Data Structures and Algorithms

Binary trees and transversals

Amilcar Aponte

amilcar@cct.ie

Today's Plan

- Binary Trees
- Transversals
  - Pre-Order
  - Post-Order
  - In-Order

Trees

- A ***tree*** is a non-linear structure in which elements are organized into a hierarchy
- A tree is comprised of a set of ***nodes*** in which elements are stored and ***edges*** connect one node to another
- Each node is located on a particular ***level***
- There is only one ***root*** node in the tree

Trees

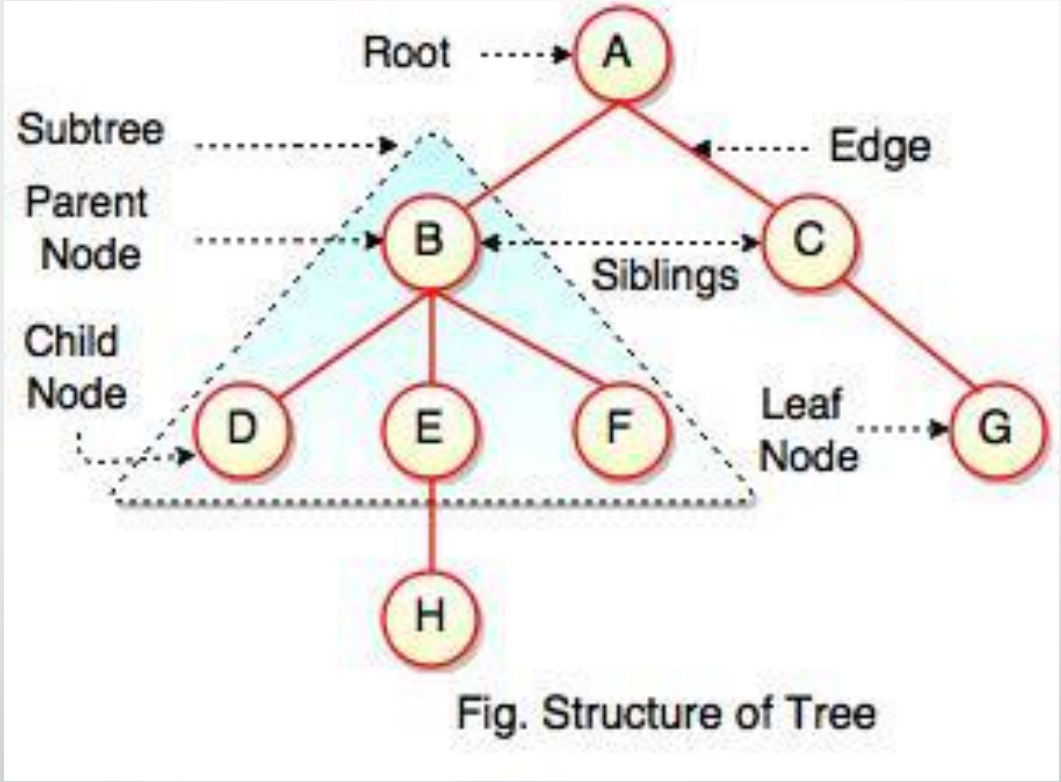


Fig. Structure of Tree

Trees

- Nodes at the lower level of a tree are the ***children*** of nodes at the previous level
- A node can have only one ***parent***, but may have multiple children
- Nodes that have the same parent are ***siblings***
- The root is the only node which has no parent

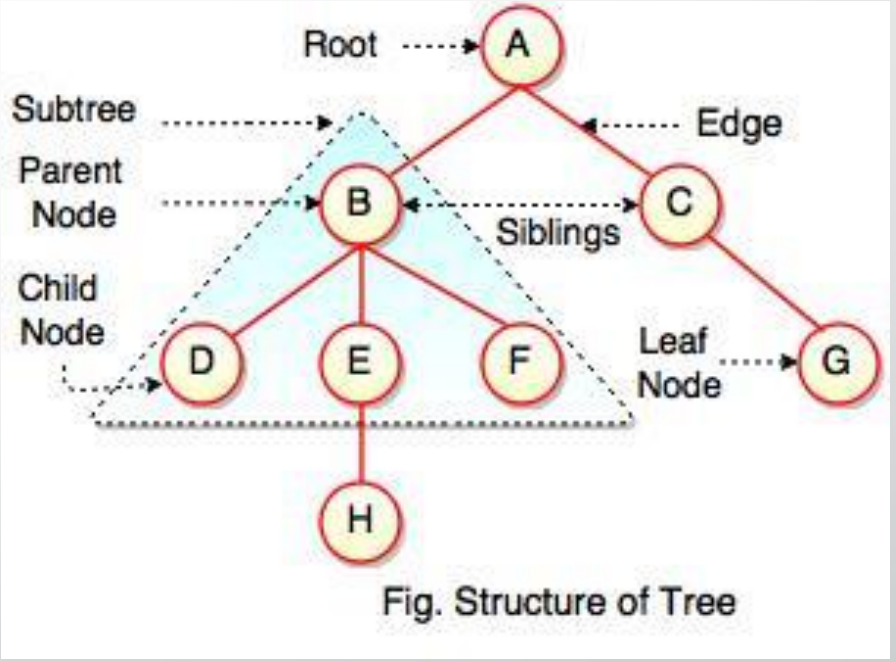
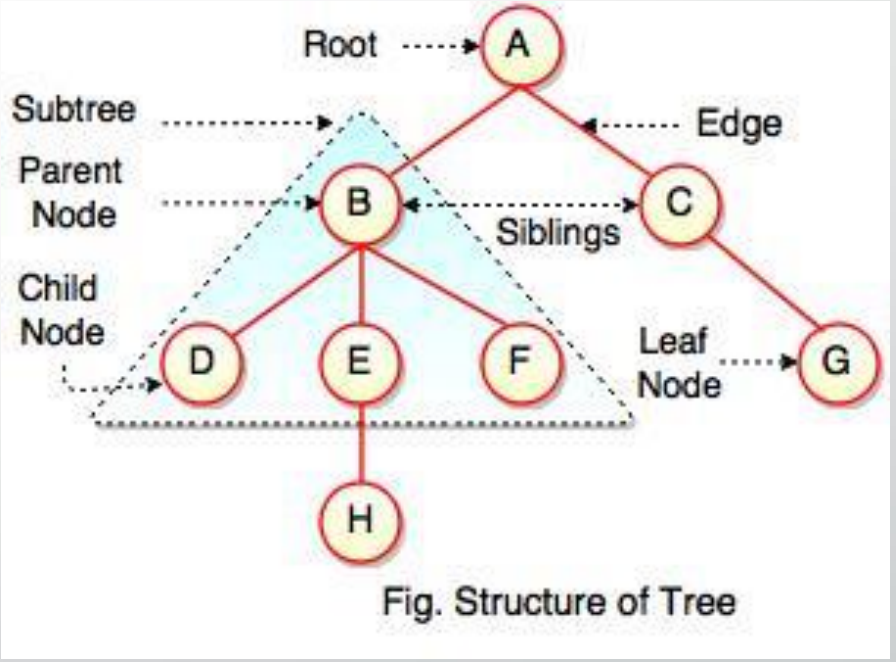


Fig. Structure of Tree

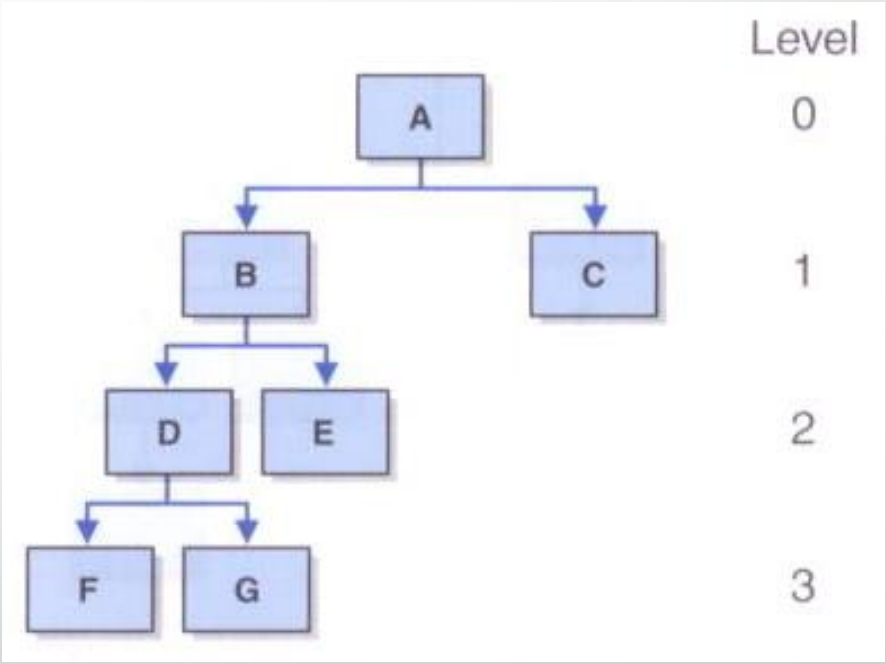
Trees

- A node that has no children is a ***leaf*** node
- A node that is not the root and has at least one child is an ***internal node***
- A ***subtree*** is a tree structure that makes up part of another tree
- We can follow a ***path*** through a tree from parent to child, starting at the root



Trees

- The ***path length*** is the number of edges followed to get from the root to the node
- The ***level*** of a node is the length of the path from the root to the node
- The ***height*** of a tree is the length of the longest path from the root to a leaf



Examples of Tree

- Organization charts
- File systems
- Programming environments

Binary Trees

- Trees in which nodes may have at most two children are called ***binary trees***
- When it comes to coding, we can implement Binary trees using array-based structures, however, this approach is less common.
- What will do instead, and that we'll use a **linked structure** for this.

Let’s code this

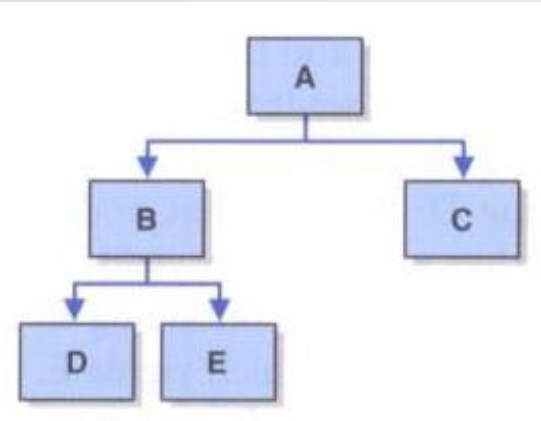
- Let’s try to do a linked tree

Tree Transversals

- Just like linear structures, sometimes we need to iterate or transverse over the whole structure
  - To look for a particular element
  - To apply some changes to some/all of them
- However, unlike linear data structures, which have only one logical way to traverse them, trees can be traversed in different ways.

Tree Traversals

1. **Preorder:** visit the root, then traverse the subtrees from left to right
2. **Inorder:** traverse the left subtree, then visit the root, then traverse the right subtree
3. **Postorder:** traverse the subtrees from left to right, then visit the root



Preorder: A B D E C

Inorder: D B E A C

Postorder: D E B C A

Tree Traversals

1. **Preorder:** visit the root, then traverse the subtrees from left to right

   ROOT LEFT RIGHT

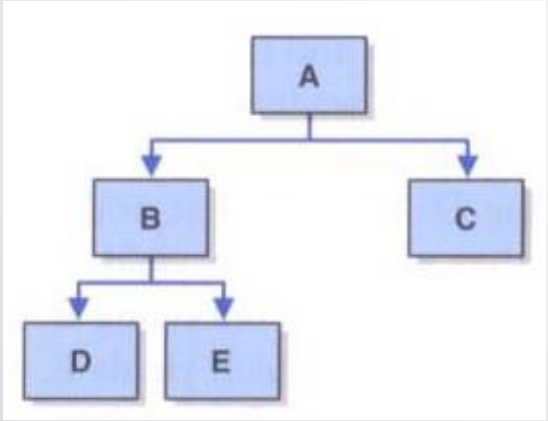
2. **Inorder:** traverse the left subtree, then visit the root, then traverse the right subtree

   LEFT ROOT RIGHT

3. **Postorder:** traverse the subtrees from left to right, then visit the root
4.

   LEFT RIGHT ROOT

Tree Traversals



Preorder: A B D E C

Inorder: D B E A C

Postorder: D E B C A

Tree Traversals in Simplified Form

- Recursion simplifies the implementation of tree traversals

- Preorder:

```
Visit node
Traverse (left child)
Traverse (right child)
```

- Inorder:

```
Traverse (left child)
Visit node
Traverse (right child)
```

- Postorder:

```
Traverse (left child)
Traverse (right child)
Visit node
```

Traversals of a Binary Tree

1. Preorder traversal

- Visit root before we visit root's subtrees

```
Algorithm preOrder(v)
    visit(v)
    for each child w of v
        preorder (w)
```

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)

Traversals of a Binary Tree

2. Inorder traversal

- Visit root of a binary tree between visiting nodes in root's subtrees.

```
Algorithm inOrder(v)
    if left (v) ≠ null
        inOrder (left (v))
    visit(v)
    if right(v) ≠ null
        inOrder (right (v))
```

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)

Traversals of a Binary Tree

3. Postorder traversal

- Visit root of a binary tree after visiting nodes in root's subtrees

```
Algorithm postOrder(v)
    for each child w of v
        postOrder (w)
    visit(v)
```

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.

Let’s try to get this coding

- Let’s do the methods for these three transversals using recursion

That’s all folks

- Any questions?