Fundamental Data Structures & Algorithms using the C language

David Corzo

2020 May 18

Contents

Chapter 1

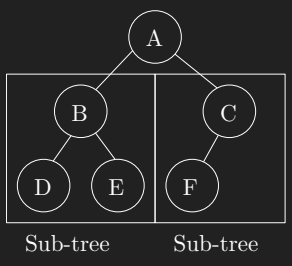
Binary tree and binary search tree

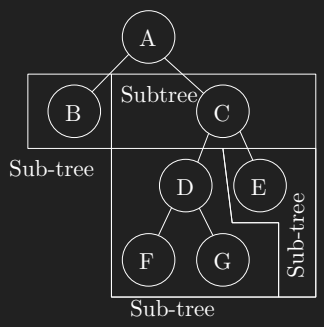
1.1 Introduction to binary tree

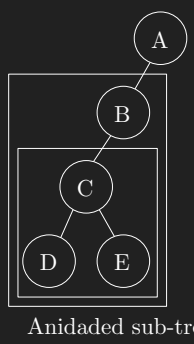
- In a binary search tree, the elements have a hierarchical relationship or *parent-child* relationship. These are not linear data structures, linked lists, arrays, stacks, queues are all linear because they do not have a hierarchy.
- Any tree has hierarchical relationships.

1.1.1 Examples of binary trees

- A tree can have 0,1 or at most two children.
- Each child has 0 or 1 parent.
- The root node is the only node that can have no parents.
- A root can have an empty right tree and an empty left tree.
- The definition of binary trees is recursive, since for instance we can remove node A and be left with two trees by themselves, in this case B would be the root for the left tree, and C would be the root for the right tree.

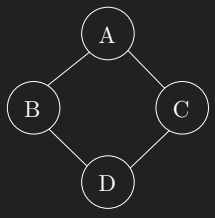






Anidaded sub-tree.

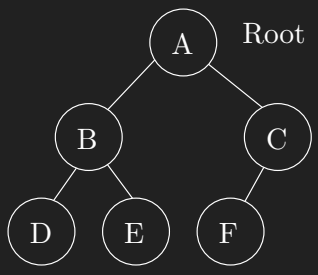
A sub-tree in itself.



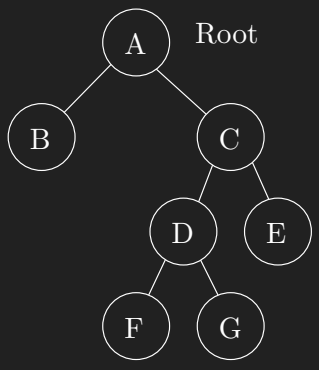
Node D has two parents. We can also see it's not a tree because there is no hierarchy.

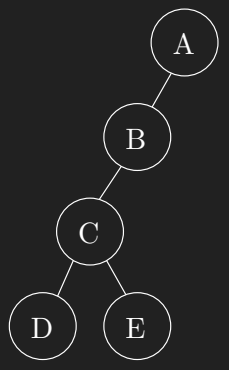
1.1.2 Valid examples and non-valid examples

Examples of valid binary trees



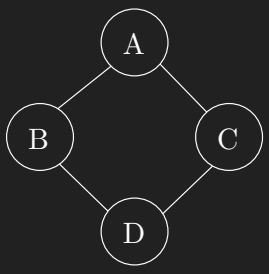
In all examples node A is the root.





We can have a root by itself and this can be considered a tree.

Examples of non binary trees



Node D has two parents. Has no hierarchy. This is actually a graph.

1.2 Formal definition

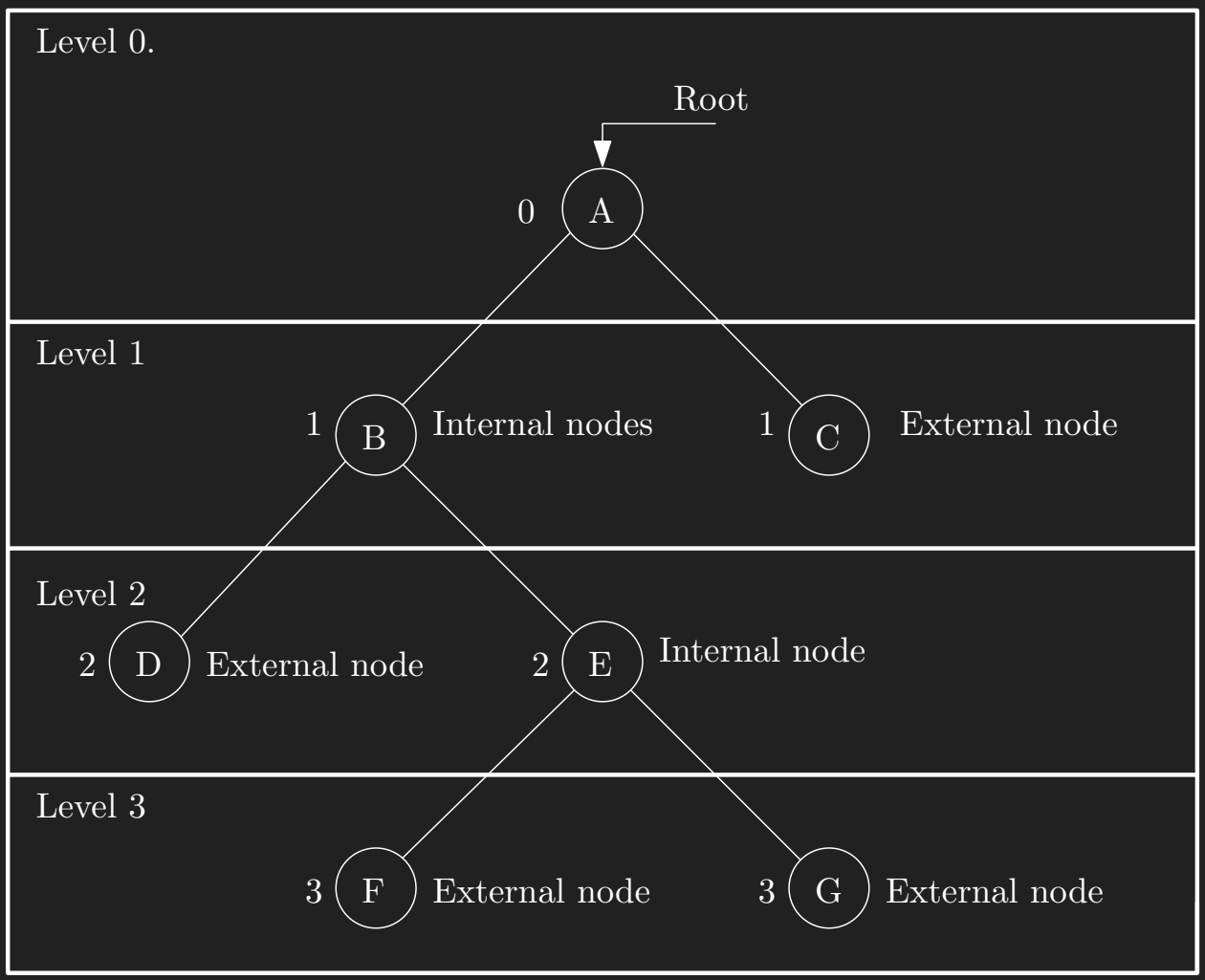
Definición de "Binary tree": *Binary tree is a set of 3* disjoint *subsets, the first one is the root of the tree and the other 2 subsets are either empty or they are themselves binary trees.[1, From Data Structures Using C and C++]*

1.3 Understanding different terminologies related with binary trees

- The root is located at level 0.
- The children of any parent node will have a level equivalent to the level of the parent plus one.
- A leaf node or external node is a node that does not have any children.
- An internal node is a parent node with two children.
- A half-leaf node is a parent node with only one child.
- The height of the binary tree is the level of the leaf that is at the bottom, some books have it as the level of the leafs at the bottom + one.
- Two nodes are considered siblings when they are children of the same parents.

1.3.1 Example

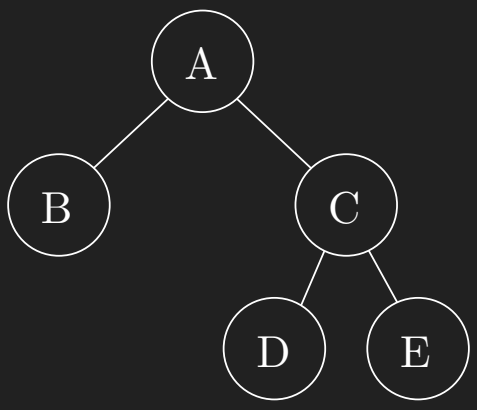
- Root is A.
- C, D, F & G are leaf nodes or external nodes.
- A, B & E are internal nodes.
- There are no half-leafs except for node A.
- The height is 3 (considering start at 0).
- $(B, C), (D, E)$ & (F, G) are siblings because they have the same parent.



1.4 Two tree / strictly binary tree

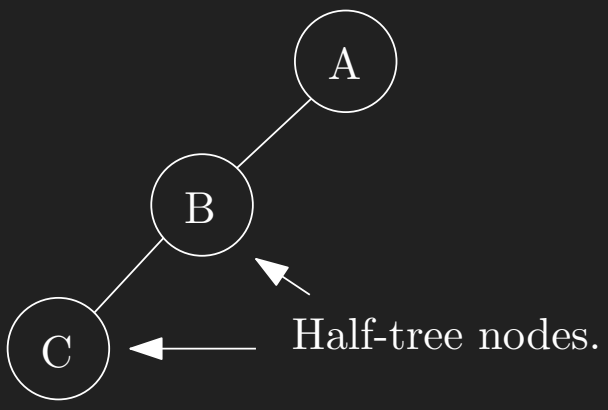
- ***Definición de "Two tree or strictly binary tree":*** *is a binary tree where each node is either a leaf, or they are having both children. Meaning no half-leaf nodes.*

Strictly binary tree.



No half-leaf nodes.

Not a strictly binary tree.

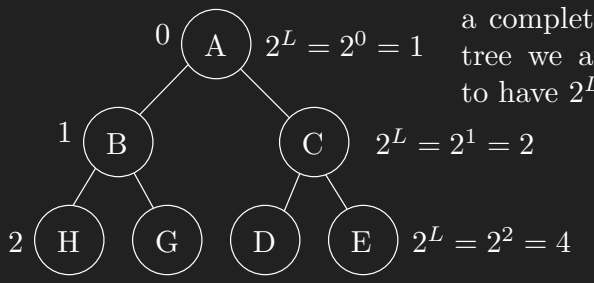


1.5 Complete binary tree / full tree

- ***Definición de "Complete binary tree / full tree ":*** *A complete binary tree is a 2-tree where all leaves must reside at the same level. OR, in a complete binary tree at any leaf k, there are always $2k$ nodes.*

Example of complete binary tree

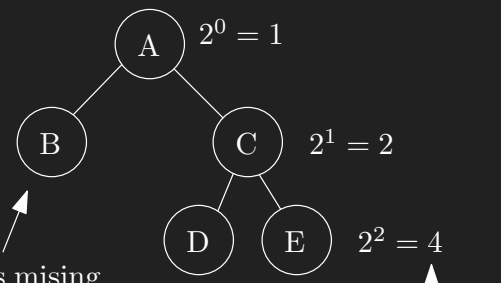
Each level is full.



At any level in a complete binary tree we are going to have 2^L nodes.

Example of incomplete binary tree

Every level is not full.



Two leafs mising in the B node sub-tree.

There are not 4 nodes in level 2.

- The total nodes in a complete binary tree is calculated by:

$$t_n = 2^{h+1} - 1$$

or

$$t_n = 2^0 + 2^1 + 2^2 + 2^3 + \cdots + 2^h$$

Where the h is the height.

  - In the example above is $2^{2+1} - 1 = 8 - 1 = 7$.

- To calculate the number of non-leaves.

$$\text{non-leaves } = 2^h - 1$$

- Total leaves:

$$\text{total-leaves } = 2^h$$

1.6 How to traverse a binary tree

There are three strategies for traversing a binary tree:

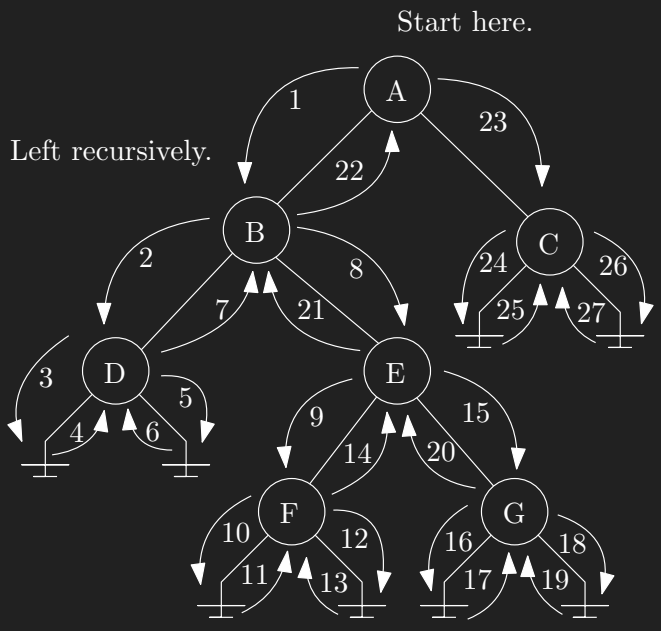
1. In-order traversal.
2. Pre-order traversal.
3. Post-order traversal.

In a binary trees it's impossible to traverse linearly, they are not like arrays or linked lists in the sense where you can do a loop and traverse. Trees are not linear data structures and in order to visit every node of the tree at least once we need to have proper traversal strategies. Whenever we traverse a binary tree we must do it recursively.

1.6.1 In-order traversal strategy

This strategy consists of implementing a recursive algorithm. This algorithm considers every current node as a sub-tree in itself.

1. Traverse the left sub-tree using in-order routine.
2. Access root.
3. Traverse right sub-tree using in order routine.



Order of nodes visited: D at step 4, B at step 7, F at step 11, E at step 14, G at step 17, A at step 22, C at step 25.

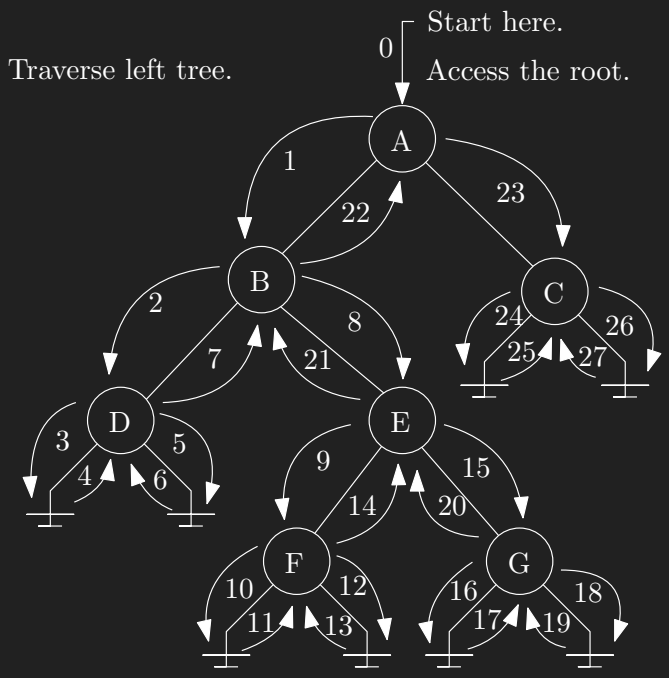
$$D, B, F, E, G, A, C$$

Insight: left $\rightarrow$ root $\rightarrow$ right.

1.6.2 Pre-order traversal strategy

It's different from the in-order because here the root stays constant, this algorithm doesn't consider each node the root.

1. Access the root.
2. Traverse left sub-tree using the pre-order algorithm using recursion.
3. Traverse right-sub-tree using pre-order.



Order of nodes visited: A accessed at step 0, B accessed at step 1, D accessed at step 2, E accessed at step 8, F accessed at step 9, G accessed at step 15, C accessed at step 23.

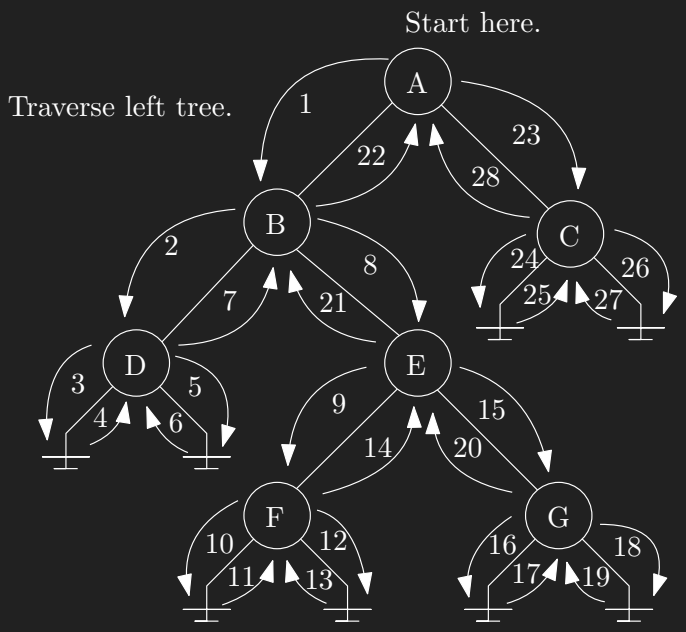
$$A, B, D, E, F, G, C$$

Insight: root $\rightarrow$ left $\rightarrow$ right.

1.6.3 Post-order traversal strategy

In this implementation the root is accessed at the end. The position where we access the root is at the end. This is also recursive.

1. Traverse the left-sub-tree using post-order.
2. Traverse the right-sub-tree using post-order.
3. Access the root.



Order of nodes visited: D accessed at step 6, F accessed at step 13, G accessed at 19, E accessed at step 20, B accessed at step 21, C accessed at 27, A accessed at step 28.

$$D, F, G, E, B, C, A$$

Insight: left $\rightarrow$ right $\rightarrow$ root.

1.7 Constructing a binary tree from a given traversal list

- In-order traversal list: $D, B, F, E, G, A, H, C, I$
  - Because we know that the root is A we know that everything on the right of A will be the left-sub-tree and anything on the right will be the right-sub-tree.
- Pre-order traversal list: $A, C, D, E, F, G, C, H, I$
  - As a rule we know that the first letter will always be the root.
  -
- Post-order traversal list: $D, F, G, E, B, H, I, C, A$
  -

Bibliography

[1] Augenstein Tanenbaum Langsam. *Data Structures using C and C++.*