Tree

quarta-feira, 16 de outubro de 2024 13:49

Each node of a tree is an object that contains key attributes and pointers to other nodes

↳ Can vary depending on the type of the tree

- Binary tree

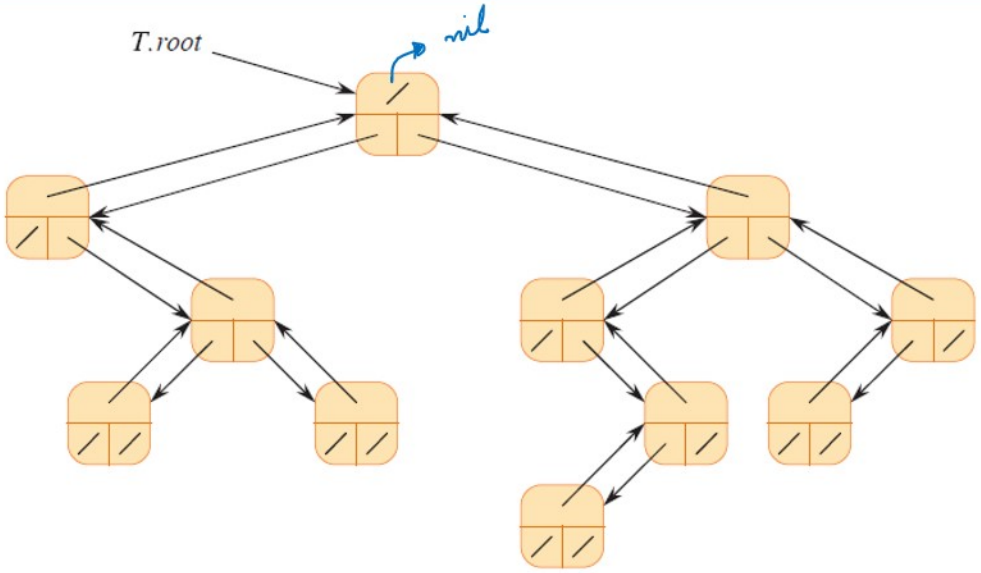


Figure 10.6 The representation of a binary tree T. Each node x has the attributes $x.p$ (top), $x.left$ (lower left), and $x.right$ (lower right). The *key* attributes are not shown.

Store pointers to the parent & childs on each node

Binary trees

Figure 10.6 shows how to use the attributes p, *left*, and *right* to store pointers to the parent, left child, and right child of each node in a binary tree T. If $x.p = \text{NIL}$, then x is the root. If node x has no left child, then $x.left = \text{NIL}$, and similarly for the right child. The root of the entire tree T is pointed to by the attribute $T.root$. If $T.root = \text{NIL}$, then the tree is empty.

- Rooted trees with unbounded branching

↳ Can use the example above to grow the number of childrens

Uses $O(n)$ space for a n-node rooted tree

↳ left-child, right-sibling representations

↳ left-child, right-sibling representation

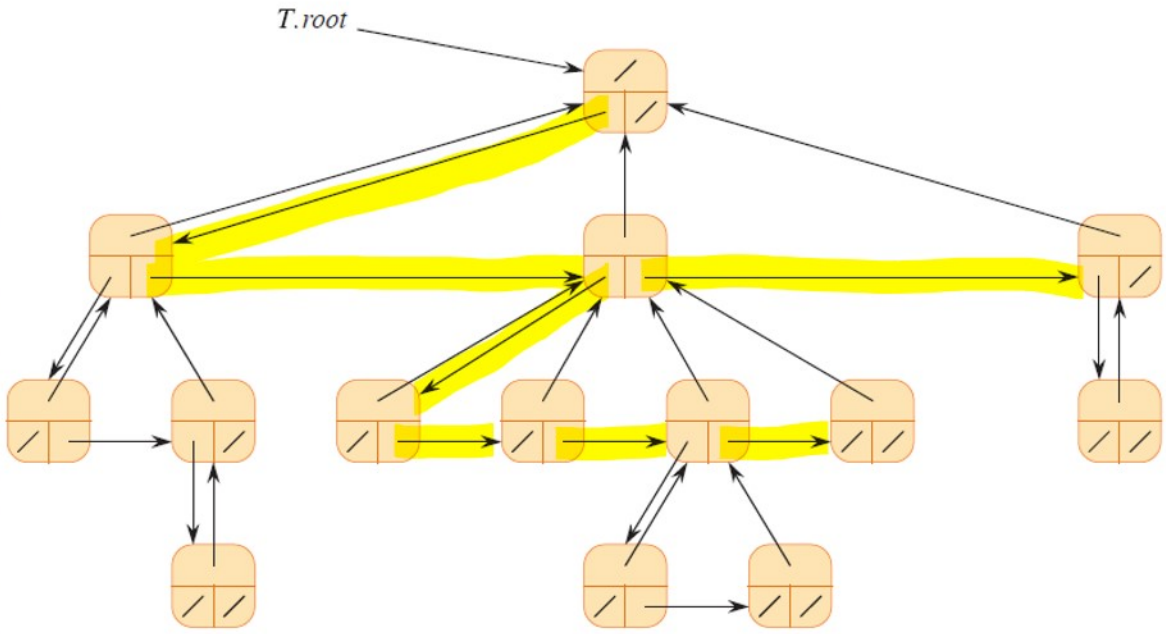


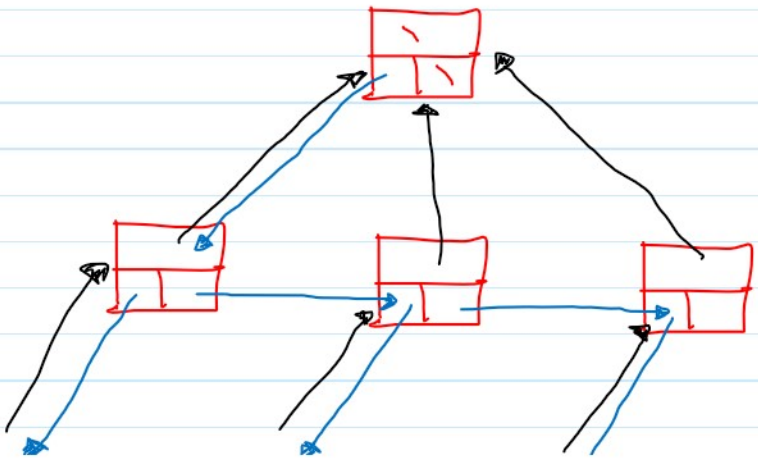
Figure 10.7 The left-child, right-sibling representation of a tree T. Each node x has attributes $x.p$ (top), $x.left\text{-}child$ (lower left), and $x.right\text{-}sibling$ (lower right). The *key* attributes are not shown.

↳ Each node has a parent pointer, pointing to the root

↳ Each node x has only 2 pointers (instead of pointers to each children)

- ↳ x.left-child → left child
- ↳ x.right-sibling → Points to the first soon (left one)

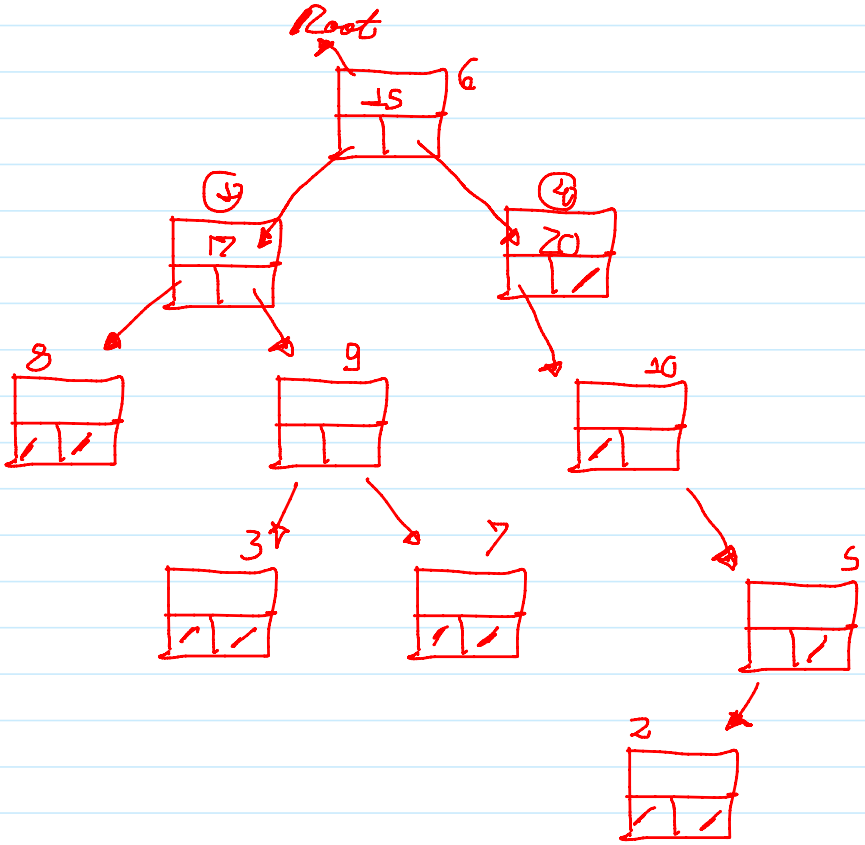
↳ But all the parents points to its fathers



10.3-1

Draw the binary tree rooted at index 6 that is represented by the following attributes:

index	*key*	*left*	*right*
~~1~~	17	8	9
~~2~~	14	NIL	NIL
~~3~~	12	NIL	NIL
~~4~~	20	10	NIL
~~5~~	33	2	NIL
~~6~~	15	1	4
~~7~~	28	NIL	NIL
~~8~~	22	NIL	NIL
~~9~~	13	3	7
~~10~~	25	NIL	5

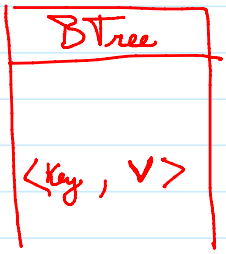


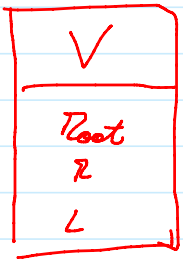
→ Represent a node in a Btree

Class

↳ value
↳ pointer to the root
↳ pointer to the right child
↳ pointer to the left child

- Empty it —
- get the value —
- define values





→ How to Implement

B tree

↳ Each node has: (at most)
- it's value & 2 children
- Nodes at the last level is called leaf

↳ Represented in a class called node
data
pointers to { Right, Left }

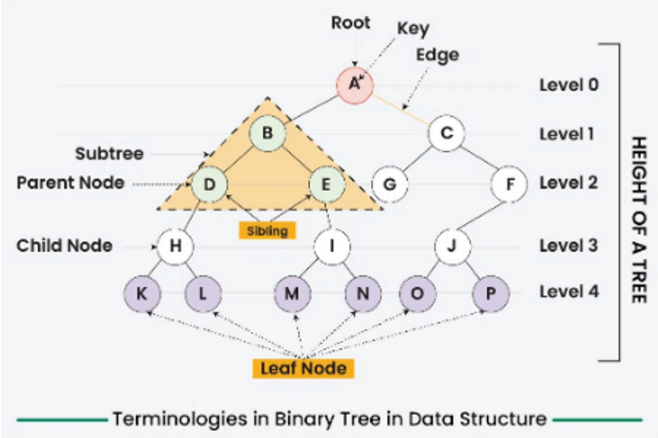
```
template <typename T>
class Node {
public:
    T data;
    Node* left;
    Node* right;
```

→ Declare a Binary tree class to encapsulate the data members

Terminologies in Binary Tree

- **Nodes:** The fundamental part of a binary tree, where each node contains **data** and **link** to two child nodes.
- **Root**: The topmost node in a tree is known as the root node. It has no parent and serves as the starting point for all nodes in the tree.
- **Parent Node**: A node that has one or more child nodes. In a binary tree, each node can have at most two children.
- **Child Node**: A node that is a descendant of another node (its parent).
- **Leaf Node**: A node that does not have any children or both children are null.
- **Internal Node**: A node that has at least one child. This includes all nodes except the **root** and the **leaf** nodes.
- **Depth of a Node**: The number of edges from a specific node to the root node. The depth of the **root** node is zero.
- **Height of a Binary Tree**: The number of nodes from the deepest leaf node to the root node.

The diagram below shows all these terms in a binary tree.



Terminologies in Binary Tree in Data Structure

Basic Operations on Binary Tree in C++

Following are some of the basic operations of binary tree that are required to manipulate its elements:

Operation Name	Description	Time Complexity	Space Complexity
Insertion	Inserts a new node into the binary tree.	O(N)	O(N)
Deletion	Deletes a specific node from the binary tree.	O(N)	O(N)
Searching	Searches for a specific value in the binary tree.	O(N)	O(N)
Traversal	In-order, Pre-order, Post-order traversals	O(N)	O(N)

↳ Reading some more documentations, there is a way to add vertices to the tree without an configured order.

↳ Grows from left to right child indefinetly

Approach: Insert

1. ***Check if the tree is empty***:
   - *If the tree is empty, create a new node and set it as the root.*

2. ***Use level order traversal to find the insertion point***:
   - *Initialize a queue and enqueue the root node.*
   - *While the queue is not empty:*
     - *Dequeue a node from the front of the queue.*
     - *If the dequeued node's left child is `NULL`:*
       - *Insert the new node as the left child of the dequeued node.*
       - *Break the loop.*
     - *If the dequeued node's right child is `NULL`:*
       - *Insert the new node as the right child of the dequeued node.*
       - *Break the loop.*
     - *If both left and right children are not `NULL`, enqueue them.*