**TREES DATA STRUCTURE**

**Trees**

* store elements hierarchically
* the top element is root
* except the root, each element has a parent
* each element has 0 or more children

**Definition of trees**

• A tree T is a set of nodes storing elements such that the nodes have a parent-child relationship that satisfies the following

• if T is not empty, T has a special tree called the root that has no parent

• each node v of T different than the root has a unique parent node w; each node with parent w is a child of w

**Recursive definition**

• T is either empty

• or consists of a node r (the root) and a possibly empty set of trees whose roots are the children of r.

**Terminology**

***Root*  -** The top node in a tree.

***Child* -** A node directly connected to another node when moving away from the root.

***Parent* -** The converse notion of a child.

***Degree* -** For a given node, its number of children. A leaf is necessarily degree zero.

***Edge* -** The connection between one node and another.

***Path* -** A sequence of nodes and edges connecting a node with a descendant.

***Level* -** The level of a node is defined as: 1 + the number of edges between the node and the root.

***Height of node* -** The height of a node is the number of edges on the longest path between that node and a leaf.

***Height of tree* -** The height of a tree is the height of its root node.

***Depth* -** The depth of a node is the number of edges from the tree's root node to the node.

***Forest* -** A forest is a set of n ≥ 0 disjoint trees

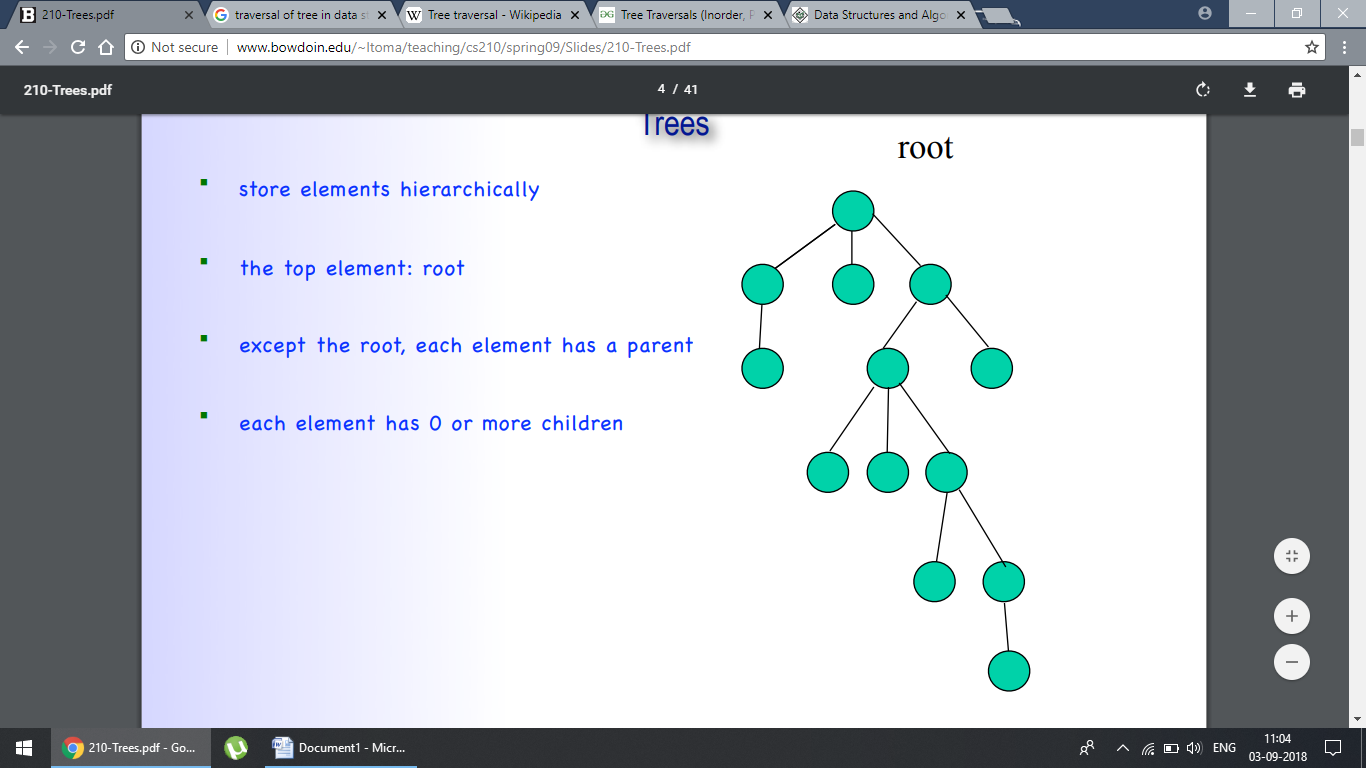
**siblings** - two nodes that have the same parent are called siblings

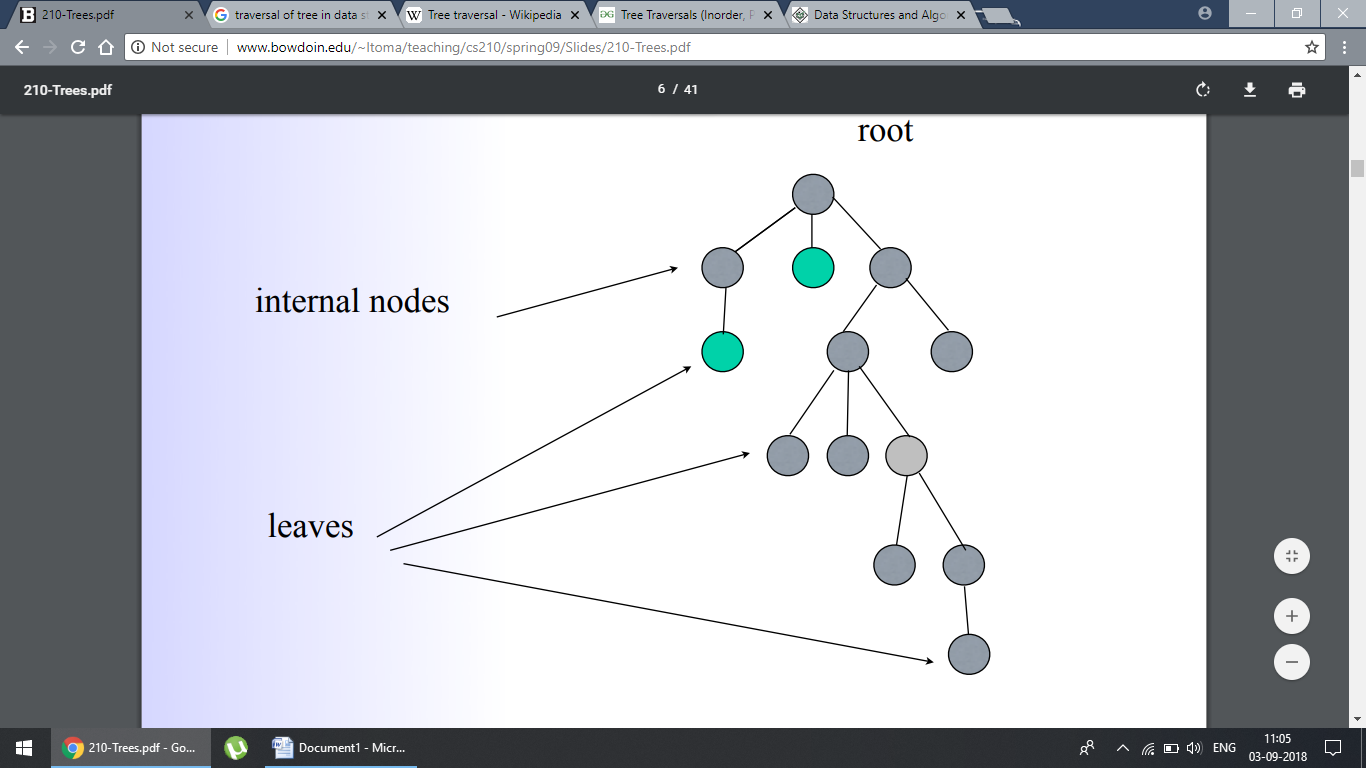
**internal nodes or branch nodes** - nodes that have children

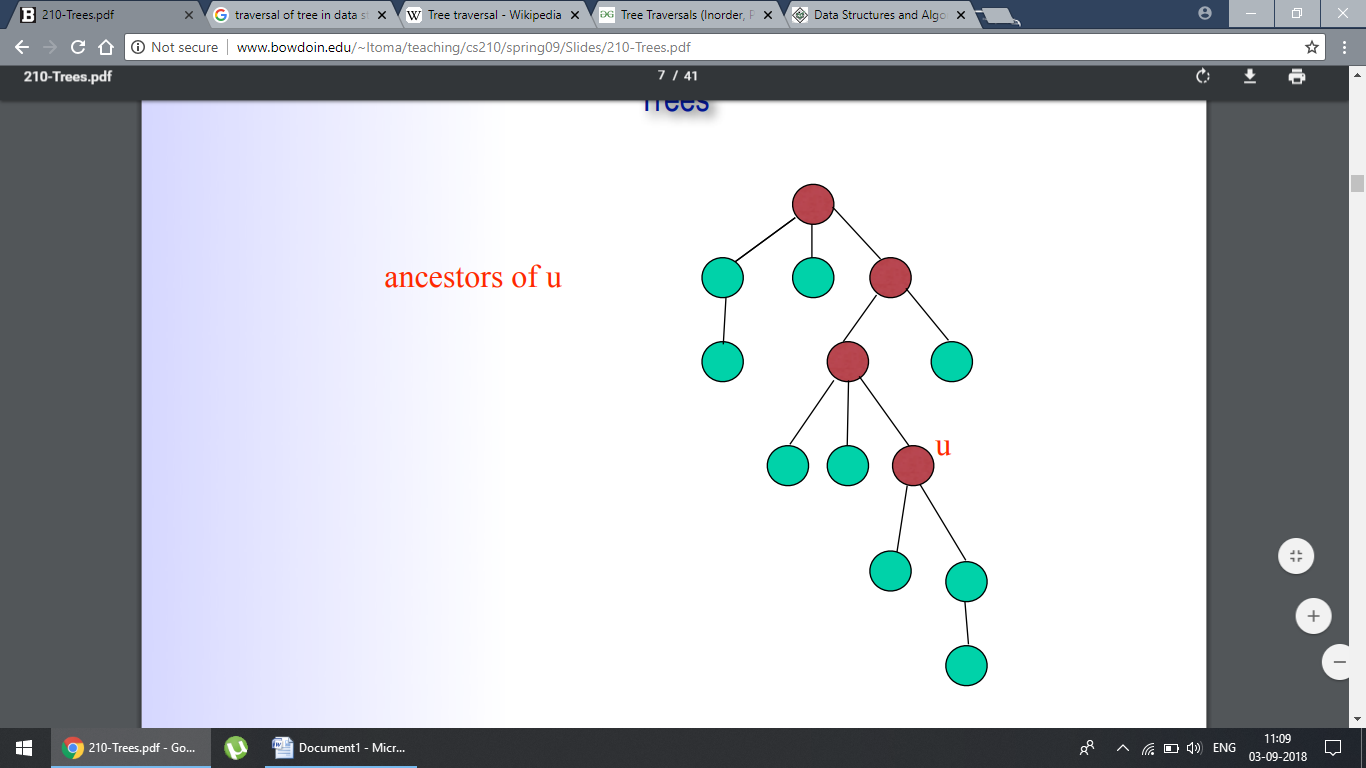
**external nodes or leaf** - nodes that don’t have children

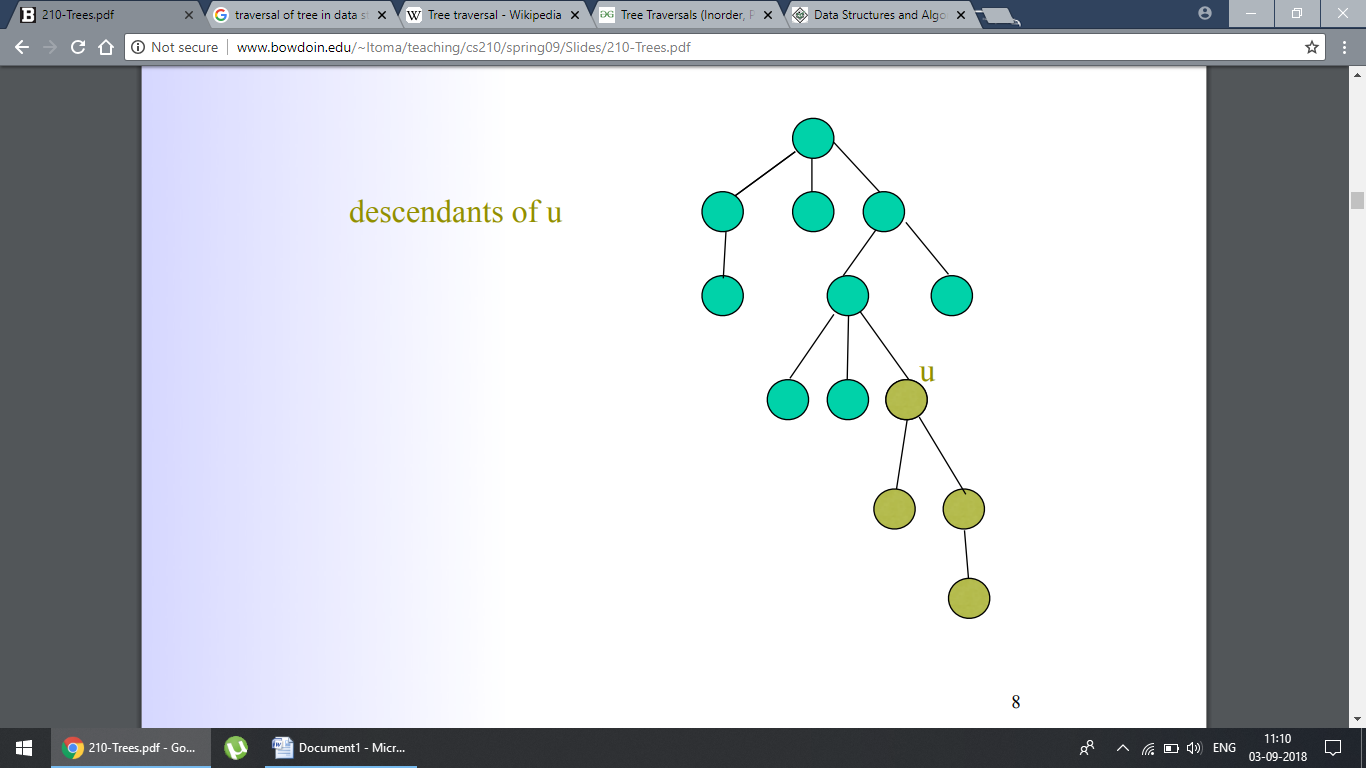
**ancestors -** A node reachable by repeated proceeding from child to parent.

**Descendants -** A node reachable by repeated proceeding from parent to child. Also known as subchild.











**Tree Traversal**

A traversal is a process that visits all the nodes in the tree. Since a tree is a nonlinear data structure, there is no unique traversal. We will consider several traversal algorithms with we group in the following two kinds

* depth-first traversal
* breadth-first traversal



* **Depth First Traversals**  
  (a) **Inorder** (Left, Root, Right) : 4 2 5 1 3  
  (b) **Preorder** (Root, Left, Right) : 1 2 4 5 3  
  (c) **Postorder** (Left, Right, Root) : 4 5 2 3 1
* **Inorder Traversal:**

Algorithm Inorder(tree)

1. Traverse the left subtree, i.e., call Inorder(left-subtree)

2. Visit the root.

3. Traverse the right subtree, i.e., call Inorder(right-subtree)

Example: Inorder traversal for the above-given figure is 4 2 5 1 3.

**Uses of Inorder**  
Inorder traversal gives nodes in non-decreasing order.

* **Preorder Traversal:**

Algorithm Preorder(tree)

1. Visit the root.

2. Traverse the left subtree, i.e., call Preorder(left-subtree)

3. Traverse the right subtree, i.e., call Preorder(right-subtree)

Example: Preorder traversal for the above given figure is 1 2 4 5 3.

**Uses of Preorder**Preorder traversal is used to create a copy of the tree. Preorder traversal is also used to get prefix expression on of an expression tree.

* **Postorder Traversal:**
* Algorithm Postorder(tree)
* 1. Traverse the left subtree, i.e., call Postorder(left-subtree)
* 2. Traverse the right subtree, i.e., call Postorder(right-subtree)
* 3. Visit the root.

Example: Postorder traversal for the above given figure is 4 5 2 3 1.

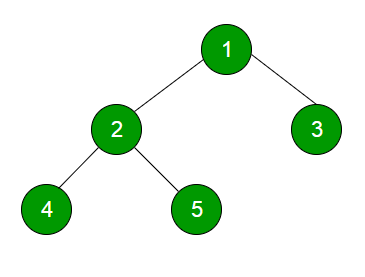
**Uses of Postorder**

Postorder traversal is used to delete the tree. Postorder traversal is also useful to get the postfix expression of an expression tree.

* **Breadth-first traversal:**

Another way is to go through them level-by-level.

Level order traversal of a tree is breadth first traversal for the tree.



Breadth-first traversal of the above tree is 1 2 3 4 5

For example, each element exists at a certain level (or depth) in the tree:

tree

----

j <-- level 0

/ \

f k <-- level 1

/ \ \

a h z <-- level 2

\

d <-- level 3

So, if we want to visit the elements level-by-level (and left-to-right, as usual), we would start at level 0 with **j**, then go to level 1 for **f** and **k**, then go to level 2 for **a, h** and **z**, and finally go to level 3 for **d**.

This level-by-level traversal is called a breadth-first traversal because we explore the breadth, i.e., full width of the tree at a given level, before going deeper.

**Main applications of trees include:**  
**1.** Manipulate hierarchical data.  
**2.** Make information easy to search (see tree traversal).  
**3.** Manipulate sorted lists of data.  
**4.** As a workflow for compositing digital images for visual effects.  
**5.**Router algorithms  
**6.**Form of a multi-stage decision-making (see business chess).

**Advantages of trees in data structures**

* The tree structure is useful because it easily accommodates the creation and deletion of folders and files.
* Trees reflect structural relationships in the data
* Trees are used to represent hierarchies
* Trees provide an efficient insertion and searching
* Trees are very flexible data, allowing to move subtrees around with minimum effort

**Disadvantages of trees in data structures**

The shape of the tree depends on the order of insertions, and it can be degenerated.

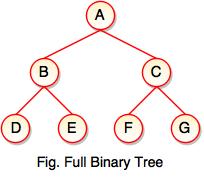
**Binary Tree**

A tree whose elements have at most 2 children is called a binary tree. Since each element in a binary tree can have only 2 children, we typically name them the left and right child.

**There are four types of binary tree:**   
  
1. Full Binary Tree  
2. Complete Binary Tree  
3. Skewed Binary Tree  
4. Extended Binary Tree

#### 1. Full Binary Tree

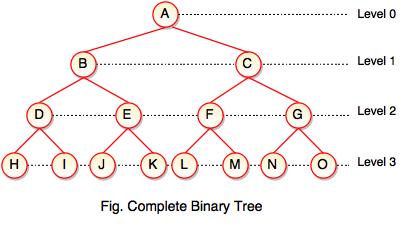
* If each node of binary tree has either two children or no child at all, is said to be a **Full Binary Tree**.
* Full binary tree is also called as **Strictly Binary Tree**.



* Every node in the tree has either 0 or 2 children.
* Full binary tree is used to represent mathematical expressions.

#### 2. Complete Binary Tree

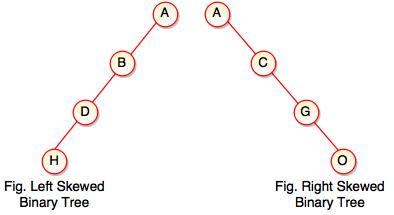
* If all levels of tree are completely filled except the last level and the last level has all keys as left as possible, is said to be a **Complete Binary Tree**.
* Complete binary tree is also called as **Perfect Binary Tree**.



* In a complete binary tree, every internal node has exactly two children and all leaf nodes are at same level.
* For example, at Level 2, there must be 22 = 4 nodes and at Level 3 there must be 23 = 8 nodes.

#### 3. Skewed Binary Tree

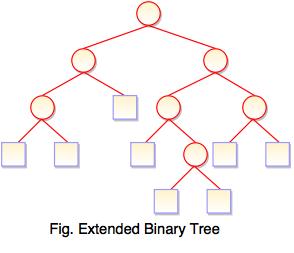
* If a tree which is dominated by left child node or right child node, is said to be a **Skewed Binary Tree**.
* In a skewed binary tree, all nodes except one have only one child node. The remaining node has no child.



* In a left skewed tree, most of the nodes have the left child without corresponding right child.
* In a right skewed tree, most of the nodes have the right child without corresponding left child.

#### 4. Extended Binary Tree

* Extended binary tree consists of replacing every null subtree of the original tree with special nodes.
* Empty circle represents internal node and filled circle represents external node.
* The nodes from the original tree are internal nodes and the special nodes are external nodes.
* Every internal node in the extended binary tree has exactly two children and every external node is a leaf. It displays the result which is a **complete binary tree**.



* **Binary Tree Representations**

A binary tree data structure is represented using two methods. Those methods are as follows...

1. Array Representation
2. Linked List Representation

Consider the following binary tree...



1. **Array Representation**

In array representation of binary tree, we use a one dimensional array (1-D Array) to represent a binary tree.

Consider the above example of binary tree and it is represented as follows...



To represent a binary tree of depth 'n' using array representation, we need one dimensional array with a maximum size of 2n+1 - 1.

1. **Linked List Representation**

We use double linked list to represent a binary tree. In a double linked list, every node consists of three fields. First field for storing left child address, second for storing actual data and third for storing right child address.

In this linked list representation, a node has the following structure...



The above example of binary tree represented using Linked list representation is shown as follows...



**Binary Tree Traversals**

**There are three types of binary tree traversals:**

* 1. Inorder Traversal
  2. Preorder Traversal
  3. Postorder Traversal
* **Inorder Traversal:**

Algorithm Inorder(tree)

1. Traverse the left subtree, i.e., call Inorder(left-subtree)

2. Visit the root.

3. Traverse the right subtree, i.e., call Inorder(right-subtree)

* **Preorder Traversal:**

Algorithm Preorder(tree)

1. Visit the root.

2. Traverse the left subtree, i.e., call Preorder(left-subtree)

3. Traverse the right subtree, i.e., call Preorder(right-subtree)

* **Postorder Traversal:**
* Algorithm Postorder(tree)
* 1. Traverse the left subtree, i.e., call Postorder(left-subtree)
* 2. Traverse the right subtree, i.e., call Postorder(right-subtree)
* 3. Visit the root.

Consider the following binary tree...



**In-Order Traversal** for above example of binary tree is

I - D - J - B - F - A - G - K - C – H

**Pre-Order Traversal** for above example binary tree is

A - B - D - I - J - F - C - G - K – H

**Post-Order Traversal** for above example binary tree is

I - J - D - F - B - K - G - H - C – A

## Node

Define a node having some data, references to its left and right child nodes.

struct node {

int data;

struct node \*leftChild;

struct node \*rightChild;

};

## Search Operation

Whenever an element is to be searched, start searching from the root node. Then if the data is less than the key value, search for the element in the left subtree. Otherwise, search for the element in the right subtree. Follow the same algorithm for each node.

## Insert Operation

Whenever an element is to be inserted, first locate its proper location. Start searching from the root node, then if the data is less than the key value, search for the empty location in the left subtree and insert the data. Otherwise, search for the empty location in the right subtree and insert the data.