SRIMAAN

PG-TRB

COMPUTER

INSTRUCTOR GRADE-1

COMPUTER SCIENCE--UNIT-4-DATA STRUCTURES

TRB-ASSISTANT PROFESSOR-(ALL SUBJECT).

TNPSC-CTSE-(DEGREE & P.G DEGREE)-(ALL SUBJECT) FULL STUDY MATERIAL AVAILABLE.

PG-TRB STUDY MATERIALS:–TAMIL/ENGLISH/ MATHEMATICS/PHYSICS CHEMISTRY/COMMERCE (T/M & E/M)/BOTANY (T/M & E/M)/ ZOOLOGY HISTORY (T/E)/ECONOMICS (T/E)/ GEOGRAPHY /BIO-CHEMISTRY PGTRB-COMPUTER INSTRUCTOR GRADE-I -TO CONTACT -8072230063.

SRIMAAN COACHING CENTRE-TRICHY-PG-TRB-COMPUTER INSTRUCTOR GRADE-1–COMPUTER SCIENCE–UNIT-4-DATA STRUCTURES STUDY MATERIAL -TO CONTACT: 8072230063.

2024-25
SRIMAAN

PG-TRB: COMPUTER INSTRUCTOR GRADE-1 FULL STUDY MATERIAL WITH UNIT WISE QUESTION BANK AVAILABLE.

TNPSC-(CTSE)-COMBINED TECHNICAL SERVICES EXAMINATION- STUDY MATERIAL AVAILABLE.

TRB-POLYTECHNIC LECTURER MATERIALS:

MATHEMATICS / ENGLISH /PHYSICS / CHEMISTRY/COMPUTER SCIENCE/ IT / EEE /ECE /EIE/ICE/MECHANICAL/CIVIL/MOP AVAILABLE.

UG-TRB: ALL SUBJECT STUDY MATERIALS AVAILABLE.

SCERT/DIET/GTTI STUDY MATERIAL AVAILABLE.

DEO & BEO (T/M & E/M) STUDY MATERIALS AVAILABLE.

TN-MAWS-(DIPLOMA & DEGREE)- (ALL SUBJECT) MATERIAL AVAILABLE.

TNEB-(ASSESSOR/AE/JA) MATERIALS WITH QUESTION BANK AVAILABLE

TNTET /PG-TRB /TRB-POLYTECHNIC/DEO/BEO MATERIALS ARE SENDING THROUGH COURIER.

TO CONTACT

8072230063

PG-TRB STUDY MATERIALS:–TAMIL/ENGLISH/ MATHEMATICS/PHYSICS CHEMISTRY/COMMERCE (T/M & E/M)/BOTANY (T/M & E/M)/ ZOOLOGY HISTORY (T/E)/ECONOMICS (T/E)/ GEOGRAPHY /BIO-CHEMISTRY PGTRB-COMPUTER INSTRUCTOR GRADE-I -TO CONTACT -8072230063.

SRIMAAN COACHING CENTRE-TRICHY.

TO CONTACT:8072230063.

PG-TRB: COMPUTER INSTRUCTOR GRADE-1

COMPUTER SCIENCE

UNIT-4: DATA STRUCTURES

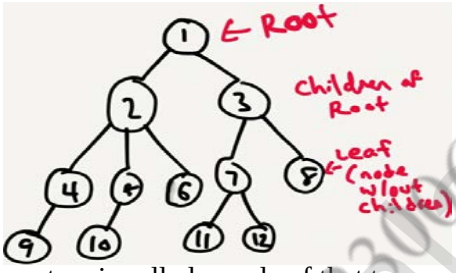
TREES & BALANCED BINARY TREES

TREES

Introduction about Trees:

A tree is a very flexible and powerful data structure that can be used for a wide variety of applications.

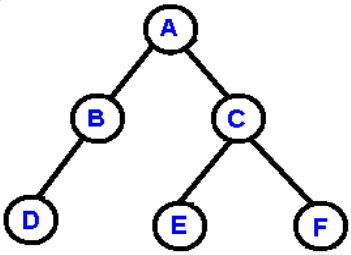
- Each tree node contains a name for data and one or more pointers to the other tree nodes.



Basic Terminology:

- ✓ **Node**: Each element presents in a binary tree is called a node of that tree.
- ✓ **Parent:** Parent of a node is the immediate predecessor of a node
- ✓ **Root:** The element represent the base node of the tree is called the root of the tree.
- ✓ **Child:** If the immediate predecessor of a node is the parent of the node then all immediate successors of a node are known as child.
- ✓ **Link:** This is a pointer to a node in a tree.
- ✓ **Left and Right sub trees:** Apart from the root, the other two sub sets of binary trees are binary trees. They are called the left and right sub trees of the original tree.
- ✓ **Leaf node**: A node that does not have any sons.
- ✓ **Degree:** The number of sub trees of a node.
- ✓ **Interior or non-interior nodes**: Nodes that have degree > 0.
- ✓ **Parent and child:** The roots of the sub trees of a node are called the children of that node.
- ✓ **Siblings:** Children's of the same parent are said to be siblings.
- ✓ **Ancestors**: The ancestors of anode are all the nodes along the path from the root of that node.
- ✓ **Level:** The level of anode is defined by initially letting the root be at level1. If a node is at level p, then its children are at level P+1.
- ✓ **Height or depth**: The height of a tree is defined to be the maximum level of any node in the tree.
- ✓ **Forest:** A forest is a set of n>=0 disjoint trees.
- ✓ **Path length of a node**: The number of edges needed to reach specified node form the root is called its path length.
- ✓ **Internal path**: The sum of path length of all the nodes in the tree.

Simple Binary tree



- A Binary Search Tree (BST) is a tree in which all the nodes follow the below-mentioned properties

1. The left sub-tree of a node has a key less than or equal to its parent node's key.
2. The right sub-tree of a node has a key greater than to its parent node's key.

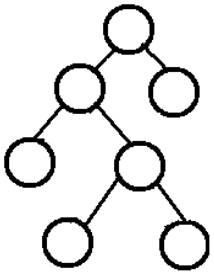
Definition and Concepts

Binary trees:

- A binary tree consists of a **finite set of elements** that can be partitioned into three distinct subsets called the **root, the left and right sub tree.**
- If there are no elements in the binary tree it is called an empty binary tree.

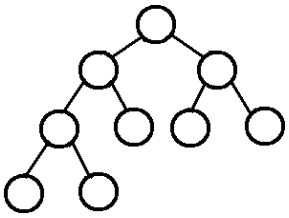
Full Binary Tree:

- All nodes have two children.
- Each sub-tree has same length of path.



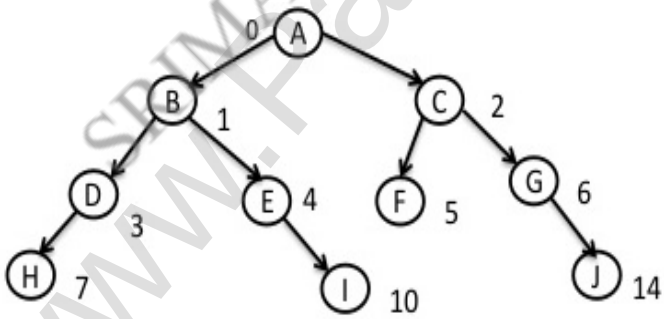
Complete binary tree:

- All nodes have two children.
- Each sub-tree has different length of path.



REPRESENTATION OF A BINARY TREE Array representation

- An array is used to store the nodes of the binary tree.
- Nodes stored in the array area access sequentially.
- Root is at index 0, and then left child and right child are stored.



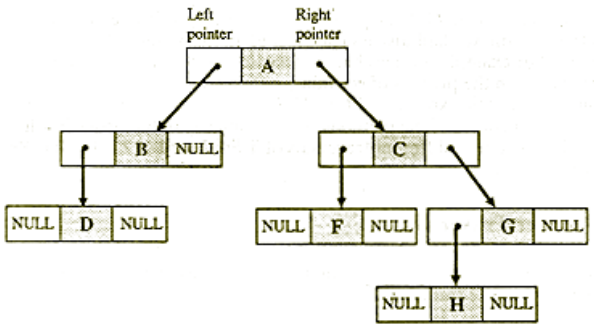
Declare array of size $2^{3+1}-1 = 15$

Char a[15]

A	B	C	D	E	F	G	H			I				J
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Linked representation of binary trees:

- In linked list every element is represented as nodes.
- A node consists of three fields such as,
  1. Left child
  2. Information node
  3. Right child



\

- The left child point to the left child of parent node.
- The right child point to the right child of parent node.
- The information holds the information of every node.

OPERATIONS ON A BINARY TREE

- ✓ **Insertion**: To include a node into an existing binary tree.
- ✓ **Deletion:** To delete a node from a non-empty binary tree.
- ✓ **Traversal:** To visit all the nodes in a binary tree.
- ✓ **Merge:** To merge two binary trees into a larger one.

Insertion Algorithm	**Deletion Algorithm**		
Algorithm for inserting a node:	**The node containing the data has no children**	**The node containing the data has one children**	**The node containing the data has two children**

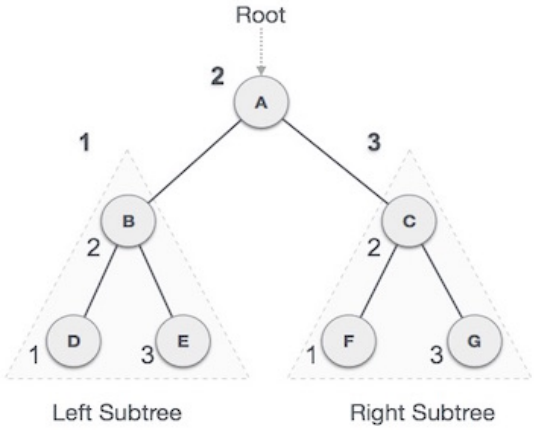
Struct node* curr;
Curr=root;
While(curr)
{
Parent=cur;
If(t→data > curr→data)
{
Curr=curr→right;
}
Else
{
Curr=curr→left;
}
}
If(t→data> parent→data)
{
Parent→right=t;
}
Else
{
Parent→left=t;
}
}
	If(curr==parent→left)
	{
	Parent→left=null;
	}
	
	Else
	{
	Parent→right=null;
	}
	Free(curr);
	}
		If(curr→left!=null)
		{
		If(curr==parent→left)
		{
		Parent→left=cur→left;
		}
		Curr→left=null;
		Free(curr);
		}
			If(curr→left!=null && cur→right!=null)
			{
			Int t;
			T=curr→right;
			If(t→right!=null && t→left==null)
			{
			Curr→data=t→data;
			Curr→right=t→right;
			T→right=null;
			Free(t);
			}

TRAVERSAL OF A BINARY TREE

- Traversal is a process to visit all the nodes of a tree and may print their values.
- That is, we cannot randomly access a node in a tree.
- There are three ways which we use to traverse a tree −
  1. In-order Traversal
  2. Pre-order Traversal
  3. Post-order Traversal

In-order Traversal

- In this traversal method, the left subtree is visited first, then the root and later the right sub-tree.
- We should always remember that every node may represent a subtree itself.
- If a binary tree is traversed in-order, the output will produce sorted key values in an ascending order.



D → B → E → A → F → C → G

- We start from A, and following in-order traversal, we move to its left subtree B.
- B is also traversed in-order. The process goes on until all the nodes are visited.
- The output of inorder traversal of this tree will be −

Algorithm

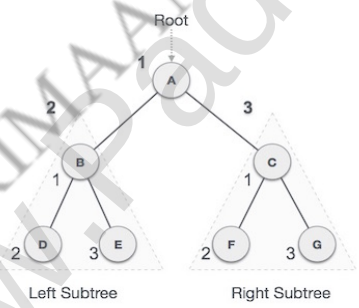
Until all nodes are traversed −

Step 1 − Recursively traverse left subtree. Step 2 − Visit root node.

Step 3 − Recursively traverse right subtree.

Pre-order Traversal

- In this the root node is visited first, then the left subtree and finally the right subtree.



A → B → D → E → C → F → G

- We start from A, and following pre-order traversal, we first visit A itself and then move to its left subtree B.
- B is also traversed pre-order. The process goes on until all the nodes are visited.
- The output of pre-order traversal of this tree will be −

Algorithm

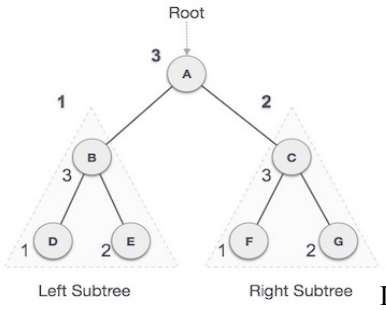
Until all nodes are traversed −

Step 1 − Visit root node.

Step 2 − Recursively traverse left subtree. Step 3 − Recursively traverse right subtree.

Post-order Traversal

In this traversal method, the root node is visited last, hence the name. First we traverse the left subtree, then the right subtree and finally the root node.



D → E → B → F → G → C → A

- We start from A, and following Post-order traversal, we first visit the left subtree B.
- B is also traversed post-order.
- The process goes on until all the nodes are visited.
- The output of post-order traversal of this tree will be −

Algorithm

Until all nodes are traversed −
Step 1 − Recursively traverse left subtree.
Step 2 − Recursively traverse right subtree.
Step 3 − Visit root node.

TYPES OF BINARY TREES

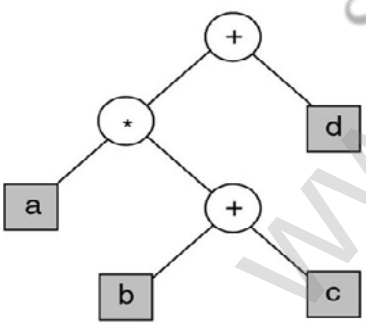
There are several types of binary trees

1. Expression tree
2. Binary search tree
3. Heap tree
4. Threaded binary tree
5. Huffman binary tree
6. Height balanced tree (AVL tree)
7. Decision tree

1. Expression tree:

- An expression tree is a binary tree which stores an arithmetic expression.
- The leaves of an expression tree are operand, such as constants or variable names and all internal nodes are the operators.
- Expression tree is always a binary tree because an arithmetic expression contains either binary operator or unary operator.

```
a * (b + c) + d
```



Operations on the expression tree:

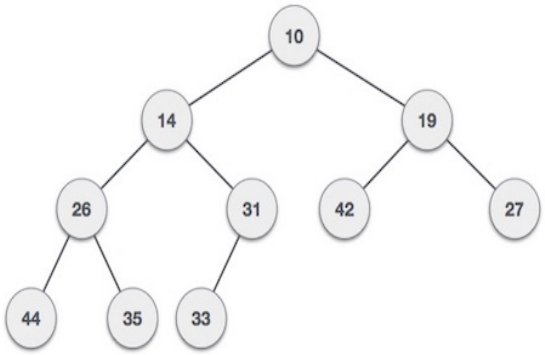
- There are two operations are possible on any expression tress:
  1. Traversing
  2. Evaluating
- Traversal operations are same as binary tree such as inorder, preorder and postorder.
- Evaluating of the expression is to evaluate the expression for which the tree is found.

3. HEAP TREES

- Heap is a binary tree such that the value at a node N is > or = the value at each of the children of node N.

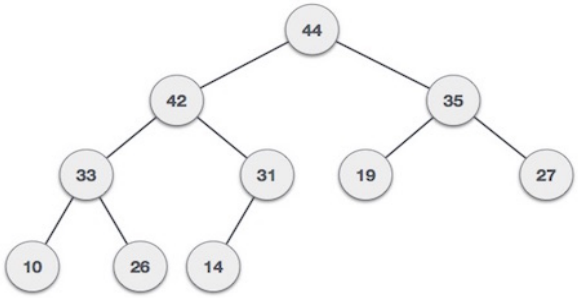
Min-Heap:

- Where the value of the root node is less than or equal to either of its children.



Max-Heap:

- Where the value of the root node is greater than or equal to either of its children.



Representation of a heap tree:

- A heap tree can be represented using linked structure.

Operations on heap tree:

1. *Insert into a heap tree*
2. *Deletion of a node*
3. *Merging two heap trees*

1. Insert into a heap tree

- This operation is used to insert a node into an existing heap tree satisfy the properties if heap tree.
  Step 1 − Create a new node at the end of heap.
  Step 2 − Assign new value to the node.
  Step 3 − Compare the value of this child node with its parent.
  Step 4 − If value of parent is less than child, then swap them.
  Step 5 − Repeat step 3 & 4 until Heap property holds.

2. Deletion of a node from a heap tree:

- Any node can be deleted from a heap tree. Deleting the root node has some special importance.
  Step 1 − Remove root node.
  Step 2 − Move the last element of last level to root.
  Step 3 − Compare the value of this child node with its parent.
  Step 4 − If value of parent is less than child, then swap them.
  Step 5 − Repeat step 3 & 4 until Heap property holds.

3. Merging two heap tree

The two heap trees H1 and H2. Merging the tree H2 with H1 means to include all the nodes from H2 to H1. H2 may be min heap or max heap and the resultant tree will be min heap if H1 is min heap else it will be max heap.

This operation consists of two steps:

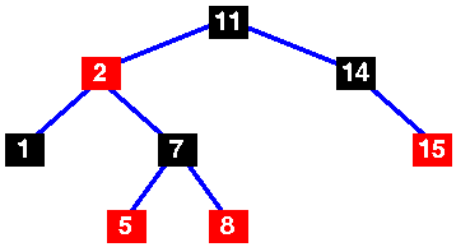
1. Delete the root node say x from H2.
2. Insert the node x into H1 satisfying the property of H1.

RED-BLACK TREE

A red-black tree is a binary search tree which has the following *red-black properties*:

1. Every node is either red or black.

2. Every leaf (NULL) is black.
3. If a node is red, then both its children are black.
4. Every simple path from a node to a descendant leaf contains the same number of black nodes.



GRAPHS

Introduction:

- Graph is another non-linear data structure.
- It is hierarchical relationship between parent and children's.

Application:

- Airlines
- Source-destination network
- Konigsberg's bridges.
- Flowchart of a program

Graph Terminology:

Graph:

- A graph consist of two sets
  (i) A set V called set of all vertices (or node)
  (ii) A set E called set of all edges (or arcs)m Set of vertices = { 1,2,3}
  Set of edges={ (1,2),(1,3)}

Digraph:

- A digraph is also called a directed graph. If a graph G, such that G=<V,E>, where V is the set of all vertices and E is the set of ordered pair of elements from V.
- Here G2 is a Digraph where
  V = {v1, v2, v3, v4}
  E = {(v1, v2), (v1, v3), (v2, v3), (v3, v4), (v4, v1)}

Weighted graph:

- A graph is termed as weighted graph if all the edges in it are labeled with some weight.
- Ex: G3 and G4 are two weighted graphs.

Adjacent vertices:

- A vertex v_i is adjacent to another vertex say v_j if there is an edge from v_i to v_j.
- Ex: Graph G11, v2 is adjacent to v3 and v4.

Self loop:

- If there is an edge whose starting and end vertices are same, that is (vi,vj) is an edge then it is called a self loop.
- Ex: GraphG5

Parallel edges:

- If there are more than one edges between the same pair of vertices, then they are known as the parallel edge.
- Ex: Graph G5.

Multi graph:

- A graph which has either self loop or parallel edges or both is called multi graph.
- Ex: Graph G5.

Simple graph (digraph):

- A graph if it does not have any self loop or parallel edges is called a simple graph.
- Ex: graph G5.

Complete graph:

- A graph G is said to be complete if each vertex vi adjacent to every other vertex vj in G
- Ex: Grapg G6 and G9.

Acyclic graph:

- If there is a path containing one or more edges which starts from a vertex vi and terminates into the same vertex then the path is known as a cycle.
- Ex: graph G4 and G7.

Isolated vertex:

- A vertex is isolated if there is no edge connected from any other vertex to the vertex.
- Ex: Graph G8.

Degree of vertex:

- The number of edges connected with vertex v_i called the degree of vertex v_i and is denoted by degree (v_i).
- In digraph there are two degrees: indegree and ourdegree.
- **Indegree** of v_i denoted as indegree(v_i) = number of edges incident into v_i.
- **Outdegree**(vi) = number of edges emanating from vi.
- Ex: In graph G4 indegree(v_1)=2 outdegree(v_1)=1 indegree(v_2)=2 outdegree(v_1)=0

Pendent vertex:

- A vertex v_i is pendent if its indegree(v_i)=1 and outdegree (v_i)=0
- Ex: G8 is a pendent vertex.

Connected graph:

- In a graph G two vertices v_I and v_j are said to be connected if there is a path in G from v_i to v_j.
- A graph is said to be connected if for every pair of distinct vertices v_i, v_j in G there is a path.
- Ex: Graph G1, G3 and G6.

REPRESENTATION OF GRAPHS

- A graph can be represented in many ways
  1. Set representation
  2. Linked representation
  3. Sequential (matrix) representation **(**

i) Set representation:

This is one of the straightforward methods of representing a graph. In this method two sets are maintained (i) V is the set of vertices

(ii) E is the set of edges.

Graph G1

V(G1)= { v1,v2,v3,v4,v5,v6,v7}

E(G1)= { (v1,v2), (v1,v3), (v2,v4), (v2,v5),(v3,v6),(v3,v7)}

Graph G2

V(G2)= { v1,v2,v3,v4,v5,v6,v7}

E(G2)= { (v1,v2), (v1,v3), (v2,v4), (v2,v5),(v3,v4),(v3,v6), (v4,v7),(v5,v7),(v6,v7)}

Graph G3

V(G3)= { A,B,C,D,E}

E(G3)= { (A,B), (A,C), (C,B), (C,A),(D,A),(D,B),(D,C),(D,E),(E,B)}

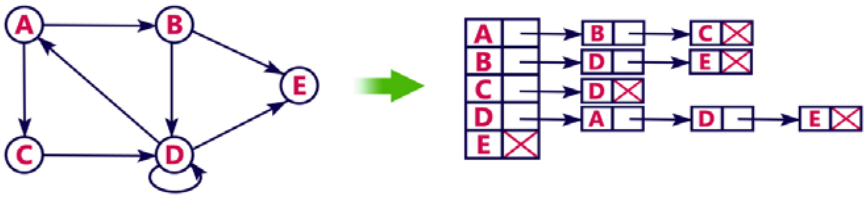
Graph G4

V(G4)= { A,B,C,D}

E(G4)= { (3,A,C), (5,B,A), (1,B,C), (7,B,D),(2,C,A),(4,C,D),(6,D,B),(8,D,C)}

(ii) Linked representation:

- Linked representation is another space-saving way of graph representation.
- Node structure is,

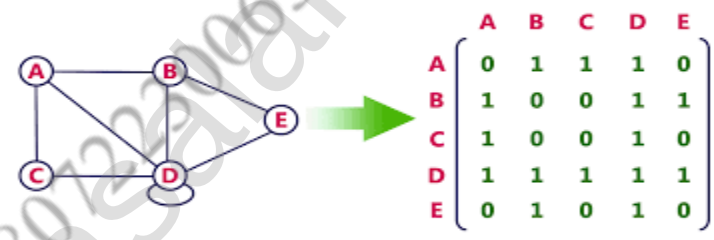


- Linked representation of graphs, the number of lists depends on the number of vertices in the graph.
- The header node in each list maintains a list of all adjacent vertices of anode for which header node is meant.

iii. Matrix Representation

- The adjacency matrix is represents in 2D array of size n*n matrix in which n is the number of vertices.

Ex: A[i][j]=1 means that the adjacency matrix has one edge. A[i][j]=0 means that the adjacency matrix has no edges.



Adjacency matrix for undirected graph:

- The adjacency matrix for an undirected graph is symmetric.
- The adjacency matrix for a directed graph need not be symmetric.
- The space needed to represent a graph using its adjacency matrix is n^2 locations.

OPERATIONS ON GRAPHS

Insertion

- To insert a vertex and hence establishing connectivity with other vertices in the existing graph
- To insert an edge between vertices in the graph.

Deletion

- To delete a vertex from the graph.
- To delete an edge from the graph.

Merging

- To merge two graph G1 and G2 into a single graph.

Traversal

- To visit all the vertices in the graph.

Operations on Linked List Representation of Graphs:

- In this representation using two representations.

(i) An array of vertices having two fields: LABEL – label for the vertices, LINK-the pointer to the linked list.

(ii) A linked list to maintain the list of all adjacent vertices for any vertex vi for which it is meant.

(iii) A node structure has two fields other than the filed LINK.

(iv) The first field WEIGHT is to store the weight of the edge and the second field LABLE to store the vertex's label

Insertion

- Insertion procedure differs for undirected graph and directed graph.
- To insert of a vertex into an undirected graph, if V_x is inserted and V_i be its adjacent vertex V_i has to be incorporated in the adjacency list of V_x as well as has to be incorporated in the adjacency list of V_i.
- If it's a digraph and if there is a path from V_x to V_i we add a node for V_i into the adjacency list of V_x, if there is an edge from V_i to V_x add a node for V_x in the adjacency list of V_i.

1) Algorithm INSERT_VERTEX_LL_UG(V_x, X)
Vx is the new vertex that has to be inserted into a graph.
Steps:

1. N=N+1, V_x=N
2. For i=1 to l do
    1. Let j=X[i]
    2. If (j>=N) then
        1. Print "No vertex labeled X[i] exist; edge from V_x into the list of vertices."
    3. Else
        1. INSERT_SL-END(UGptr[N],x[i])
        2. INSERT_SL-END(UGptr[j],Vx)
    4. Endif
3. Endfor
4. Stop

2) Algorithm INSERT_VERTEX_LL_DG(V_x, X,Y)
Vx is the new vertex that has to be inserted into a graph.
Steps:

1. N=N+1, V_x=N
2. For i=1 to m do
    1. Let j=X[i]
    2. If (j>=N) then
        1. Print "No vertex labeled X[i] exist; edge from V_x to X[i] is not established"
    3. Else
        1. INSERT_SL-END (DGptr[N],x[i])
    4. Endif
3. Endfor
4. For i=1 to n do
    1. Let j=Y[i]
    2. If (j>=N) then
        1. Print "No vertex labeled Y[i] exist; edge from V_x to X[i] is not established"
    3. Else
        1. INSERT_SL-END (DGptr[j],Vx)
    4. Endif
5. Endfor
6. stop

3) Insert the edge between two vertices in Undirected graph:
Algorithm: INSERT_EDGE_LL_UG(Vi,Vj)
Insert the edge to be inserted between vertices Vi and Vj.
Steps:

1. Let N=number of vertices in the graph
2. If(Vi>N) or (Vj>N) then
    1. Print" Edge is not possible between Vi and Vj"
3. Else
    1. INSERT_SL_END (UGptr[Vi],Vj"
    2. INSERT_SL_END (UGptr[Vj],Vi"
4. Endif
5. Stop

4)Insert the edge between two vertices in directed graph:

Algorithm: INSERT_EDGE_DG (Vi,Vj)

Insert the edge to be inserted between vertices Vi and Vj.

Steps:

1. Let N=number of vertices in the graph
2. If(Vi>N) or (Vj>N) then
    1. Print" Edge is not possible between Vi and Vj"
3. Else
    1. INSERT_SL_END (UGptr[Vi],Vj")
4. Endif
5. Stop

Deletion:

1) Delete the vertex:

Algorithm Delete_Vertex_LL_UG (Vx)

Step:

1. If (N=0) then
    1. Print" Graph is empty : No deletion"
    2. Exit
2. Endif
3. ptr=UGptr[Vx] . LINK
4. While(ptr ≠ NULL) do
    1. j=ptr.LABEL
    2. DELETE_SL_ANY(UGptr[j],Vx)
    3. DELETE_SL_ANY(UGptr[Vx], j)
    4. ptr=UGptr[Vx].LINK
5. Endwhile
6. UGptr[Vx].LABEL=NULL
7. UGptr[Vx].LINK=NULL
8. RETURN_NODE(ptr)
9. N=N-1
10. Stop

2) To delete the edge between vertices Vi and Vj

Algorithm: Delete_Edge_LL_UG(Vi,Vj)

1. let N=number of vertices in the graph
2. if(Vi>N) or (Vj >N) then
    1. Print "Vertex does not exist: Error in edge removal"
3. Else
    1. DELETE_SL_ANY(UGptr[Vi],Vj)
    2. DELETE_SL_ANY(UGptr[Vj],Vi)
4. Endif
5. Stop

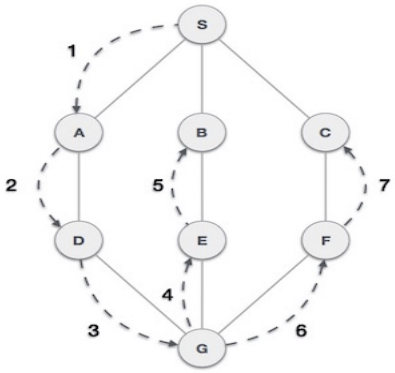
GRAPH TRAVESAL

In the traversal of a binary tree there are two ways as follows.

- **Depth First search**
- **Breadth first search**

Depth First Search:

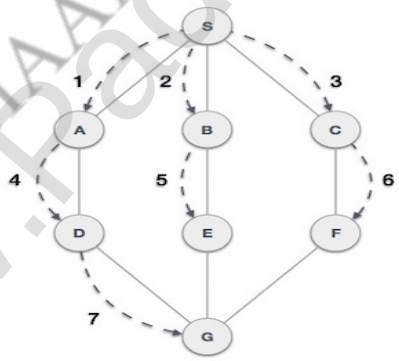
- Depth First Search (DFS) algorithm traverses a graph in a depthward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.



- As in the example given above, DFS algorithm traverses from S to A to D to G to E to B first, then to F and lastly to C.
- It employs the following rules.
  1. Rule 1 − Visit the adjacent unvisited vertex.
  2. Mark it as visited.
  3. Display it.
  4. Push it in a stack.
  5. Rule 2 − If no adjacent vertex is found, pop up a vertex from the stack.
  6. It will pop up all the vertices from the stack, which do not have adjacent vertices.
  7. Rule 3 − Repeat Rule 1 and Rule 2 until the stack is empty.

Breadth First Search:

- Breadth First Search (BFS) algorithm traverses a graph in a breadthward motion and uses a queue to remember to get the next vertex to start a search, when a dead end occurs in any iteration.



- As in the example given above, BFS algorithm traverses from A to B to E to F first then to C and G lastly to D. It employs the following rules.

  1. Rule 1 − Visit the adjacent unvisited vertex.
  2. Mark it as visited.
  3. Display it.
  4. Insert it in a queue.
  5. Rule 2 − If no adjacent vertex is found, remove the first vertex from the queue.
  6. Rule 3 − Repeat Rule 1 and Rule 2 until the queue is empty.

APPLICATION OF GRAPH STRUCTURES

- Graph is an important data structure whose extensive applications are known in almost all application areas.
- Conversion of a particular problem into this general graph theoretic problem will rest on the reader.
  1. Shortest path problem
  2. Topological sorting of a graph
  3. Spanning trees

SHORTEST PATH PROBLEM

- The shortest path problem is about finding a path between 2 vertices in a graph such that the total sum of the edges weights is minimum.
- To find shortest problem, we have three algorithm,
  1. Floyd & Warshall's Algorithms
  2. Dijkstra's Algorithm

Warshall's Algorithms:

- This is a classical algorithm by which we can determine whether there is a path from any vertex Vi to another vertex Vj either directly or through one or more intermediate vertices.

Steps:

11. For i=0 to N do
    1. For j=1 to N do 1. P[i][j]= Gptr[i][j]
    2. Endfor
12. End for
13. For k=1 to N do
    1. For i=1 to N do
       1. For j=1 to N do
          1. P[i][j]= P[i][j] v (P[i][k] ^ P[k][j]) 2. Endfor
       2. Endfor
14. End for
15. Return(p)
16. Stop

Floyd's Algorithm:

- The basic structure of the Floyd's algorithm is same as Warshall's algorithm.

Steps:

1. For i=1 to N do
   1. For j=1 to N do 1. If (Gptr[i][i] =0) then 1. Q[i][j]=∞
      2. PATHS[i][j]=NULL 2. Else 1. Q[i][j]=Gptr[i][j]
   2. P=COMBINE (i,j) 3. PATHS[i][j]=P 3. Endif
   2. Endfor
2. Endfor
3. For k=1 to N do
   1. For i=1 to N do
      1. For j=1 to N do
      1. Q[i][j]= MIN(Q[i][j], Q[i][k]+ Q[k][j])

2. If (Q[i][k]+ Q[k][j]< Q[i][j]) then
    1. p1= PATHS[i][k]
    2. p2= PATHS[k][j]
    3. PATHS[i][j]=COMBINE(p1,p2)
3. Endif

2. Endfor

2. Endfor

4. End for
5. Return (Q, PATHS)
6. Stop

Dijkstra's Algorithm:

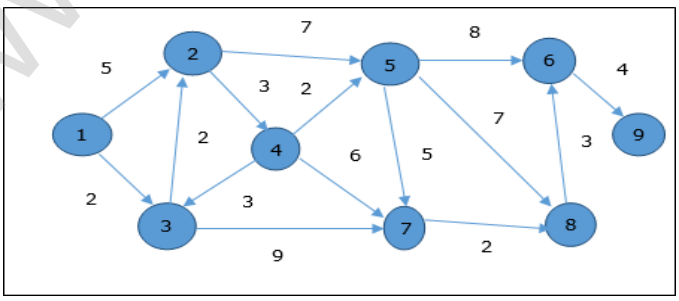
- Dijkstra's algorithm solves the single-source shortest-paths problem on a directed weighted graph G = (V, E), where all the edges are non-negative.

Example

- Let us consider vertex 1 and 9 as the start and destination vertex respectively.
- Initially, all the vertices except the start vertex are marked by ∞ and the start vertex is marked by 0.

Vertex	Initial	Step1 V1	Step2 V3	Step3 V2	Step4 V4	Step5 V5	Step6 V7	Step7 V8	Step8 V6
1	0	0	0	0	0	0	0	0	0
2	∞	5	4	4	4	4	4	4	4
3	∞	2	2	2	2	2	2	2	2
4	∞	∞	∞	7	7	7	7	7	7
5	∞	∞	∞	11	9	9	9	9	9
6	∞	∞	∞	∞	∞	17	17	16	16
7	∞	∞	11	11	11	11	11	11	11
8	∞	∞	∞	∞	∞	16	13	13	13
9	∞	∞	∞	∞	∞	∞	∞	∞	20

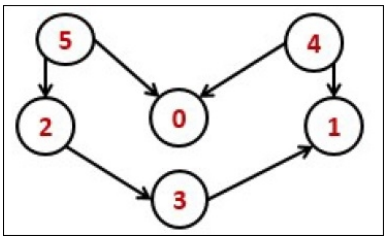
- Hence, the minimum distance of vertex 9 from vertex 1 is 20. And the path is 1→ 3→ 7→ 8→ 6→ 9



Topological Sorting:

- Topological sorting is an ordering of vertices of a graph, such that if there is a path from u to v in the graph then u appears before v in the ordering.
- A topological ordering is not possible if the graph has a cycle, since for two vertices u and v on the cycle, u precedes v and v precedes u.

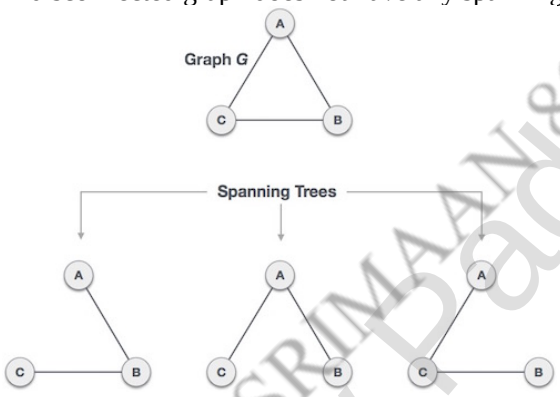
- A simple algorithm to find a topological ordering is to find out any vertex with in degree zero, that is vertex without any predecessor.



Algorithm:
Begin
  Initially mark all nodes as unvisited
  For all nodes v of the graph, do
    If v is not visited, then
      TopoSort (i, visited, stack)
  Done
  Pop and print all elements from the stack
End.

MINIMUM SPANNING TREE

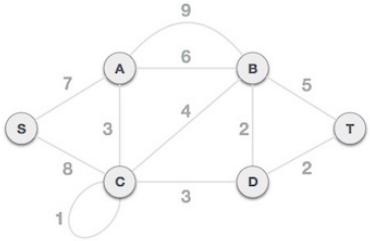
- A spanning tree is a subset of Graph G, which has all the vertices covered with minimum possible number of edges.
- Hence, a spanning tree does not have cycles and it cannot be disconnected.
- A disconnected graph does not have any spanning tree.



- There are two methods are efficient:
  1. Kruskal's algorithm
  2. Prim's Algorithm

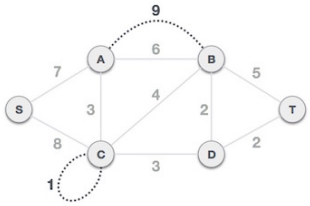
Kruskal's Algorithm:

- Kruskal's algorithm to find the minimum cost spanning tree uses the greedy approach.
- This algorithm treats the graph as a forest and every node it has as an individual tree.
- A tree connects to another only and only if, it has the least cost among all available options and does not violate MST properties.
- To understand Kruskal's algorithm let us consider the following example −



Step 1 - Remove all loops and Parallel Edges

- Remove all loops and parallel edges from the given graph.



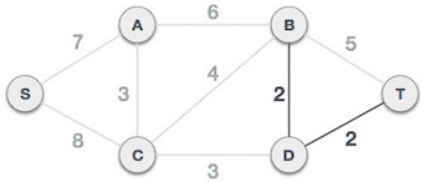
Step 2 - Arrange all edges in their increasing order of weight

- The next step is to create a set of edges and weight, and arrange them in an ascending order of weightage (cost).

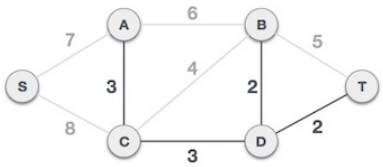
B, D	D, T	A, C	C, D	C, B	B, T	A, B	S, A	S, C
2	2	3	3	4	5	6	7	8

Step 3 - Add the edge which has the least weightage

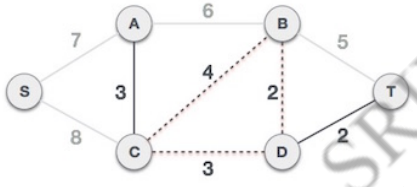
- Now we start adding edges to the graph beginning from the one which has the least weight.
- In case, by adding one edge, the spanning tree property does not hold then we shall consider not including the edge in the graph.



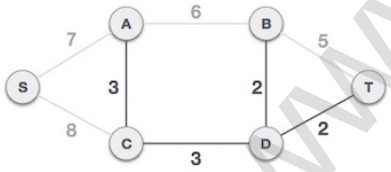
- The least cost is 2 and edges involved are B, D and D,T.
- We add them. Adding them does not violate spanning tree properties, so we continue to our next edge selection.
- Next cost is 3, and associated edges are A,C and C,D. We add them again −



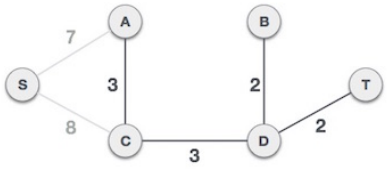
- Next cost in the table is 4, and we observe that adding it will create a circuit in the graph. −



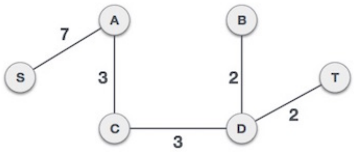
- We ignore it. In the process we shall ignore/avoid all edges that create a circuit.



- We observe that edges with cost 5 and 6 also create circuits.
- We ignore them and move on.



- Now we are left with only one node to be added.
- Between the two least cost edges available 7 and 8, we shall add the edge with cost 7.

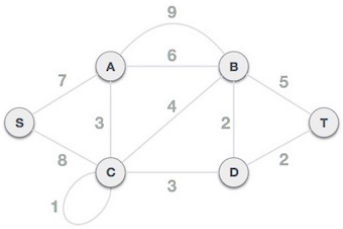


- By adding edge S, A we have included all the nodes of the graph and we now have minimum cost spanning tree.

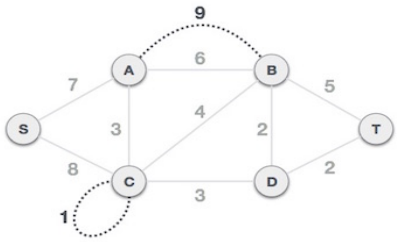
Prim's Algorithm:

- Prim's algorithm to find minimum cost spanning tree.
- Prim's algorithm, treats the nodes as a single tree and keeps on adding new nodes to the spanning tree from the given graph.

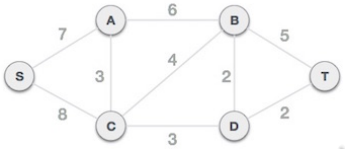
Example −



Step 1 - Remove all loops and parallel edges

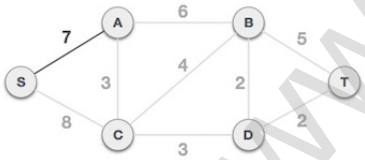


- Remove all loops and parallel edges from the given graph.
- In case of parallel edges, keep the one which has the least cost associated and remove all others.

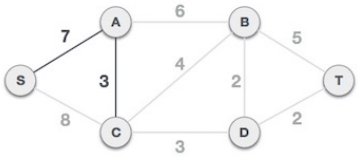


Step 3 - Check outgoing edges and select the one with less cost

- After choosing the root node S, we see that S, A and S,C are two edges with weight 7 and 8, respectively. We choose the edge S, A as it is lesser than the other.



- Now, the tree S-7-A is treated as one node and we check for all edges going out from it
- We select the one which has the lowest cost and include it in the tree.



- After this step, S-7-A-3-C tree is formed. Now we'll again treat it as a node and will check all the edges again.
- However, we will choose only the least cost edge.
- In this case, C-3-D is the new edge, which is less than other edges' cost 8, 6, 4, etc.

SEARCHING

What is searching?

- Searching is the process of finding a given value position in a list of values.
- It decides whether a search key is present in the data or not.
- It is the algorithmic process of finding a particular item in a collection of items.
- It can be done on internal data structure or on external data structure.
- There are two types of search,
  1. Linear search
  2. Non-linear search

Searching Techniques

- To search an element in a given array, it can be done in following ways.
  1. Sequential Search/Linear Search.
  2. Binary Search

1. Sequential Search/Linear Search

- Linear search is a very simple search algorithm.
- In this type of search, a sequential search is made over all items one by one.
- Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data collection.

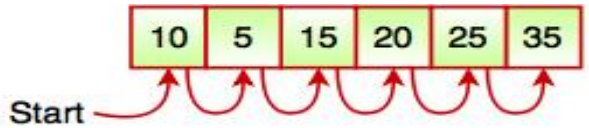


Fig. Sequential Search

i) Linear Search with Array:

- Start the search from the first element and continue till you get the element (or reach end of the array).

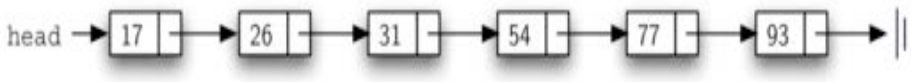
Element to Search: 5

ii) ***Linear Search with linked list****:*

- Our sequential search function for linked lists will take two arguments,
  1. A pointer to the first element in the list.
  2. The value for which we are searching.
- The function will return a pointer to the list structure containing the correct data, or will return NULL if the value wasn't found.

iii) Linear Search with ordered list:

- In order to implement the ordered list, we must remember that the relative positions of the items are based on some underlying characteristic.
- The ordered list of integers given above (17, 26, 31, 54, 77, and 93) can be represented by a linked structure as shown below.
- Again, the node and link structure is ideal for representing the relative positioning of the items.



2. Binary Search/Non Linear Search

- Binary Search is used for searching an element in a sorted array.
- It is a fast search algorithm with run-time complexity of O(log n).
- Binary search works on the principle of divide and conquer.
- This searching technique looks for a particular element by comparing the middle most element of the collection.
- It is useful when there are large numbers of elements in an array.

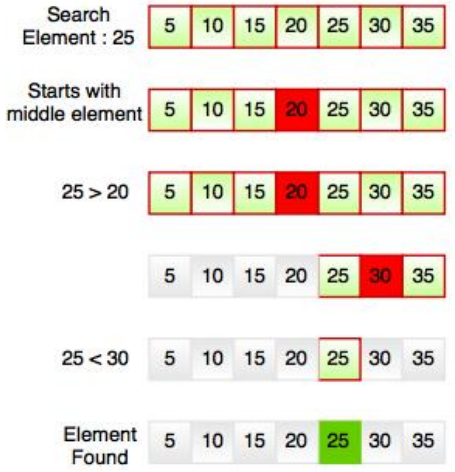


Fig. Working Structure of Binary Search

SORTING

What is sorting?

- Sorting refers to arranging data in a particular format. Sorting algorithm specifies the way to arrange data in a particular order.
- Most common orders are in numerical or lexicographical order.
- The importance of sorting lies in the fact that data searching can be optimized to a very high level, if data is stored in a sorted manner.
- Sorting is also used to represent data in more readable formats.
- Following are some of the examples of sorting in real-life scenarios –

TO BE CONTINUED....

SRIMAAN COACHING CENTRE-TRICHY- TET/PG-TRB / UG-TRB BEO/ DEO/TRB-POLY/ASST.PROF/TN-MAWS /TNEB /SCERT STUDY MATERIALS AVAILABLE- CONTACT:8072230063.

2024-25 SRIMAAN

TN-MAWS-MUNICIPAL ADMINISTRATION & WATER SUPPLY DEPARTMENT-(DEGREE & DIPLOMA) STUDY MATERIALS AVAILABLE.

TRB-ASSISTANT PROFESSORS IN GOVERNMENT ARTS AND SCIENCE COLLEGES & COLLEGES OF EDUCATION STUDY MATERIALS AVAILABLE.

UG-TRB MATERIALS

GRADUATE TEACHERS / BLOCK RESOURCE TEACHER EDUCATORS (BRTE) & SGT

- UG TRB: TAMIL MATERIAL WITH QUESTION BANK.
- UG TRB: ENGLISH STUDY MATERIAL +Q. BANK.
- UG-TRB: MATHEMATICS MATERIAL WITH Q. BANK (E/M)
- UG TRB: PHYSICS MATERIAL WITH QUESTION BANK (E/M)
- UG TRB: CHEMISTRY MATERIAL + QUESTION BANK (E/M)
- UG TRB: HISTORY MATERIAL + Q.BANK (E/M)
- UG TRB: ZOOLOGY MATERIAL + QUESTION BANK (E/M)
- UG TRB: BOTANY MATERIAL +QUESTION BANK (T/M& E/M)
- UG TRB: GEOGRAPHY STUDY MATERIAL (E/M)

SCERT/DIET/GTTI (LECTURER) STUDY MATERIAL AVAILABLE.

TNPSC-(CESE)-JSO STUDY MATERIAL AVAILABLE.

TANGEDCO (TNEB)-(T/M & E/M)

ASSESSOR/ASSISTANT ENGINEER (A.E)/JUNIOR ASSISTANT (ACCOUNTS)

SRIMAAN COACHING CENTRE-TRICHY- TET/PG-TRB / UG-TRB BEO/ DEO/TRB-POLY/ASST.PROF/TN-MAWS /TNEB /SCERT STUDY MATERIALS AVAILABLE- CONTACT:8072230063.

2024-25 SRIMAAN

PG-TRB MATERIALS

PG-TRB: COMPUTER INSTRUCTOR-GRADE-I (NEW SYLLABUS)-2024-2025 STUDY MATERIAL WITH Q.BANK AVAILABLE

- PG TRB: TAMIL STUDY MATERIAL +QUESTION BANK (T/M)
- PG TRB: ENGLISH MATERIAL WITH QUESTION BANK.
- PG-TRB: MATHEMATICS MATERIAL WITH Q.BANK (E/M)
- PG TRB: PHYSICS MATERIAL WITH QUESTION BANK (E/M)
- PG TRB: CHEMISTRY MATERIAL + QUESTION BANK (E/M)
- PG TRB: COMMERCE MATERIAL WITH Q.BANK (T/M)&(E/M)
- PG TRB:ECONOMICS MATERIAL+Q. BANK (T/M & E/M)
- PG TRB: HISTORY MATERIAL + Q. BANK (T/M & E/M)
- PG TRB: ZOOLOGY MATERIAL + QUESTION BANK (E/M)
- PG TRB: BOTANY MATERIAL +QUESTION BANK (T/M& E/M)
- PG TRB: GEOGRAPHY STUDY MATERIAL (E/M)

TNPSC-DEO (District Educational Officer(Group – I C Services) (TAMIL & ENGLISH MEDIUM) STUDY MATERIAL AVAILABLE.

TRB-BEO (Block Educational Officer) (TAMIL & ENGLISH MEDIUM) STUDY MATERIAL AVAILABLE.

TRB-POLYTECHNIC LECTURER-(NEW SYLLABUS) STUDY MATERIALS AVAILABLE

- MATHEMATICS STUDY MATERIAL with Question Bank.

SRIMAAN COACHING CENTRE-TRICHY- TET/PG-TRB / UG-TRB BEO/ DEO/TRB-POLY/ASST.PROF/TN-MAWS /TNEB /SCERT STUDY MATERIALS AVAILABLE- CONTACT:8072230063.

2024-25
SRIMAAN

- ENGLISH STUDY MATERIAL with Question Bank.
- PHYSICS STUDY MATERIAL with Question Bank.
- CHEMISTRY STUDY MATERIAL with Question Bank.
- MODERN OFFICE PRACTICE STUDY MATERIAL with Q.B.
- COMPUTER SCIENCE STUDY MATERIAL with Question Bank.
- INFORMATION TECHNOLOGY STUDY MATERIAL with Q.Bank.
- ECE STUDY MATERIAL with Question Bank.
- EEE STUDY MATERIAL With Question Bank.
- MECHANICAL STUDY MATERIAL With Question Bank.
- CIVIL STUDY MATERIAL With Question Bank.
- EIE STUDY MATERIAL with Question Bank.
- ICE STUDY MATERIAL with Question Bank.

TNPSC-CTSE (NON-INTERVIEW POST) STUDY MATERIAL AVAILABLE.

10% Discount for all materials. Materials are sending through COURIER.

TO CONTACT

8072230063

SRIMAAN