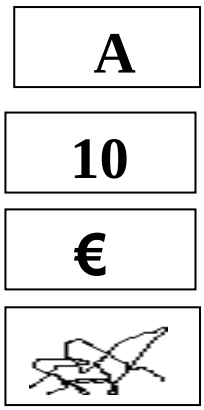
International Journal of Scientific Research in Science and Technology
Print ISSN: 2395-6011 | Online ISSN: 2395-602X (www.ijsrst.com)
doi : https://doi.org/10.32628/IJSRST

Trie State Tree in Data Structure

Amit Kumar Dinkar
Maharana Pratap College, Mohania, Kaimur, Bihar, India

I. INTRODUCTION

Data is a collection of facts figures or raw material of computers like.



A is an alphabet or facts figure as we know that. And specified by a group or teaches by teacher or any other person so we can say that it is a Data

10 is Number so it is also Data.

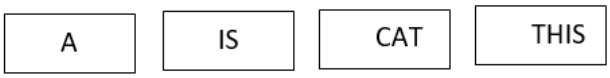
€ is a Euro symbol so it is also Data.

This Symbol is not specified or facts figure means what is this? So it is not a data in real life, but in computer science it can be a data because it is raw material of computer or it med from the group of bits and bytes.

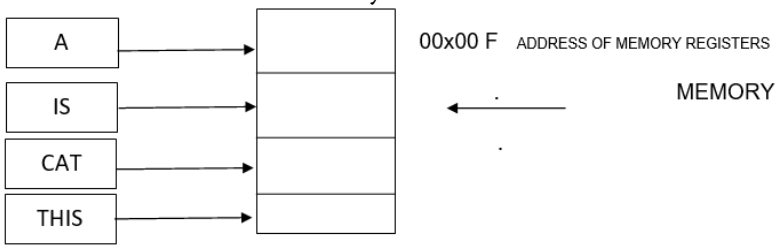
For example when we input only this symbol into the computer and that time we can say that it is a Data. Arranging of this data into the computer we study into the data structure.

Data Structure:- The arranging of data in the systematically manner is called data structure. In computer science these data are store into the computer's memory address. Memory address are auto generated space where data are exist, we can arrange them using logically, mathematically, by programming (like Pascal, C, C++) or help of Algorithm.

When we not arrange them in systematically manner then all information (that is a collection of meaningful data) are manning less for Example.



These data are store in memory like.



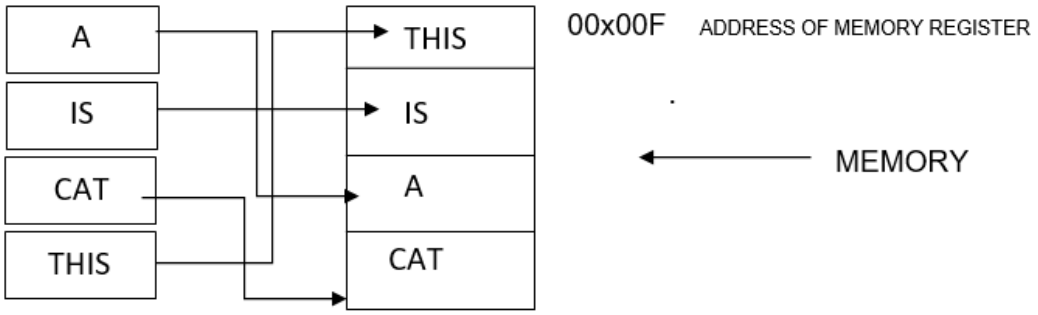
Article History:

Accepted: 05 April 2023
Published: 28 April 2023

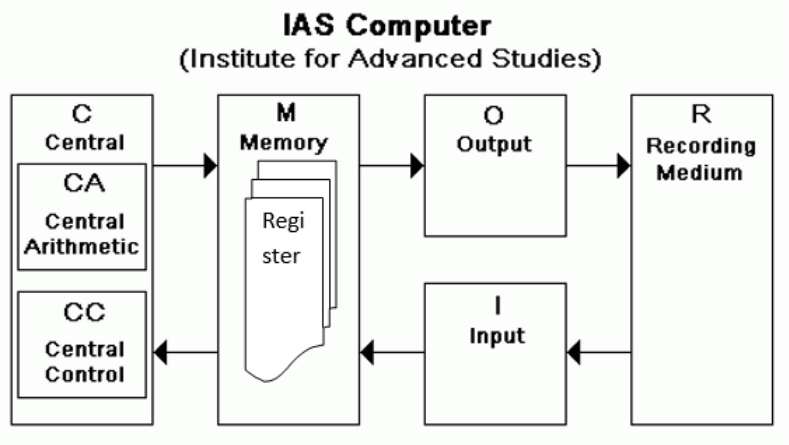
Publication Issue
Volume 10, Issue 2
March-April-2023

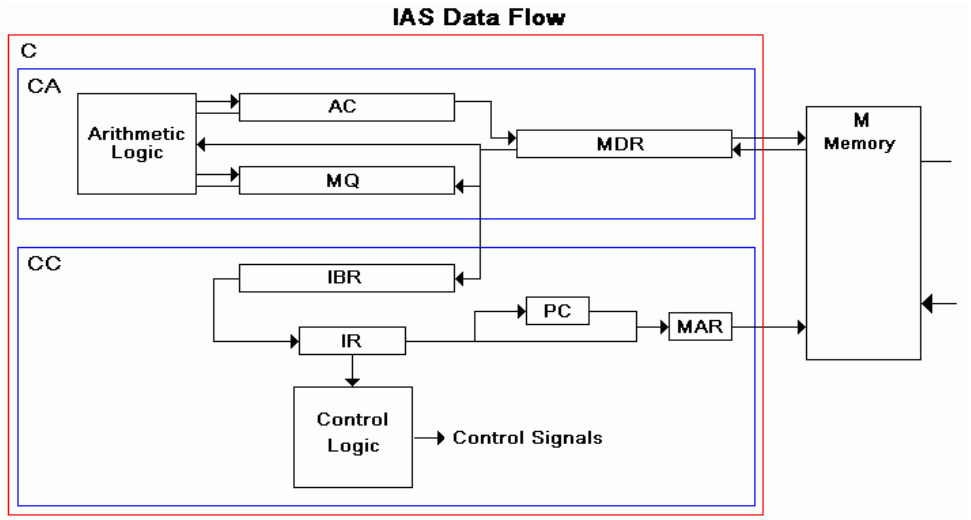
Page Number
992-1003

It should be store in memory Address like this.



Arranging of data elements and memory address we can study into the data structure.
According to IAS (institute of Advance Study) data are store in computers memory as flow
Internal Structure of Arranging data or data flow into the computer memory and how it will be store into the memory‘s register? Is given below.





Internal structure of data flow into the register
Aims and Objectives of Data Structures
• A data structure is a collection of one or more variables, possibly of different types, organized into a single "record" with a single name for easy reference and manipulation
• A data structure could be used to hold personnel details, co-ordinates of a complex 2- or 3-D shapes, etc.
• You can copy and assign to structures, pass them to functions, return them from functions, and initialize them
• Structures are syntactically analogous to data types
• A structure can have a tag (a name for the data type) and variables can be declared to be of that type

Classification of Data structures
Data structures can be classified as
1. Simple Data Structure
2. Compound Data Structure
   a. Linear Data Structure
   b. Non-Liner Data Structure

Simple Data Structure: Simple data structure can be constructed with the help of primitive data structure. A primitive data structure used to represent the standard data types of any one of the computer languages. Variables, arrays, pointers, structures, unions, etc. are examples of primitive data structures.
Compound Data structure:
Compound data structure can be constructed with the help of any one of the primitive data structure and it is having a specific functionality. It can be designed by user. It can be classified as.

a) Linear data structure
b) Non-linear data structure

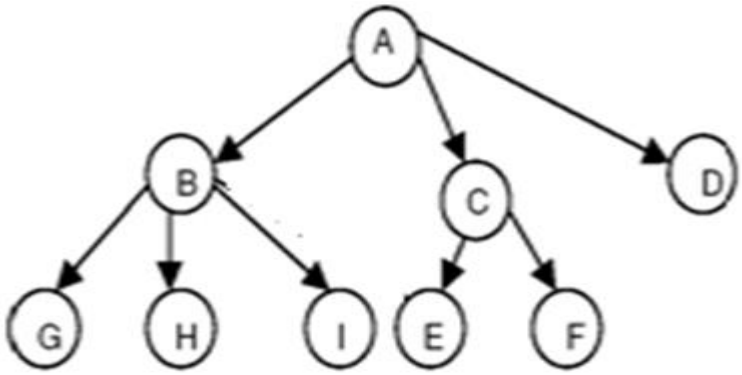
Linear data structure: Collection of nodes which are logically adjacent in which logical adjacency is maintained by pointers and we can say that. Linear data structures can be constructed as a continuous arrangement of data elements in the memory. It can be constructed by using array data type. In the linear Data Structures the relationship of adjacency is maintained between the Data elements, Example of liner data structure.

1. Stack
2. Queue
3. List

Non-linear data structure: Non-linear data structure can be constructed as a collection of randomly distributed set of data item joined together by using a special pointer or tag. In non-linear Data structure the relationship of adjacency is not maintained between the Data items, For Example of non-liner data structure.

1. Tree
2. Graph

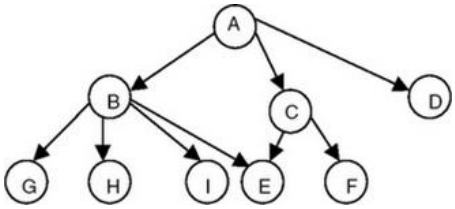
Tree Introduction: Arrays, linked lists, stacks and queues are used to represent linear and tabular data. These structures are not suitable for representing hierarchical data.
Trees are flexible, powerful non-linear data structure that can be used to represent data items which has hierarchical relationship; it is an ideal data structure for representing hierarchical data. And tree can be defined as a finite set of one or more data items (that is nodes) such that
(a) There is a special node called the root of the tree
(b) The remaining nodes are partitioned in to n disjointed set of nodes each of which is a sub tree.



Example of a tree(figure 1)

This is a tree contains a set of nodes {A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S},with node A as a root node and the remaining nodes partitioned in to 3 disjointed sets {B,E,F,L,M,N},{C,G,H,O} and {D,I,J,K,P,Q,R,S}.Each of these sets is a tree because each satisfies the tree definition.

• Example of a structure that is not a tree



This also contains a set of nodes {A,B,C,D,G,H,I,E,F} with node A as a root node but this is not a tree because the node E is shared. So we cant subdivide the nodes in to disjointed sets.

Tree Terminologies:

Node:

• Each data item in a tree is called a node.

• Node specifies the data information and links to other data items

Root:

• Root is the first node in the tree

• Node A is the root node in the tree example

Degree of a Node:

• Degree of a node is the number of subtrees of a node in a given tree.

• In figure1degree of node A is 3,degree of node B is 2,degree of node C is 2,degree of node D is 3 and degree of node j is 4

Degree of a Tree:

• The Degree of a tree is the maximum degree of node in a given tree.

• In figure1 node j has maximum degree as 4,all the other nodes have less or equal degree so the degree of the tree in figure 1 is 4

Leaf Node (or) Terminal Node:

• A node with degree 0 is called a leafnode

• ie a node with no subtrees or children is called leafnode.

• In figure1 nodes L,M,N,O,H,I,P,Q,R,S,K are leaf nodes

• Any node whose degree is non-zero is called non-terminal node

Level of a Node:

• The tree is structured in different levels.

• The root node is always at level 0.Then its immediate children are at level 1and their immediate children are at level 2 and so on up to the terminal nodes

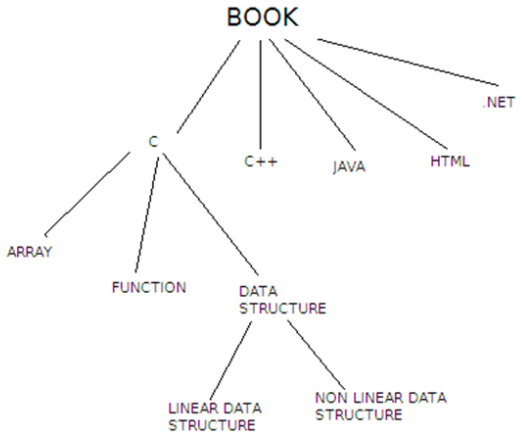
• ie If a node is at level n, then its children will be at level n+1.

Depth of a tree:

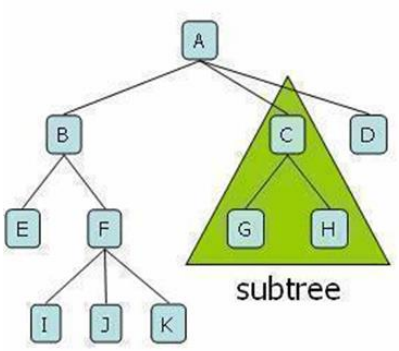
• Depth of a tree is the maximum level of any node in a given tree.

• The depth of a tree in figure 1 is 4.(including level 0)

Tree Structure

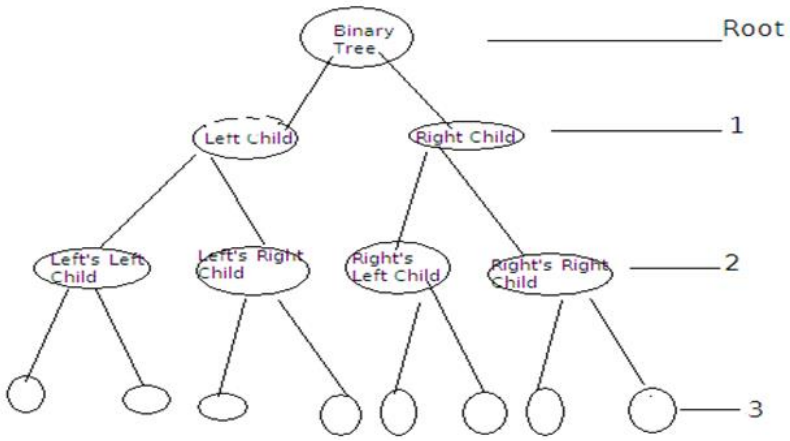


Use of Trees
• Fundamental data storage structures used in programming
• Combine advantages of ordered arrays and linked lists
• Searching can be made as fast as in ordered arrays
• Insertion and deletion as fast as in linked lists
Tree characteristics
• Consists of nodes connected by edges
• Nodes often represent entities (complex objects) such as people, car parts etc.
• Edges between the nodes represent the way the nodes are related.
• It's easy for a program to get from one node to another if there is a line connecting them.
• The only way to get from node to node is to follow a path
• along the edges.

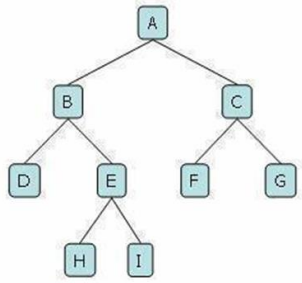


• Path: traversal from node to node along the edges that results in a sequence
• Root: node at the top of the tree
• Parent: any node, except root has exactly one edge running upward to another node. The node above it is called parent.
• Child: any node may have one or more lines running downward to other nodes. Nodes below are children.
• Leaf: a node that has no children
• Subtree: any node can be considered to be the root of a subtree, which consists of its children and its children's children and so on.
• Visiting: a node is visited when program control arrives at the node, usually for processing.
• Traversing: to traverse a tree means to visit all the nodes in some specified order.
• Levels: the level of a particular node refers to how many generations the node is from the root. Root is assumed to be level 0.
• Keys: key value is used to search for the item or perform other operations on it.
Height of Tree:
The maximum distance of any leaf from the root of atree. If a tree has only one node (the root), the height is zero. The height of a tree is also known as theorder.
The degree of a node is the number of subtrees associated with that node. For example, the degree of tree T is n. A node of degree zero has no subtrees. Such a node is called a leaf.
Binary Tree:
A binary tree T is defined as a finite set of elements, called nodes, such that
1. T is empty (called the null tree or empty tree)
2. T contains a distinguished node R called the root of T, and the remaining nodes of T form an ordered pair of disjoint binary tree T1 and T2.
If T does contain a root R, than the two trees, T1 and T2 are called respectively the left and right subtrees or R. If T1 is non empty, than its root is called the left successor of R, similarly, if T2 is nonempty, than its root is called the right successor of R.

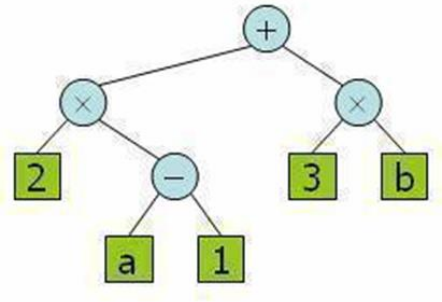
Every node in a binary tree can have at most two children. These nodes called the left child and right child corresponding to their positions.
A node can have only a left child or only a right child or it can have no children at all.



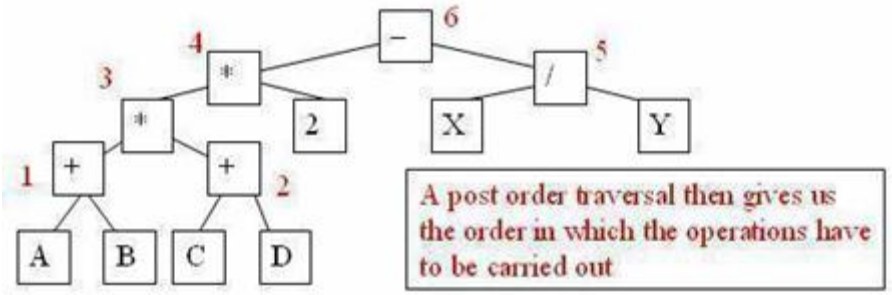
A binary tree is a tree with the following properties:
• Each internal node has at most two children (exactly two for proper binary trees)
• The children of a node are an ordered pair
• We call the children of an internal node left child and right cild
• Alternative recursive definition: a binary tree is either
• a tree consisting of a single node, or
• a tree whose root has an ordered pair of children, each of which is a binary tree
• Applications:
• arithmetic expressions
• decision processes
• searching



Arithmetic Expression Tree
• Binary tree associated with an arithmetic expression
o internal nodes: operators
o external nodes: operands
• Example: arithmetic expression tree for the expression (2 * (a – 1) + (3 * b))



Compiling arithmetic expressions
We can represent an arithmetic expression such as
(A + B) * (C + D) * 2 – X / Y as a binary tree, in which the leaves are constants or variables and the nodes are operations:



Properties of Proper Binary Trees Notation

- n number of nodes
- e number of external nodes
- i number of internal nodes
- h height

Properties

- e = i +1
- n =2e -1
- h <= i
- h <= (n -1)/2
- e <= 2h

Binary Search Tree: Binary Search tree is a binary tree in which each internal node T stores an element such that the element stored in the left subtree of T are less than or equal to T and elements stored in the right subtree of T are greater than or equal to T. T is called binary-search-tree property.

Traversing Binary Trees: There are three standard ways of traversing a binary tree T with root R. These three algorithm are called preorder, inorder and postorder.

Preorder:

1. Process the root R.
2. Traverse the left subtree of R in preorder.
3. Traverse the right subtree of R in preorder.

Inorder:

1. Traverse the left subtree of R in inorder.
2. Process the root R.
3. Traverse the right subtree of R in inorder.

Postorder:

1. Traverse the left subtree of R in postorder.
2. Traverse the right subtree of R in postorder.
3. Process the root R.

Breath-first Traversal: The breadth-first traversal of a tree visits the nodes in the order of their depth in the tree. Breadth-first traversal first visits all the nodes at depth zero (i.e., the root), then all the nodes at depth one, and so on. At each depth the nodes are visited from left to right.

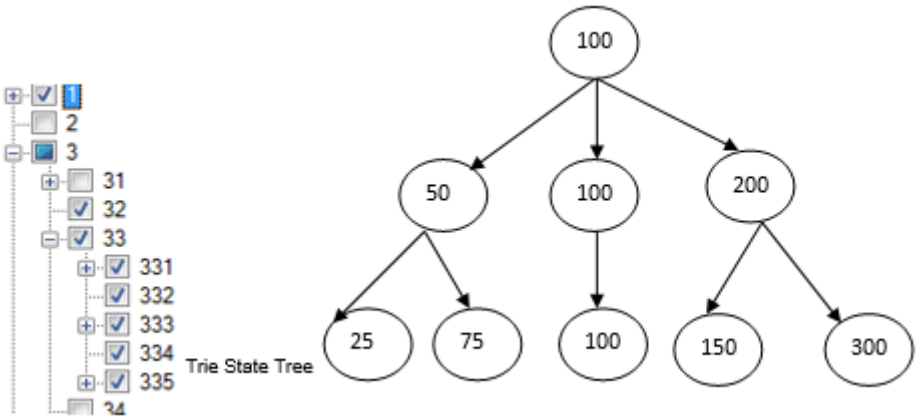
Depth-first Traversal: Depth-first traversal is an algorithm for traversing or searching a tree, tree structure, or graph. One starts at the root (selecting some node as the root in the graph case) and explores as far as possible along each branch before backtracking.

Height Balanced Tree: A height-balanced tree which is also a binary search tree. It supports membership, insert, and delete operations in time logarithmic in the number of nodes in the tree.

self-balancing binary search tree or height-balanced binary search tree is a binary search tree that attempts to keep its height, or the number of levels of nodes beneath the root, as small as possible at all times, automatically.

AVL Trees: An AVL tree is a self-balancing binary search tree, and it is the first such data structure to be invented. In an AVL tree, the heights of the two child subtrees of any node differ by at most one; therefore, it is

also said to be height-balanced. Lookup, insertion, and deletion all take O(log n) time in both the average and worst cases, where n is the number of nodes in the tree prior to the operation. Insertions and deletions may require the tree to be rebalanced by one or more tree rotations.



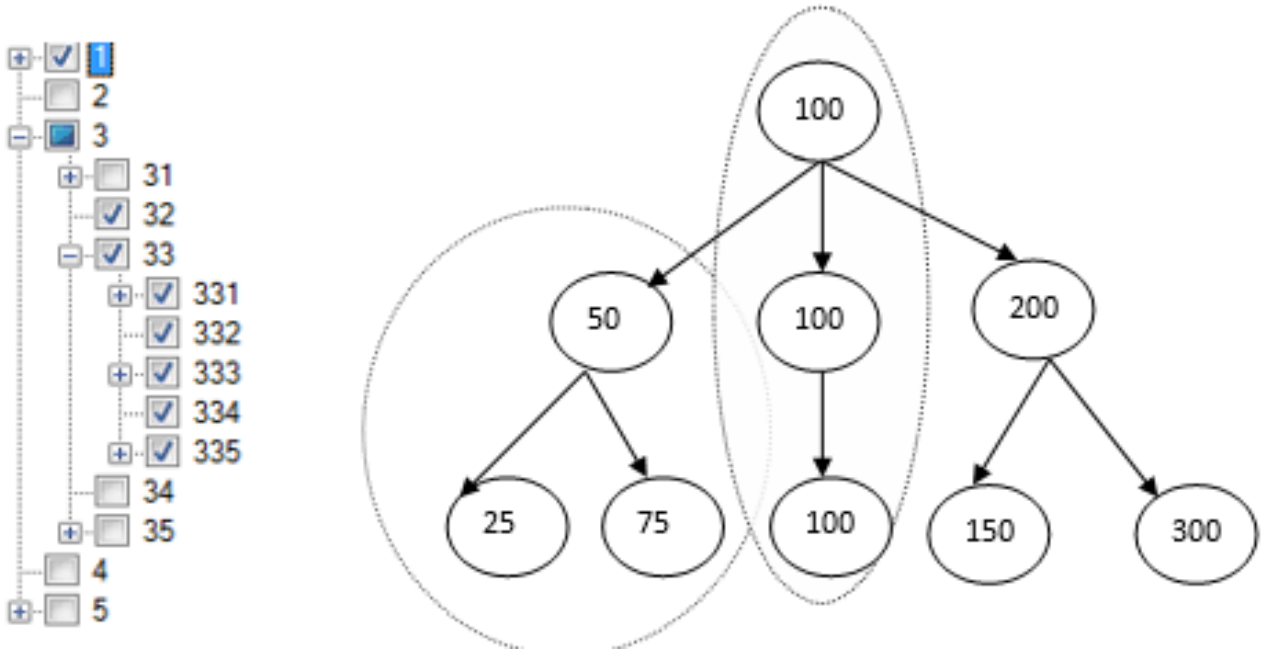
Trie State Tree

In the example shown, keys are listed in the nodes and values below them. Each complete the Number has an arbitrary integer value associated with it. A trie can be seen as a deterministic finite automaton, although the symbol on each edge is often implicit in the order of the branches.

It is not necessary for number to be explicitly stored in nodes. (In the figure, Numbers are shown only to illustrate how the trie works.)

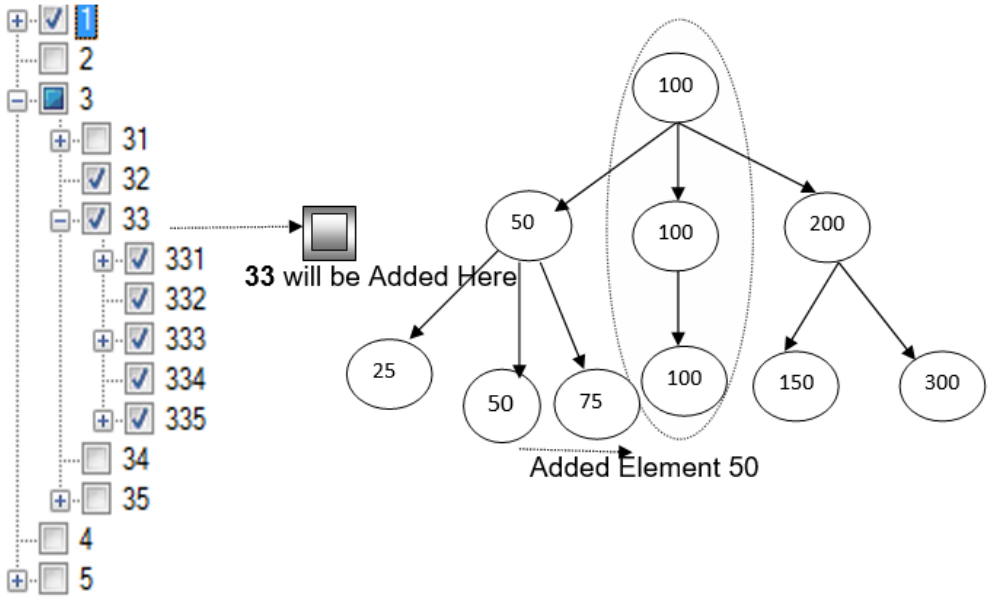
Though it is most common, tries need not be keyed by character strings. The same algorithms can easily be adapted to serve similar functions of ordered lists of any construct that is permutations on a list of digits or shapes. In particular, a bitwise trie is keyed on the individual bits making up a short, fixed size of bits such as an integer number or pointer to memory.

In the feature we can easily maintain large number of records using trie state tree representation. Before trie state tree records are also mention but when similar priority values are mentions then it insert the value in the side of similar priority values. These tree structure are mention similar value in just below of that similar priority nodes and new adage can also hold the values in the features.



Trie StateTree Diagram

User want to insert again 50 then the structure of trie state tree should be.

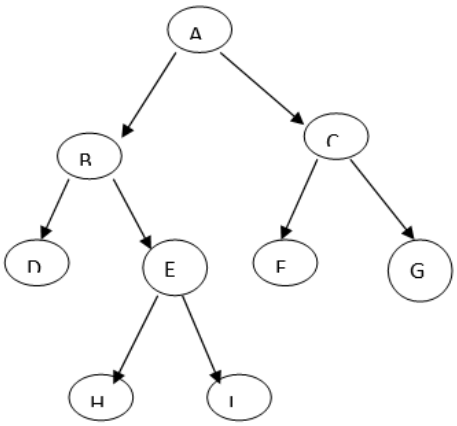


The trie state diagrams also add the 33 shape in the tree. There is no any concept to add this shape here. If we want to add then what should be position of this shape? The trie state tree clarify the position of 33 similarly we want to add new concept of data structure and that can also hold the values in the features.

User can add the similar element on the denoted doted point and it can also be hold some elements but greatest number and smallest number should be inserted on root positions means according to the rules of trees.

The main advantages of trie state trees is that we can easily move on the any number of elements and access them if it have same priority element then we cannot access the same priorities records when new nodes have some records, like it have double 33 shape and each have some value without going on particular shape we cannot access the record of that 33 shape.

Binary tree has three nodes as we know that root left and right. In real life Avery tree started with root. There is several ways to represent to tree structure one way is given below.



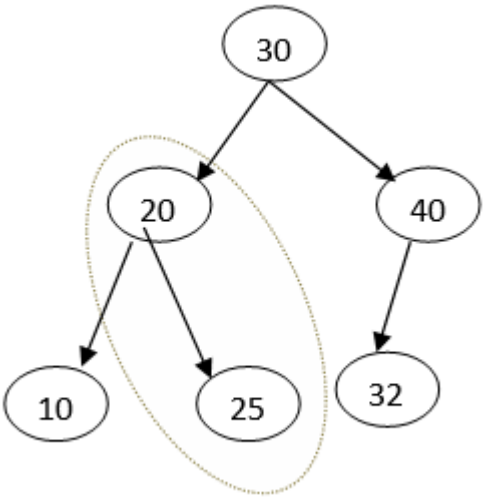
The root of binary tree defines as level 0 and sub tree is defined as 1 level similarly further node is defined.

Hear A is level 0 and B and C level is define level 2 and D E F G is define as level 2 and H I is define as level 3.

Here largest number is 3 which is the level number H and I the number of tree is 3+1=4.

There is main disadvantage of binary three is that in binary tree Difficult to get the sorted. Which is easy for BST?

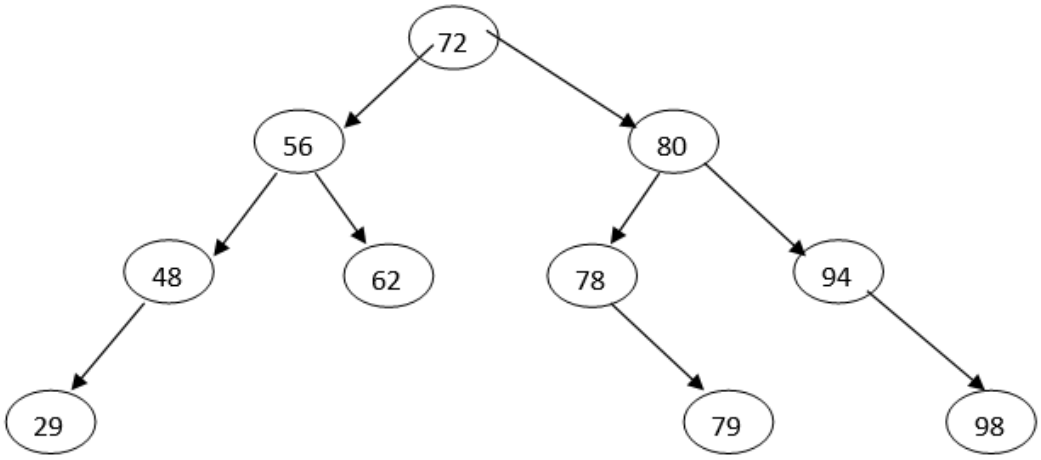
A binary search tree is a binary tree in which each node has value greater then every node of left sub tree and less then every tree node of right sub tree. In which data item can be searches in O (Log2 N) where N is the number of nodes



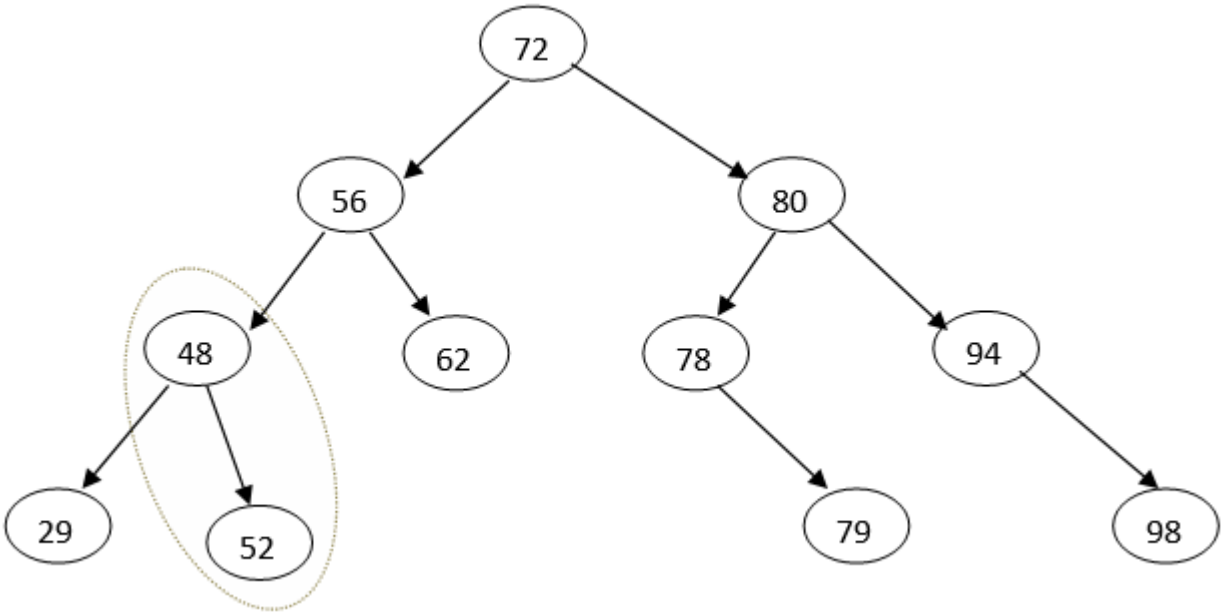
BST is fast in insertion and deletion etc when balanced. Very efficient and its code is easier than link lists. But Shape of the tree depends upon order of insertion and it can be degenerated. Searching takes long time. And BST cannot be balanced so it can be balanced in AVL tree.

Optimization of searching is totally dependent on the order of inserted element. If the binary searches tree is complete balance tree then this optimization can be achieved. We know that complete balance tree is an ideal situation but we can be near to this optimization of search time for insertion and deletion. The AVL come with the new concept for balancing binary searching tree

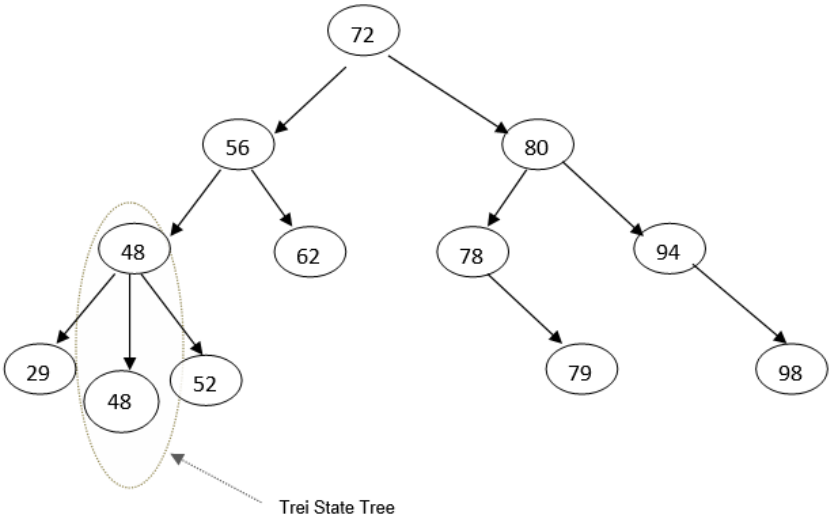
This tree has technique for efficient search and insertion so AVL complete binary tree.



We want to insert the 52 in the above tree



But if we want to again insert 48 then it cannot be balanced, so we implement new tree that is called trei state tree.



Here all duplicate element means same priority element will be added in the third edge. Here 48 will be added in the bellow of 48 nodes because it has same priority.

References:

1. Data Structure using C by A.K Sharma in 2011 ISBN 978-81-317-5566-2
2. Mastering Data Structures Through C Language By J. B. Dixit - 01-Aug-2010 published by university science place
3. Data Structures And Algorithms By A.A.Puntambekar - 01-Jan-2009
4. Data Structures Using C++ By D. S. Malik 2009
5. Programming In Ansi C By Balagurusamy - 07-Jul-2008
6. Data Structures and Algorithm Analysis in C++ by Weiss Mark Allen - 01-Sep-2007
7. Fundamentals Of Data Structures In C++ By Ellis Horowitz, Sahni, Dinesh Mehta - 2006
8. ADTs, Data Structures, and Problem Solving with C++ By Larry R Nyhoff – 2005

9. Data Structures Using C ISRD Group - 01-Nov-2005
10. Data Structures Using C By Bandyopadhyay - 01-Sep-2004
11. Data Structures: A Pseudocode Approach With C By Richard F. Gilberg, Behrouz A. Forouzan - 04-Oct-2004
12. C & Data Structures (With Cd) By Prof. P.S. Deshpande, Prof. O.G. Kakde - 2003
13. Data Structures Through C By Yashavant P. Kanetkar - 01-Feb-2003
14. Data Structures Using C By Aaro M Tenenbaum - 01-Sep-1990
15. Fundamentals of data structures By Ellis Horowitz, Sartaj Sahni – 1983
16. The C programming language By Brian W. Kernighan, Dennis M. Ritchie – 1978