© 2014 IJIRT | Volume 1 Issue 7 | ISSN: 2349-6002

Tree concept in data structure

Rubi Dhankhar , Sapna Kamra , Vishal Jangra

Abstract- **This paper purposes the tree concept in data structure. Terminology used in tree concept is also discussed in details. Difference between data type and data structure , tree traversal concept are also explained in details. There are many different ways to represent trees. These methods are also dicussed in this paper.**

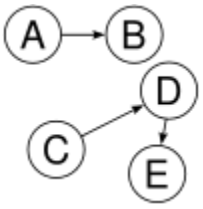
Index Terms- **tree , node , path , traversal , null , parent node.**

I. INTRODUCTION

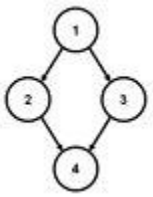
In computer science, a **tree** is a broadly used abstract data type (ADT) or data structure implementing this ADT that simulates a hierarchical tree structure, with a root value and subtrees of children, represented as a set of linked nodes. A tree data structure can be explained recursively (locally) as a collection of nodes (starting at a root node), where each node is a data structure consisting of a value, together with a list of references to nodes (the "children"), with the constraints that no reverence is duplicated, and none points to the root.

II. DISCUSSION

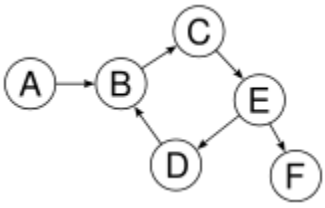
Definition:



Not a tree: two non-connected parts, A→B and C→D→E

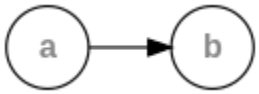


Not a tree: undirected cycle 1-2-4-3



Not a tree: cycle B→C→E→D→B

Not a tree: cycle A→A



Each linear list is trivially a tree

A tree is a (possibly non-linear) data structure made up of nodes or vertices and edges outside having any cycle. The tree having no nodes is called the **null** or **empty** tree. A tree that is not empty consists of a root node and potentially many levels of additional nodes that form a order.

Terms used in Trees

- **Root** – The top node in a tree is known as root node.
- **Parent** – The opposite notion of *child*.
- **Siblings** – Nodes having same parent.
- **Descendant** – a node accessible by repeated proceeding from parent to child.
- **Ancestor** – a node accessible by repeated proceeding from child to parent.
- **Leaf** – a node have no children.
- **Internal node** – a node having at least one child.
- **External node** – a node accompanying no children.
- **Edge** – associate between one node to another.
- **Path** – a sequence of nodes and edges associating a node with a descendant.

- **Level** – The level of a node is explained by 1 + the number of connections between the node and the root.
- **Height of tree** –The height of a tree is the number of edges on the longest downward path among the root and a leaf.
- **Height of node** –The height of a node is the number of edges on the longest downward path among that node and a leaf.
- **Depth** –The depth of a node is the figure of edges from the node to the tree's root node.

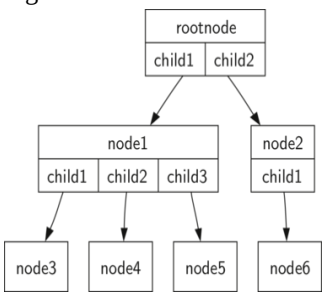


Fig: A Tree Consisting of a Set of Nodes and Edges

Data type vs. data structure

There is a distinction among a tree as an abstract data type and as a data structure, analogous to the distinction between a list and a linked list.

As a data type, a tree has a value and children, and the children are themselves trees; the value and children of the tree are integrated as the value of the root node and the subtrees of the children of the root node. To confess finite trees, one must either allow the list of children to be empty (in which case trees can be required to be non-empty, an "empty tree" instead being represented by a forest of zero trees), or allow trees to be empty, in which case the list of children can be of permanent size (branching factor, especially 2 or "binary"), if desired.

As a data structure, a linked tree is a group of nodes, where each node has a value and a list of reverences to other nodes (its children). This data structure actually explaines a directed graph, because it may have loops or several reverences to the same node, just as a linked list may have a loop. Thus there is also the requirement that no two reverences point to the same node (that each node has at most a single parent, and in fact exactly one parent, except for the root), and a tree that violates this is "corrupt".

Terminology

A **node** is a structure which may contain a value or condition, or represent a disconnected data structure (which could be a tree of its own). Each node in a tree has zero or more **child nodes**, which are lower it in the tree (by convention, trees are drawn growing downwards). A node having a child is called the child's **parent node** A node has merely one parent.

An **internal node** is any node of a tree that has child nodes. Similarly, an **external node** an **outer node**, is any node that does not have child nodes.

The apical node in a tree is called the **root node**. Depending on definition, a tree may be recommended to have a root node, or may be allowed to be empty, in which case it does not compulsorily have a root node. Being the apical node, the root node will not have a parent. It is the node at which algorithms on the tree begin, because as a data structure, one can only pass from parents to children. Note that some algorithms (such as post-order depth-first search) start at the root, but first visit leaf nodes (access the value of leaf nodes), only visit the root last (i.e., they first access the children of the root, but only ingress the *value* of the root last). All other nodes can be attained from it by following **edges** or **links**. In diagrams, the root node is normally drawn at the top. In some trees, such as heaps, the root node has distinct properties.

The **height** of a node is the length of the longest earthward path to a leaf from that node. The height of the root is known as the height of the tree. The **depth** of a node is the distance of the path to its root. This is commonly required in the manipulation of the various self-balancing trees, AVL Trees in particular. The root node has depth zero, leaf nodes have height zero, and a tree with only a single node (therefore both a root and leaf) has depth and height zero. Commonly, an empty tree (tree with no nodes, if such are allowed) has depth and height −1.

Representations

There are many different ways to represent trees; common representations represent the nodes as demoniacally allocated records with pointers to their children, their parents, or both, or as items in an array, with relationships between them determined by their positions in the array (e.g., binary heap).

Traversal methods

Tree traversal

Stepping through the items of a tree, distinctly of the connections between parents and children, is called **walking the tree**, and the action is a **walk** of the tree. generally, an operation might be performed when a pointer arrives at a particular node. A walk in which each parent node is passed through before its children is called a **pre-order** walk; a walk in which the children are traversed before their respective parents are traversed is called a **post-order** walk; a walk in which a node's left subtree, then the node itself, and finally its right subtree are traversed is called an **in-order** traversal.A **level-order** walk efficiently performs a breadth-first search over the entirety of a tree; nodes are traversed level by level, where the root node is visited first, followed by its direct child nodes and their siblings, followed by its grandchild nodes and their siblings, etc., instill all nodes in the tree have been traversed.

III. APPLICATIONS

- Representing stratified data
- Storing data in a way that makes it easily reachable
- exhibiting sorted lists of data
- As a workflow for complotting digital images for visual effects
- Chasing algorithms

IV. CONCLUSION

This paper concludes Trees are used in many areas of computer science, consisting operating systems, graphics, database systems, and computer networking. Tree data structures have many same things with their botanical cousins. A tree data structure consist a root, branches, and leaves. The difference among a tree in nature and a tree in computer science is that a tree data structure has its root at the top and its leaves on the bottom.

REFERENCES

[1] ece.colorado.edu/bart/book
[2] tree concept - en.Wikipedia.org