International Journal of Research (IJR) Vol-1, Issue-10 November 2014 **ISSN 2348-6848**

A study on Binary Tree

Ankit Dalal1* and Ankur Atri2[1]

Department of Information Technology, Computer Science and Information Technology,
Dronacharya College of Engineering,Gurgaon-122001, India
**E-mail: ankit.16898@ggnindia.dronacharya.info, ankur.16900@ggnindia.dronacharya.info*

Abstract:

This research paper is a general overview of dynamic data structure called Tree which provides flexibility in adding new data elements and deleting existing data elements onto the structure.In computer science, a binary tree is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child.Binary trees are a good way to express arithmetic expressions. In this paper we have studied about binary tree. This paper covers discussion about properties of binary tree, types of binary trees, traversal techniques, searching, and insertion and deletion operation.

Keywords:

Node, root, child, sub-tree and siblings etc.

Introduction

A Tree is a dynamic, non-linear data structure which is an acyclic graph or graph having no cycles. This structure is mainly used to represent data containing a hierarchical relationship between elements.

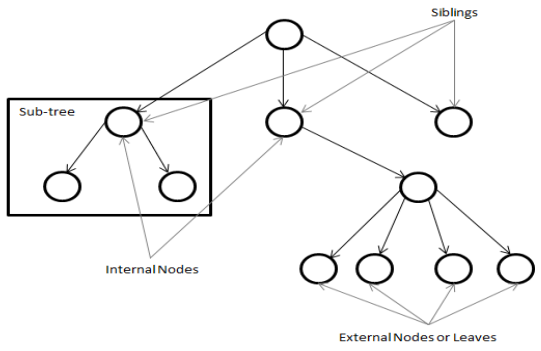


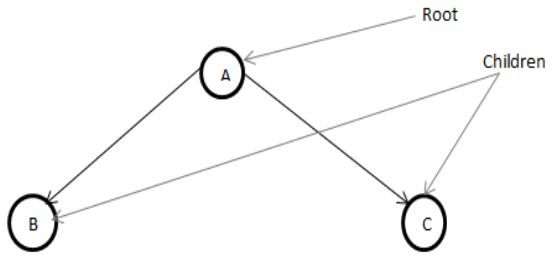
Figure 1

Binary trees are hierarchical data structures which allow insertion and a fast, nearest-neighbours search in one-dimensional data. It can be used instead of qsort and binary search to quickly find the closest points in a data array. The binary tree has the advantage of having a simple structure that allows generalization for more then one dimension - the so-called KD Tree[1]. Therefore, it is good to understand how it works and how it performs data searches.In this data is stored in non-consecutive memory location or inconsequential form. There is no unique predecessor or unique successor. A tree data structure is depicted upside down with the root at the top and the leaves at the bottom[2,3].

Basic Terminology

Root

A root is a specially designated node in a tree which has no parent. There can be only one root in a tree. In Fig 2. A is the root node.



Node

Figure 2

This is the main components of any tree structure. It stores the actual data along with links to other nodes. In Fig 2. A, B and C are nodes because these are holding data elements.

Parent

A node having left child or right child or both is termed as parent node. In Fig 2.node A is parent of node B and C.

Child

The immediate successors of a node are called child nodes. A child which is placed at the left side is called the left child and a child which is placed at the right side is called the right child[4]. In Fig 2.node B and C are children of node A.

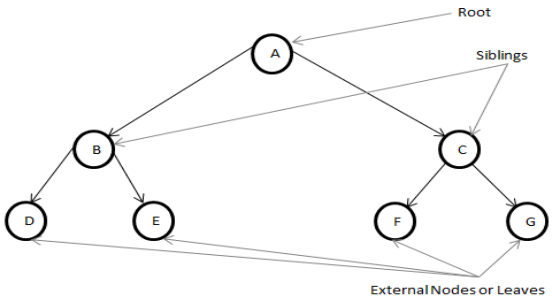


Figure 3

Leaf

A leaf node is the external node which does not have any child node. In Fig 3. D, E, F, G are leaf nodes.

Siblings

The child nodes of a given parent node are called siblings. In Fig 3.nodes B and C are siblings of node A.

Branch

It is a connecting line between two nodes. A node can have more than one edge.

Path

Each node has to be reachable from the root through a unique sequence of edges called a path. The number of edges in a path is called the length of path.

Level of a Node

The level of node is its distance from the root. The level of root is defined as zero. The level of all other nodes is one more than its parent node.

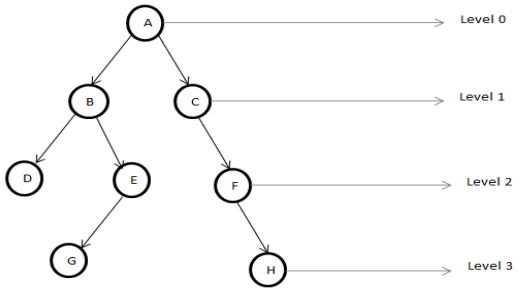


Figure 4

Height of a Tree

The height is defined as the maximum number of nodes in a branch of tree. This is one more than the maximum level of the tree. The height of tree in Fig 4.is 4.

Rooted Tree

A rooted tree is one where we designate one node as the root. In Fig 4.node A is the root node. Fig 4.is a rooted tree.

Ordered Tree

If in a tree at each level, an ordering is defined, then such a tree is called an ordered tree[5,6].

Binary Tree

A binary tree is a rooted tree in which each node has at most two children, which are referred to as the *left* child and the *right* child.

Properties of Binary tree

1. The number of external nodes is one more than internal nodes.
2. The number of external nodes is at least h + 1 and at most 2h, where h is the height of the tree.
3. The number of internal nodes is at least h and at most $2^h - 1$.
4. The total number of nodes in a binary tree is at least 2h + 1 and at most 2h + 1 – 1.
5. The height h, of binary tree with n nodes is at least log n + 1 and at most n.
6. A binary tree with n nodes has exactly n – 1 edges[7].

Types of Binary Tree

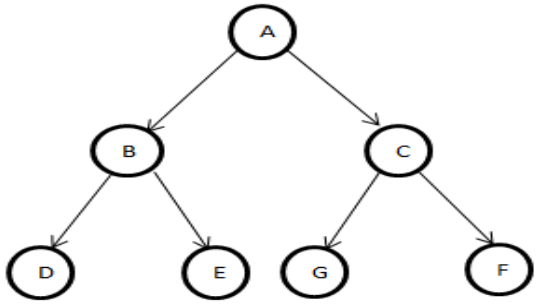
Strictly Binary Tree

A binary tree is a strictly binary tree if and only if each node has exactly two child nodes or no nodes. A strictly binary tree with n leaves always contains 2n – 1 nodes.

Complete binary tree

A binary tree is a full or complete binary tree if each non-leaf node has exactly two child nodes and all leaf nodes are at the same level.

Figure 5



Almost Complete Binary Tree

A binary tree of depth h is an almost complete binary tree, if any node at level less than d – 1 has two children. For any node 'x' in the tree with a right descent at level d, x must have a left child and every left descendant of x is either a leaf at level d or has two children. An almost complete binary tree with n leaves has 2n – 1 nodes and an almost complete binary tree with n leaves which is not strictly binary has 2n nodes[8].

Traversing in a Binary Tree

Traversing means visiting all the nodes of tree. There are three standard methods to

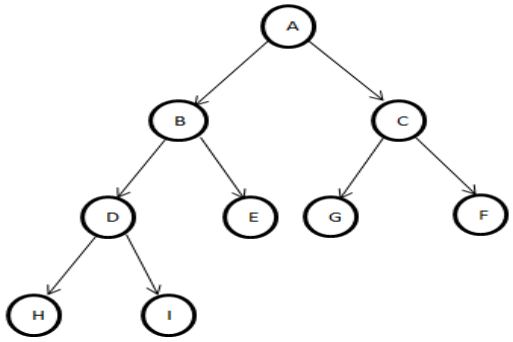


Figure 6

traverse the binary trees

i. Preorder traversal

ii. Postorder traversal

iii. Inorder traversal

Preorder traversal

The preorder traversal of a binary tree is a recursive process. The preorder traversal of a tree is

a. Visit the root of the tree.

b. Traverse the left subtree in preorder

c. Traverse the right subtree in preorder

Postorder traversal

The postorder traversal of a binary tree is a recursive process. The postorder traversal of a tree is

a. Traverse the left subtree in postorder

b. Traverse the right subtree in postorder

c. Visit the root of the tree.

Inorder traversal

The inorder traversal of a binary tree is a recursive process. The preorder traversal of a tree is

a. Traverse the left subtree in inorder

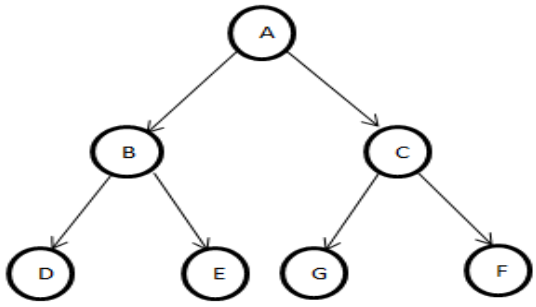
b. Visit the root of the tree.

c. Traverse the right subtree in inorder

That is,

Pre-order	Root	Left-sub-tree	Right-sub-tree
In-order	Left-sub-tree	Root	Right-sub-tree
Post-order	Left-sub-tree	Right-sub-tree	Root

Example: Consider the following binary tree.



Preorder Traversal: A, B, D, E, C, G, F

Figure 7

Postorder Traversal: D, E, B, G, F, C, A

Inorder Traversal: D, B, E, A, G, C, F

Binary Search tree

Binary search tree has the property that the left child contains a smaller value than the root node and the right child contains a larger value than the root node.

Operations on Binary Search Tree

The most common operation performed on a Binary Search Tree is searching for a key/element stored in the tree. Besides the search operation, it supports many operations such as traversal operation insertion and deletion of data etc.

Searching

Searching in Binary Search Tree is much faster as compared to other data structures such as array or linked lists. The TREE-SEARCH (x,k) algorithm searches the tree root at x for a node whose key value is

equal to k. It returns a pointer to the node if it exists otherwise NIL[9].

TREE-SEARCH (x,k)

1. if x = NIL or k = key[x]
2. then return x
3. if k < key[x]
4. then return TREE-SEARCH(left[x],k)
5. else return TREE-SEARCH(right[x],k)

Insertion Operation

The process of adding new element is called insertion. Insertion of element in the binary search tree invokes adding of element to the leaf node.

Consider a binary search tree T. Suppose an ITEM of information to be inserted in T. Then the procedure written below is followed

1. Compare ROOT with ITEM.
2. If ITEM > ROOT, proceed to right child and it becomes root for the right subtree.
3. If ITEM < ROOT, proceed to left child.
4. Repeat the above steps until LEAF is reached.
5. Now compare ITEM with LEAF
    a. if LEAF > ITEM, insert ITEM as left child
    b. if LEAF < ITEM, insert ITEM as right child

Deletion Operation

The process of deleting an element is called deletion. Deletion can be performed on any node i.e. internal node or external node.

Consider a binary search tree T. Suppose an ITEM of information to be deleted from T. To delete an ITEM, there are three cases, depending upon the number of children of the node to be deleted.

1. If the node to be deleted is a leaf node then just replace the node with NULL.
2. If the node z to be deleted has a child node then simply swap the root node and child node and then replace the child node with NULL.
3. If the node z to be deleted has a left child and right child or left subtree and right subtree then find the inorder successor of the node to be deleted and then swap the values of node to be deleted and inorder successor in the binary search tree and replace inorder successor with NULL[10].

Conclusion

The hierarchical file system tell us, how do files get saved and deleted under hierarchical directories. Tree is a data structure which allows you to associate a parent-child relationship between various pieces of data and thus allows us to arrange our records, data and files in a hierarchical fashion. The central focus in this paper is to know more about binary tree. We have learnt properties of binary tree, its various types. Working with binary search tree i.e. search, insertion and deletions operations is done along with algorithms. We have also learnt tree traversal techniques i.e. reorder, in order and post order traversal.

REFERENCES

1. Rowan Garnier; John Taylor (2009). *Discrete Mathematics: Proofs, Structures and Applications, Third Edition*. CRC Press. p. 620. ISBN 978-1-4398-1280-8.
2. Steven S Skiena (2009). *The Algorithm Design Manual*. Springer Science & Business Media. p. 77. ISBN 978-1-84800-070-4.
3. Knuth (1997). *The Art Of Computer Programming, Volume 1, 3/E*. Pearson Education. p. 363. ISBN 0-201-89683-4.
4. Iván Flores (1971). *Computer programming system/360*. Prentice-Hall. p. 39.
5. Kenneth Rosen (2011). *Discrete Mathematics and Its Applications, 7th edition*. McGraw-Hill Science. p. 749. ISBN 978-0-07-338309-5.
6. David R. Mazur (2010). *Combinatorics: A Guided Tour*. Mathematical Association of America. p. 246. ISBN 978-0-88385-762-5.
7. Alfred V. Aho; John E. Hopcroft; Jeffrey D. Ullman (1983). *Data Structures and Algorithms*. Pearson Education. section 3.4: "Binary trees". ISBN 978-81-7758-826-2.
8. J.A. Storer (2002). *An Introduction to Data Structures and Algorithms*. Springer Science & Business Media. p. 127. ISBN 978-1-4612-6601-3.
9. David Makinson (2009). *Sets, Logic and Maths for Computing*. Springer Science & Business Media. p. 199. ISBN 978-1-84628-845-6.
10. Jonathan L. Gross (2007). *Combinatorial Methods with Computer Applications*. CRC Press. p. 248. ISBN 978-1-58488-743-0.