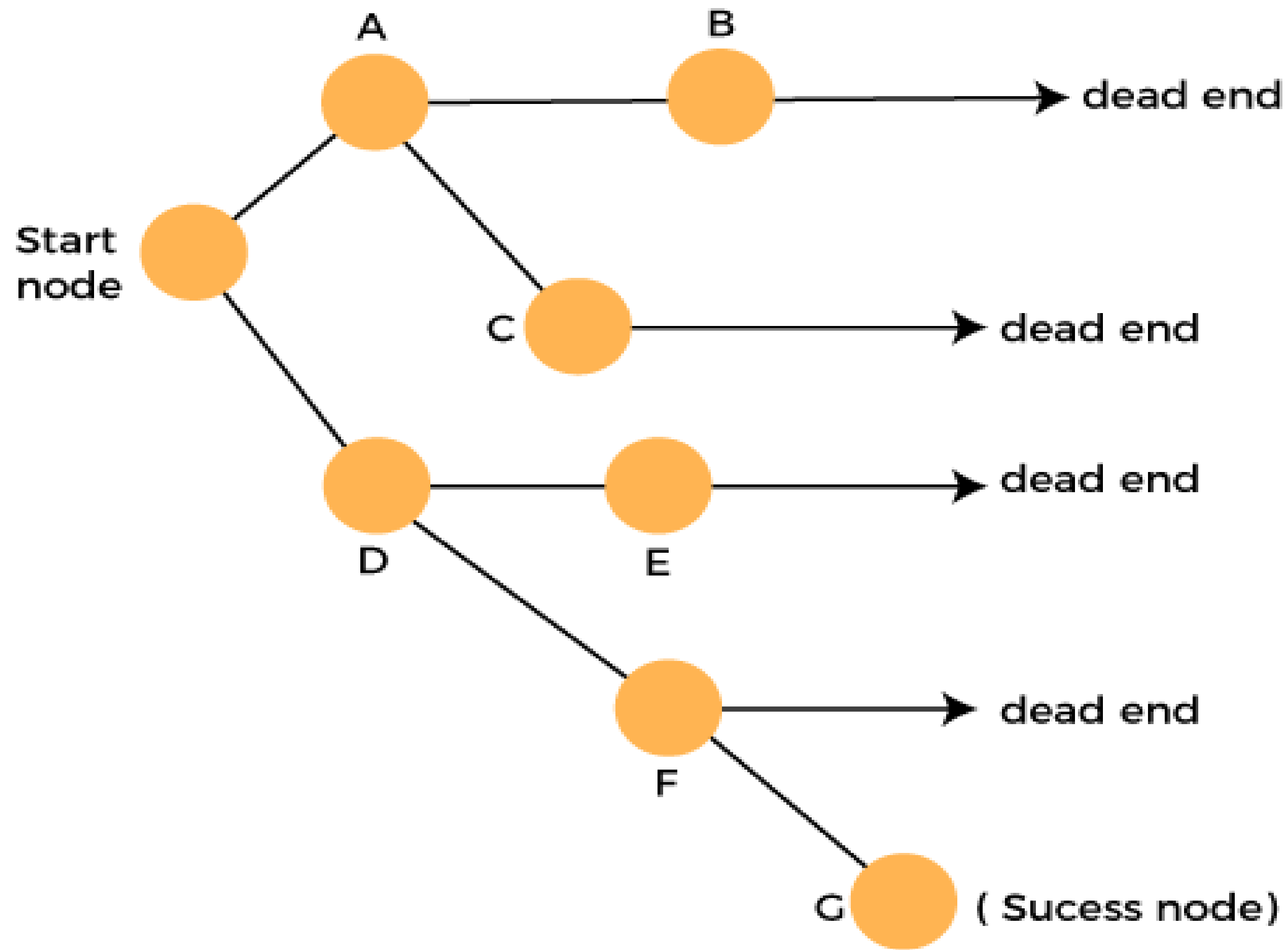
How does Backtracking work?

Backtracking is a systematic method of trying out various sequences of decisions until you find out that works. Let's understand through an example.

We start with a start node. First, we move to node A. Since it is not a feasible solution so we move to the next node, i.e., B. B is also not a feasible solution, and it is a dead-end so we backtrack from node B to node A.



BACKTRACKING



The terms related to the backtracking are:

- **Live node:** The nodes that can be further generated are known as live nodes.
- **E node:** The nodes whose children are being generated and become a success node.
- **Success node:** The node is said to be a success node if it provides a feasible solution.
- **Dead node:** The node which cannot be further generated and also does not provide a feasible solution is known as a dead node.

Many problems can be solved by backtracking strategy, and that problems satisfy complex set of constraints, and these constraints are of two types:

- **Implicit constraint:** It is a rule in which how each element in a tuple is related.
- **Explicit constraint:** The rules that restrict each element to be chosen from the given set.

Applications of Backtracking

- N-queen problem
- Sum of subset problem
- Graph coloring
- Hamiliton cycle

N-Queens Problem:

- A classic combinational problem is to place n queens on a n*n chess board so that no two attack, i.,e
- No two queens are on the same row, column or diagonal.
- If we take n=4then the problem is called 4 queens problem.
- If we take n=8 then the problem is called as 8 queens problem.

4-Queens problem:

- Consider a 4*4 chessboard. Let there are 4 queens. The objective is place the 4 queens on 4*4 chessboard in such a way that no two queens should be placed in the same row, same column or diagonal position.
- The explicit constraints are 4 queens are to be placed on 4*4 chessboards in 44 ways.
- The implicit constraints are no two queens are in the same row column or diagonal.
- Let{x1, x2, x3, x4} be the solution vector where x1 column on which the queen i is placed.
- First queen is placed in first row and first column.

1			

(a)

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

1			
•	•		

(b)

1			
•	•	2	
•	•	•	•

(c)

We are unable to place queen 3 in third row, so go back to queen 2 and place it somewhere else

1			
			2

(d)

1			
			2
	3		

(e)

Now the fourth queen should be placed in 4th row and 3rd column but there will be a diagonal attack from queen 3. So go back, remove queen 3 and place it in the next column. But it is not possible, so move back to queen 2 and remove it to next column but it is not possible. So go back to

	1		

	1		
			2

queen 1 and move it to next column.

(f) (g)

	1		
			2
3			

(h)

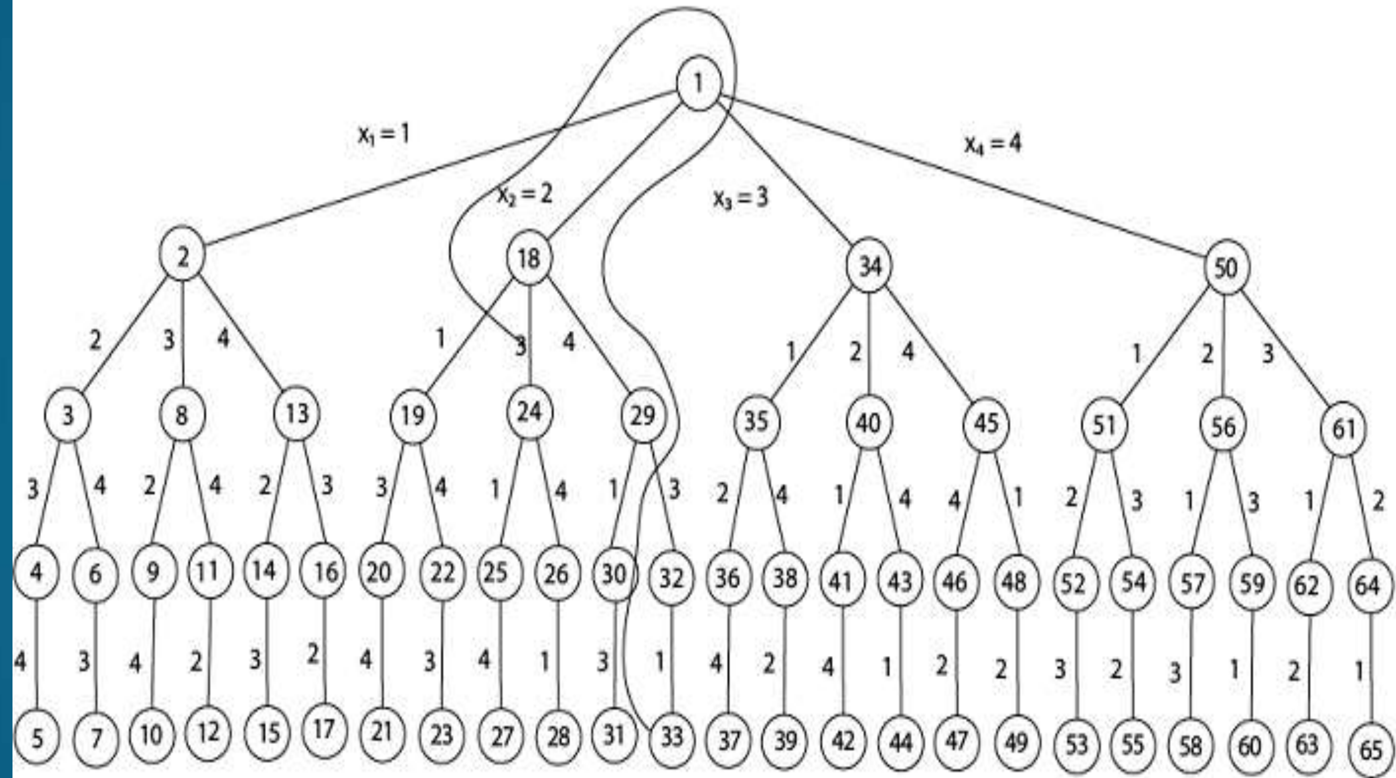
1			
			2
3			
		4	

(i)

Fig: Example of Backtrack solution to the 4-queens problem

Hence the solution of to 4-queens's problem is x1=2, x2=4, x3=1, x4=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.

Fig shows the complete state space for 4 - queens problem. But we can use backtracking method to generate the necessary node and stop if the next node violates the rule, i.e., if two queens are attacking.



4 - Queens solution space with nodes numbered in DFS

8-QUEENS PROBLEM

- A classic combinatorial problem is to place 8 queens on a 8*8 chess board so that no two

  attack, i.,e no two queens are to the same row, column or diagonal.
- Now, we will solve 8 queens problem by using similar procedure adapted for 4 queens problem.
- The algorithm of 8 queens problem can be obtained by placing n=8, in N queens algorithm.
- If two queens are placed at positions (i,j) and (k,l). They are on the same diagonal only if
- i-j=k-l ……………….(1)or
- i+j=k+l ....(2).
- From (1) and (2) implies j-l=i-k and j-l=k-i
- Two queens lie on the same diagonal iff |j-l|=|i-k|
- The solution of 8 queens problem can be obtained similar to the solution of 4 queens problem.

8-QUEENS PROBLEM

The solution can be shown as X1=3, X2=6, X3=2, X4=7, X5=1, X6=4, X7=8, X8=5

One possible solution for 8 queens problem is shown in fig:

	1	2	3	4	5	6	7	8
1				q_1				
2						q_2		
3								q_3
4		q_4						
5							q_5	
6	q_6							
7			q_7					
8					q_8			

SUM OF SUBSET PROBLEM

- Subset sum problem is the problem of finding a subset such that the sum of elements equal a given number. The backtracking approach generates all permutations in the worst case but in general, performs better than the recursive approach towards subset sum problem.
- A subset A of n positive integers and a value sum(d) is given, find whether or not there exists any subset of the given set, the sum of whose elements is equal to the given value of sum.

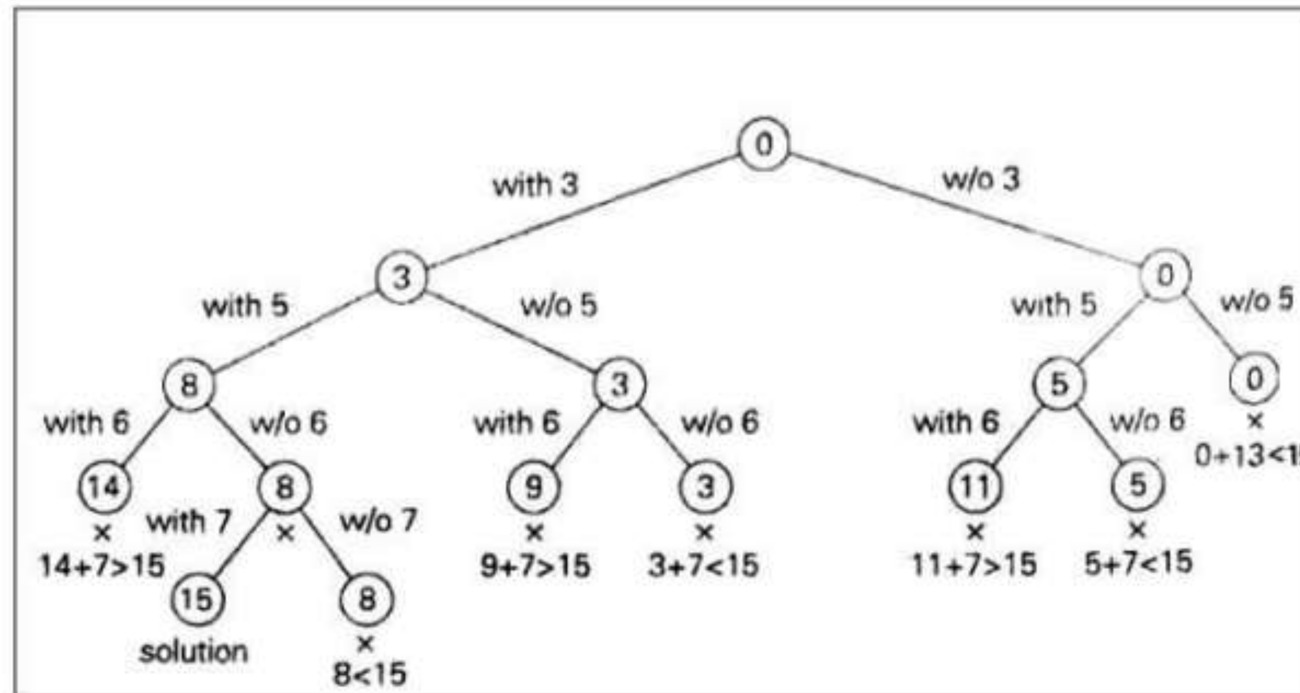
SUM OF SUBSET PROBLEM

ALGORITHM:

1. Start with an empty set
2. Add the next element from the list to the set
3. If the subset is having sum M, then stop with that subset as solution.
4. If the subset is not feasible or if we have reached the end of the set, then backtrack through the subset until we find the most suitable value.
5. If the subset is feasible (sum of subset < d) then go to step 2.
6. If we have visited all the elements without finding a suitable subset and if no backtracking is possible then stop without solution

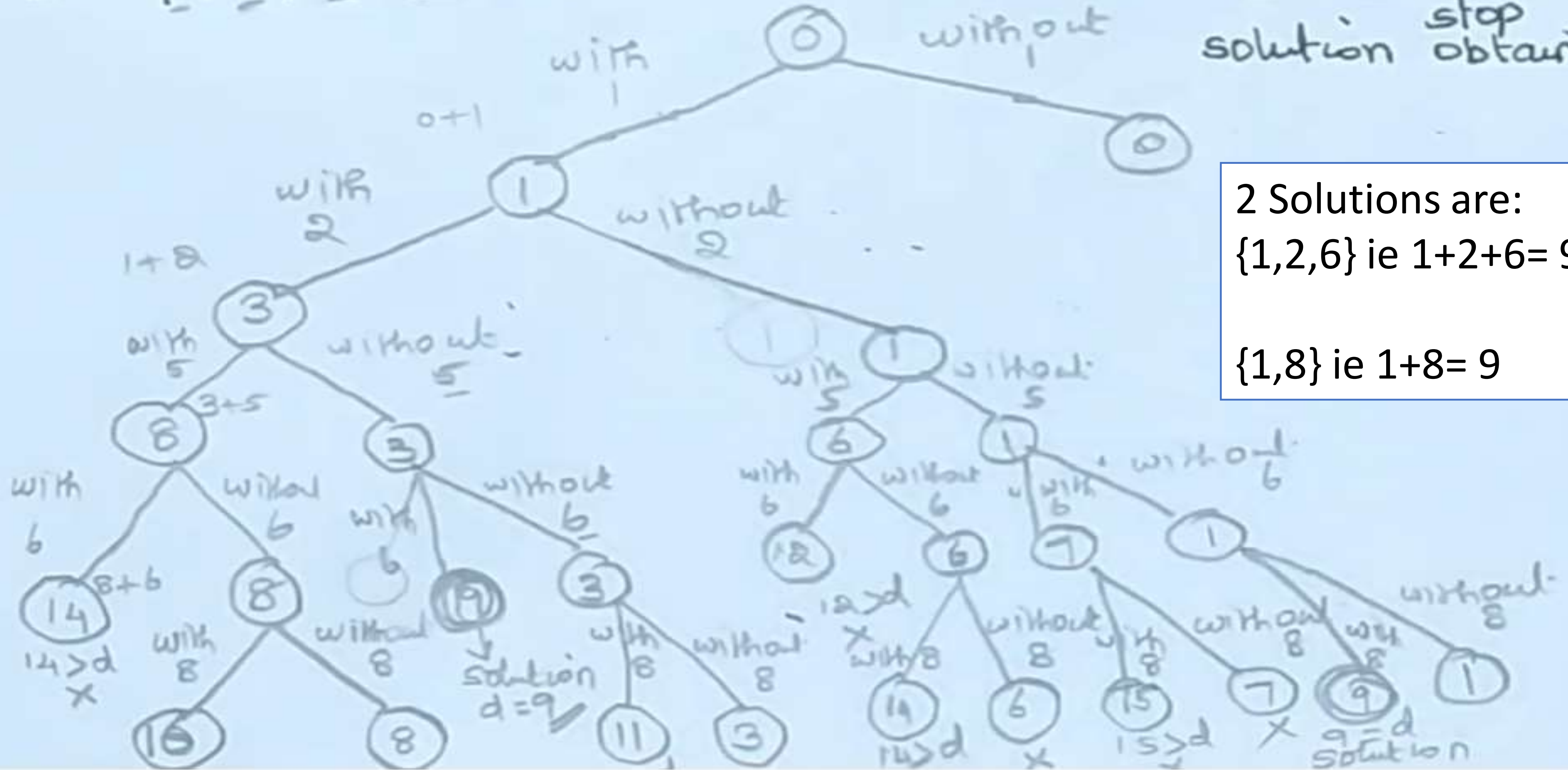
SUM OF SUBSET PROBLEM

Example: S = {3,5,6,7} and d = 15, Find the sum of subsets by using backtracking



Solution {3,5,7}

State Space Tree



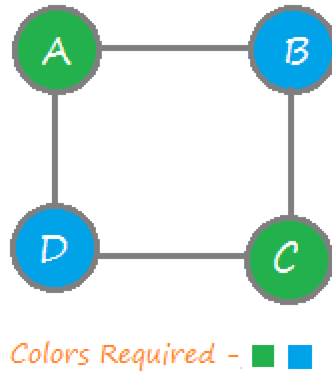
2 Solutions are:

{1,2,6} ie 1+2+6= 9

{1,8} ie 1+8= 9

What is Graph Coloring Problem?

We have been given a graph and we are asked to color all vertices with the 'M' number of given colors, in such a way that no two adjacent vertices should have the same color.



it is possible to color all the vertices with the given colors then we have to output the colored result, otherwise output 'no solution possible'.

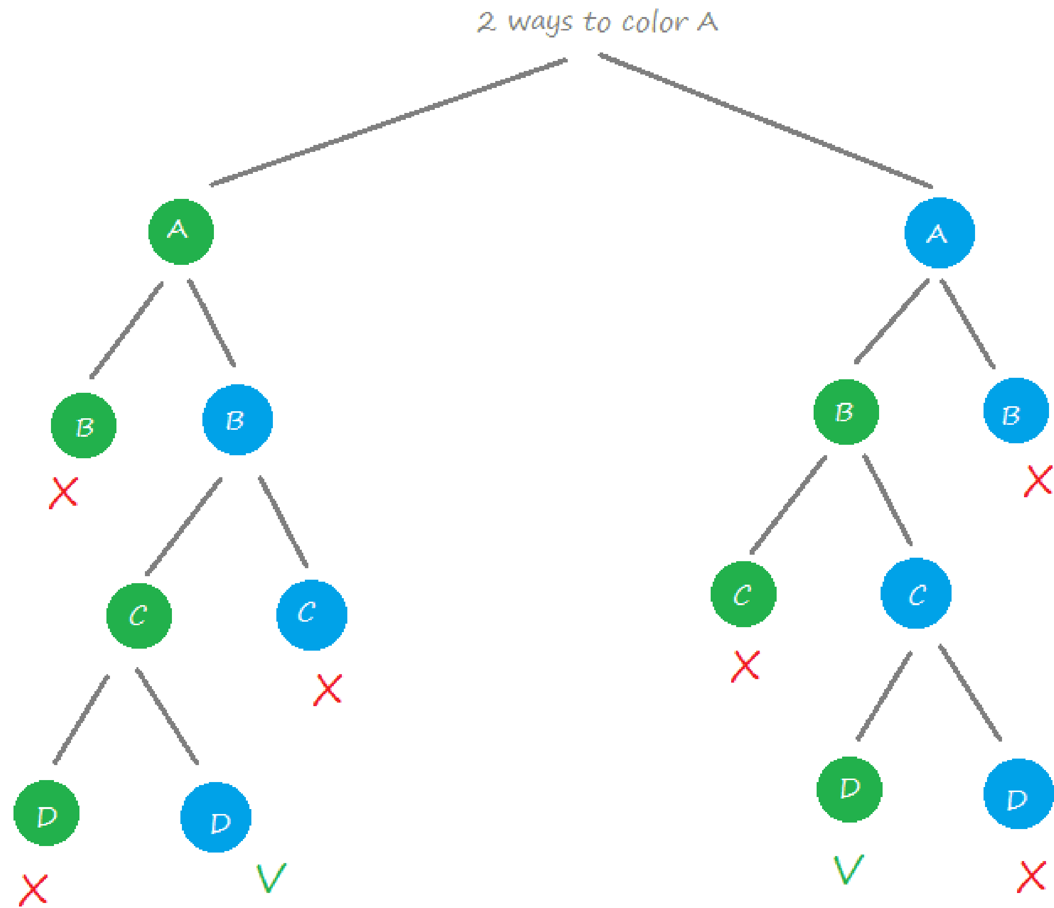
Graph Coloring by backtracking:

- In this approach, we color a single vertex and then move to its adjacent (connected) vertex to color it with different color.

- After coloring, we again move to another adjacent vertex that is uncolored and repeat the process until all vertices of the given graph are colored.

- In case, we find a vertex that has all adjacent vertices colored and no color is left to make it color different, we backtrack and change the color of the last colored vertices and again proceed further.

Graph Coloring by backtracking:



Graph Coloring by backtracking:

- **Steps To color graph using the Backtracking Algorithm:**
- Different colors:
  - Confirm whether it is valid to color the current vertex with the current color (by checking whether any of its adjacent vertices are colored with the same color).
  - If yes then color it and otherwise try a different color.
  - Check if all vertices are colored or not.
  - If not then move to the next adjacent uncolored vertex.
- If no other color is available then backtrack (i.e. un-color last colored vertex).
- *Here **backtracking means to stop further recursive calls** on adjacent vertices by returning false. In this algorithm Step-1.(Continue) and Step-2 (backtracking) is causing the program to try different color option.*
- ***Continue** – try a different color for current vertex.*
  ***Backtrack** – try a different color for last colored vertex.*

Graph Coloring by backtracking

Graph Coloring

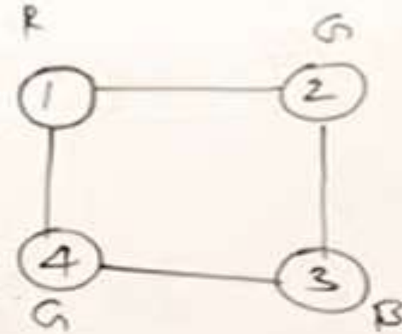
Example

$x_1 = R$

$x_2 = R$, G, B

$x_3 = R$, G, B

$x_4 = R$, G, B



$m = 3$

$\{R, G, B\}$

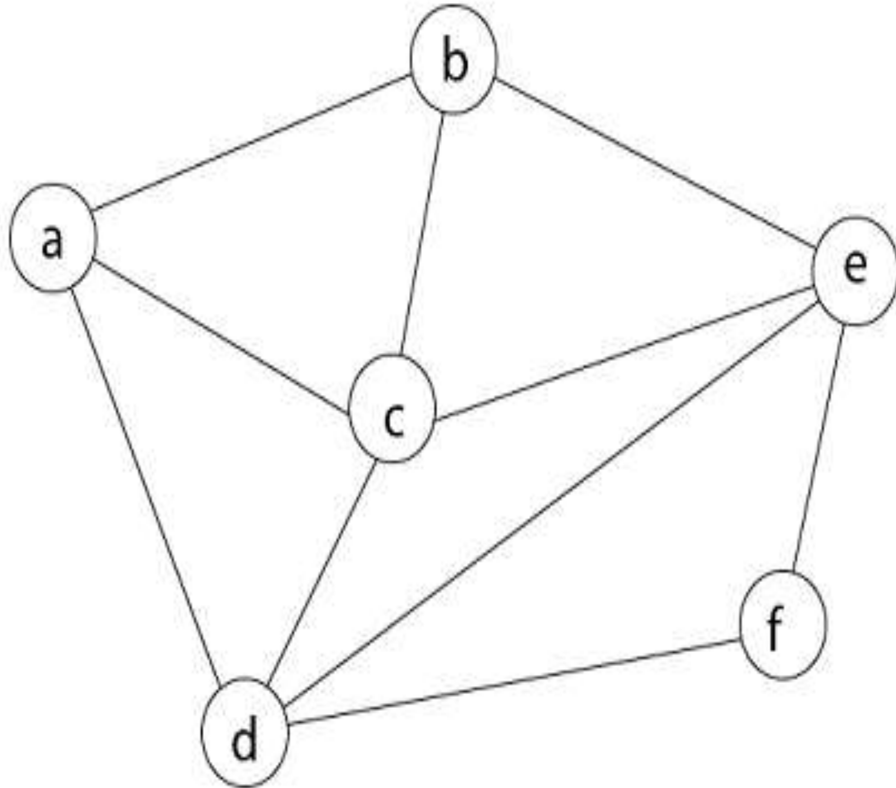
1) R, G, R, G
2) R, G, R, B
3) R, G, B, G

Hamiltonian Circuits Problem

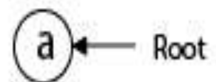
- A Hamiltonian circuit or tour of a graph is a path that starts at a given vertex, visits each vertex in the graph exactly once, and ends at the starting vertex.
- We use the Depth-First Search algorithm to traverse the graph until all the vertices have been visited.
- We traverse the graph starting from a vertex (arbitrary vertex chosen as starting vertex) and at any point during the traversal we get stuck (i.e., all the neighbor vertices have been visited), we backtrack to find other paths (i.e., to visit another unvisited vertex).
- If we successfully reach back to the starting vertex after visiting all the nodes, it means the graph has a Hamiltonian cycle otherwise not.
- We mark each vertex as visited so that we don’t traverse them more than once.

Hamiltonian Circuit Problems

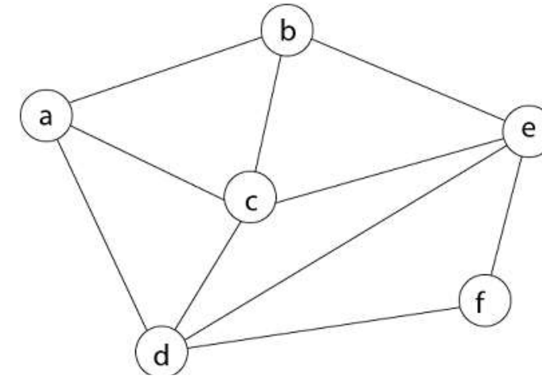
Example: Consider a graph G = (V, E) shown in fig. we have to find a Hamiltonian circuit using Backtracking method.

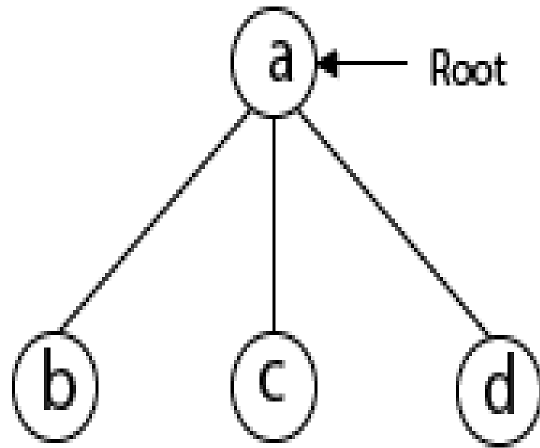


Solution: Firstly, we start our search with vertex 'a.' this vertex 'a' becomes the root of our implicit tree.

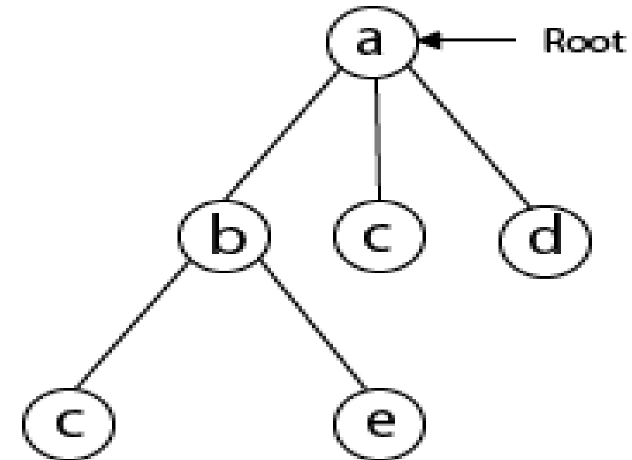


Hamiltonian Circuit Problems



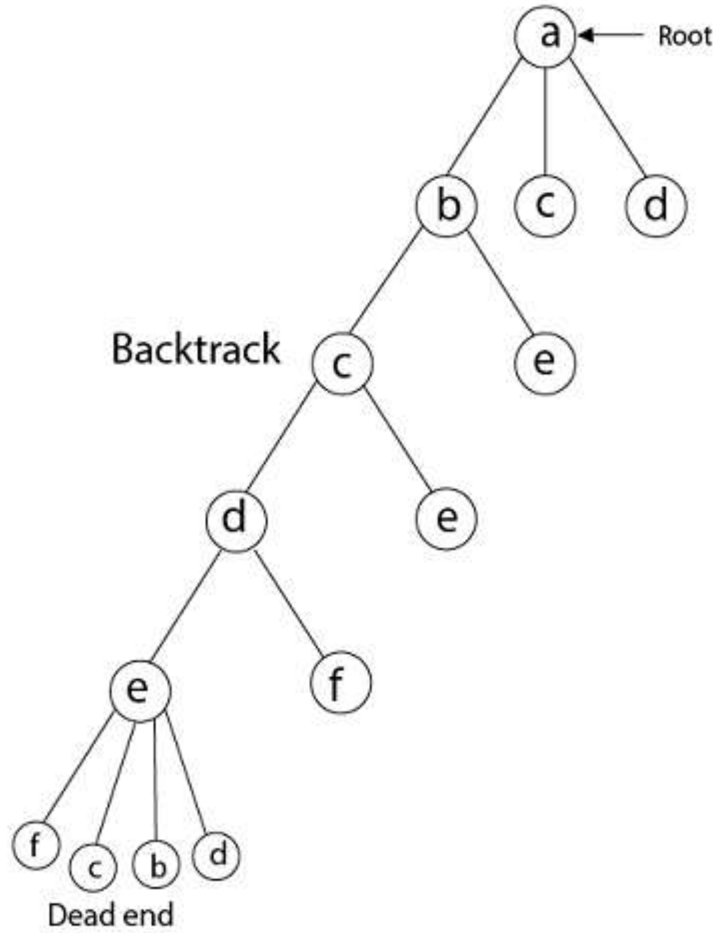


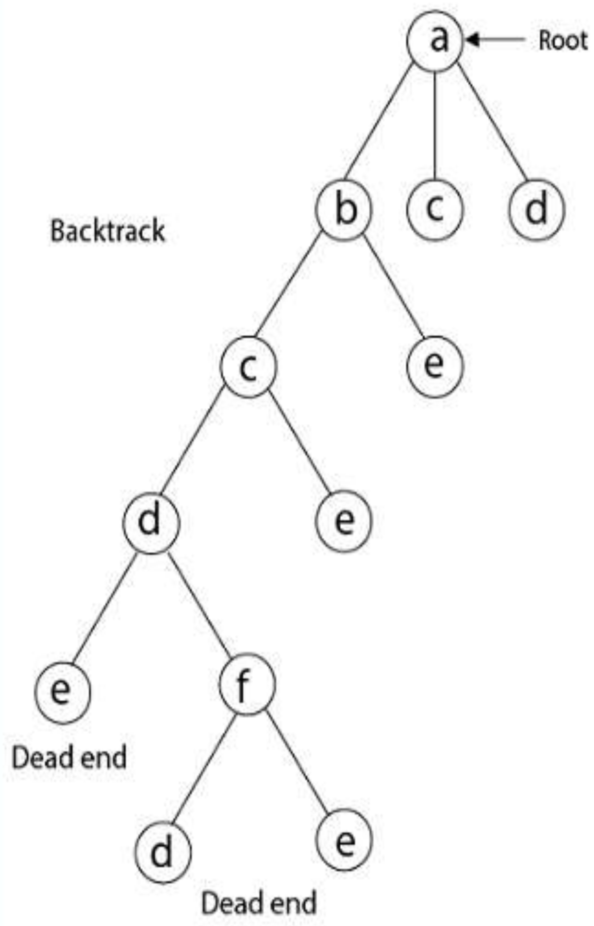
Next, we select 'c' adjacent to 'b.'

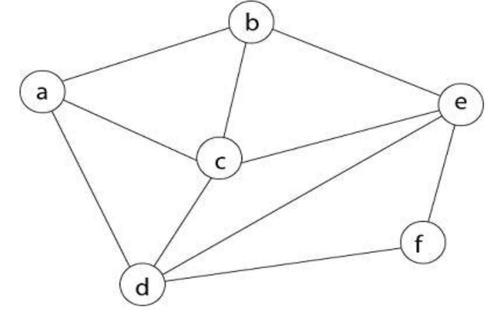


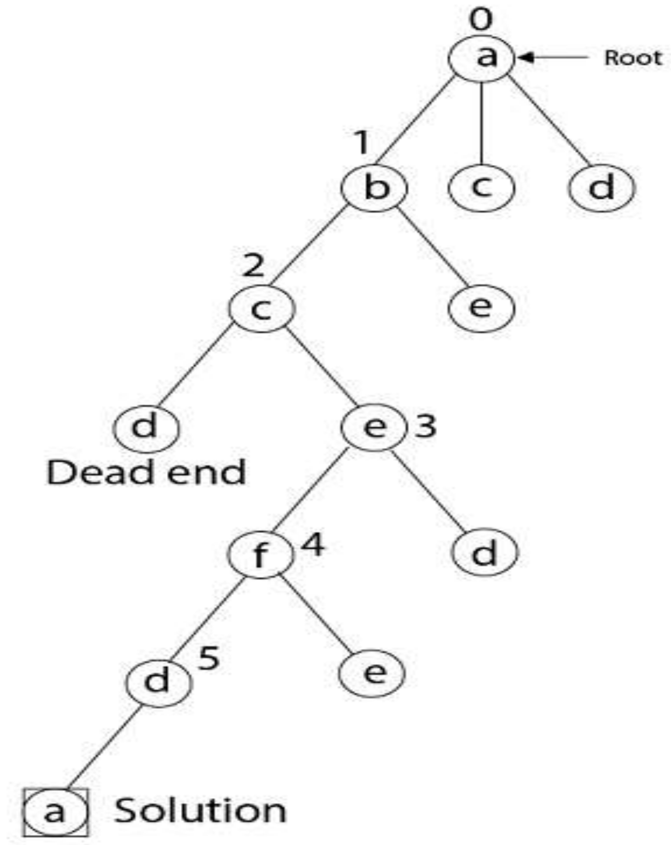
Next, we select 'd' adjacent to 'c.'

Hamiltonian Circuit Problems









Here we have generated one Hamiltonian circuit, but another Hamiltonian circuit can also be obtained by considering another vertex.

Thank you!