VTC Academy
Think Ahead, Beyond Boundaries

DATA STRUCTURES AND ALGORITHMS

LESSON 4

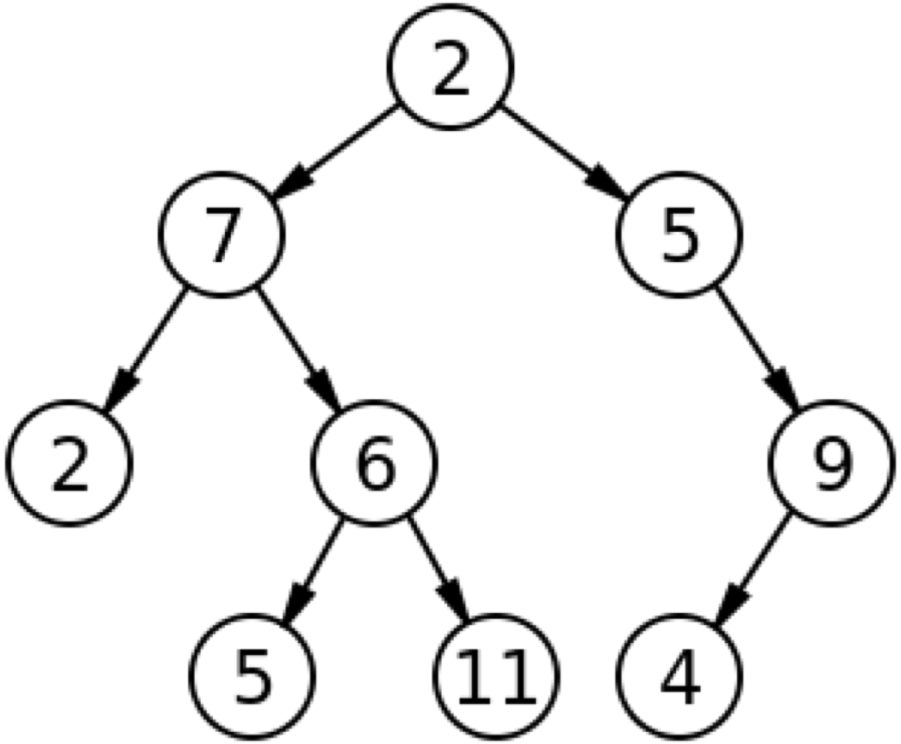
Trees and Binary Search Trees

CONTENT

Introduction to Tree

- A tree is a widely 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 with a parent node, represented as a set of linked nodes.
- A tree data structure can be defined 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 reference is duplicated, and none points to the root.

Introduction to Tree



Introduction to Tree

- Important terms about Trees:
  - Path – Path refers to the sequence of nodes along the edges of a tree.
  - Root – The node at the top of the tree is called root. There is only one root per tree and one path from the root node to any node.
  - Parent – Any node except the root node has one edge upward to a node called parent.
  - Child – The node below a given node connected by its edge downward is called its child node.
  - Leaf – The node which does not have any child node is called the leaf node.

Introduction to Tree

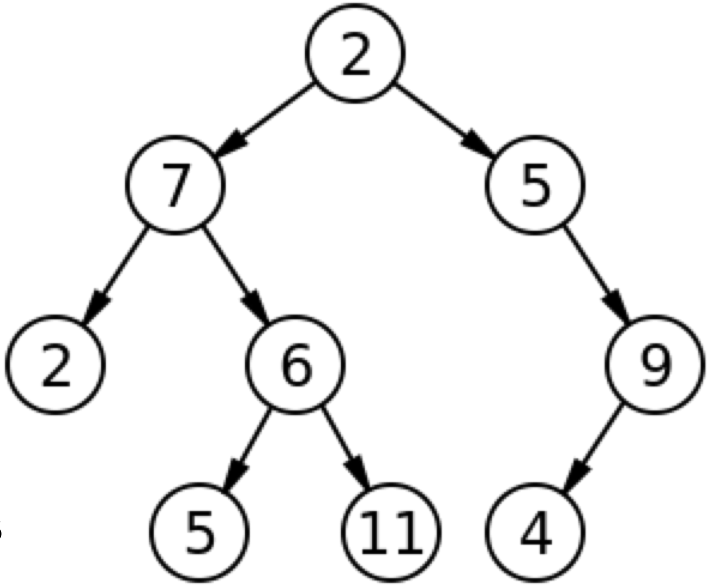
- Subtree – Subtree represents the descendants of a node.
- Visiting – Visiting refers to checking the value of a node when control is on the node.
- Traversing – Traversing means passing through nodes in a specific order.
- Levels – Level of a node represents the generation of a node. If the root node is at level 0, then its next child node is at level 1, its grandchild is at level 2, and so on.
- keys – Key represents a value of a node based on which a search operation is to be carried out for a node.

Types of Tree

- General Tree
- Binary Tree
- Binary Search Tree
- AVL Tree
- Red Black Tree
- Splay Tree
- Treap
- B-Tree

Binary Tree

- A binary tree is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child.
- A recursive definition using just set theory notions is that a (non-empty) binary tree is a triple (L, S, R), where L and R are binary trees or the empty set and S is a singleton set.
- Some authors allow the binary tree to be the empty set as well.



Types of Binary Tree

- **Full Binary Tree:** A Binary Tree is a full binary tree if every node has 0 or 2 children.
- The following are the examples of a full binary tree. We can also say a full binary tree is a binary tree in which all nodes except leaf nodes have two children.

```
               18
           /       \
         15         30
       /  \        /  \
     40    50   100    40
```

```
             18
           /    \
         15      20
       /  \
     40    50
   /  \
 30    50
```

Types of Binary Tree

- **Complete Binary Tree:** A Binary Tree is a Complete Binary Tree if all the levels are completely filled except possibly the last level and the last level has all keys as left as possible
- The following are examples of Complete Binary Trees:

```
           18
       /        \
     15         30
    /  \         /  \
  40    50    100   40
```

```
           18
       /        \
     15         30
    /  \         /  \
  40    50    100   40
 /  \   /
8   7  9
```

Types of Binary Tree

- **Perfect Binary Tree:** A Binary tree is a Perfect Binary Tree in which all the internal nodes have two children and all leaf nodes are at the same level.
- The following are the examples of Perfect Binary Trees.

```
               18
           /         \
         15           30
        /  \         /  \
      40    50     100    40
```

```
               18
           /         \
         15           30
```

Binary Search Trees

- Binary Search Trees (BST), sometimes called ordered or sorted binary trees, are a particular type of container: data structures that store "items" (such as numbers, names etc.) in memory.
- They allow fast lookup, addition and removal of items, and can be used to implement either dynamic sets of items, or lookup tables that allow finding an item by its key (e.g., finding the phone number of a person by name).

Binary Search Trees

- Binary search trees keep their keys in sorted order, so that lookup and other operations can use the principle of binary search: when looking for a key in a tree (or a place to insert a new key), they traverse the tree from root to leaf, making comparisons to keys stored in the nodes of the tree and deciding, on the basis of the comparison, to continue searching in the left or right subtrees.

Binary Search Tree Definition

- A binary search tree is a rooted binary tree, whose internal nodes each store a key (and optionally, an associated value) and each have two distinguished sub-trees, commonly denoted left and right.
- The tree additionally satisfies the binary search tree property, which states that the key in each node must be greater than or equal to any key stored in the left sub-tree, and less than or equal to any key stored in the right sub-tree.

Binary Search Tree Definition

- The information represented by each node is a record rather than a single data element.
- The major advantage of binary search trees over other data structures is that the related sorting algorithms and search algorithms such as in-order traversal can be very efficient; they are also easy to code.

Binary Search Tree Definition

- Tree Node:

```
struct node
{
    int value;
    struct node *left;
    struct node *right;
};
```

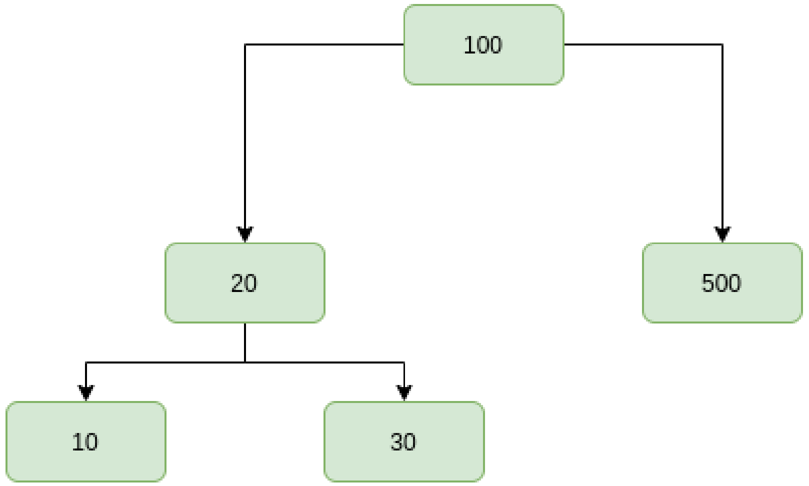
- Declare Binary Tree

```
struct node *root;
root = NULL;
```

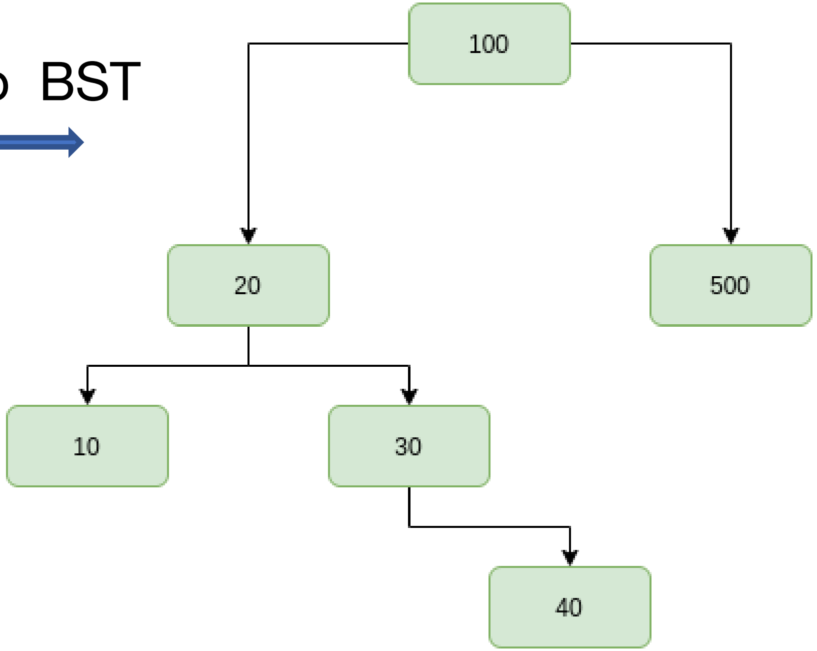
BST Basic Operations

- Insert – Inserts an element in a tree/create a tree.
- Preorder Traversal – Traverses a tree in a pre-order manner.
- Search – Searches an element in a tree.
- Delete – Delete an element in a tree.

BST Insert / Create Operation



Insert 40 to BST

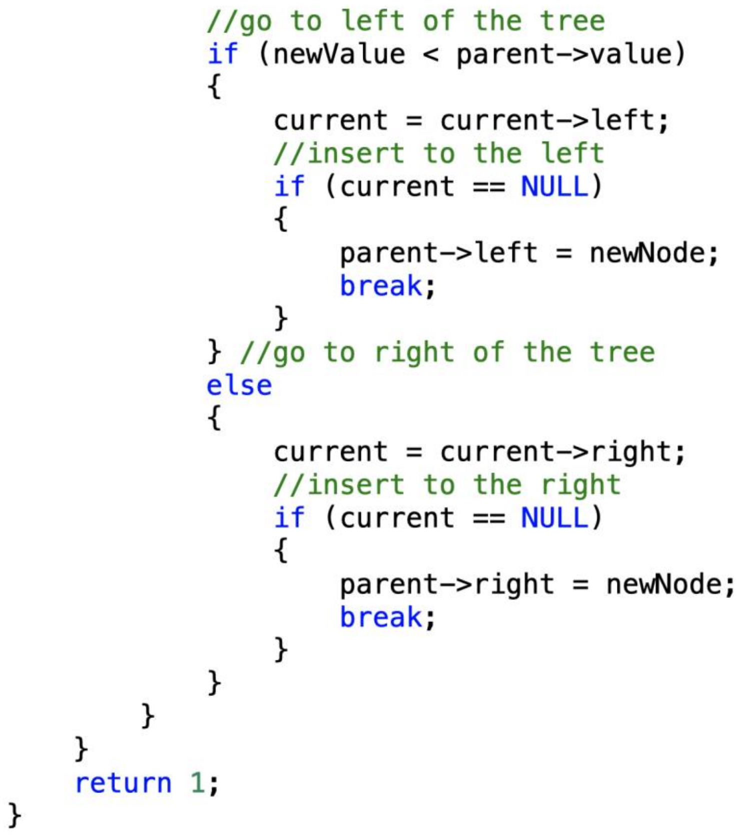


BST Insert / Create Operation

```
int insert(struct node **root, int newValue)
{
    struct node *newNode;
    struct node *current;
    struct node *parent;
    newNode = (struct node *)malloc(sizeof(struct node));
    newNode->value = newValue;
    newNode->left = NULL;
    newNode->right = NULL;

    if (*root == NULL)
    {
        *root = newNode;
    }
    else
    {
        current = *root;
        parent = NULL;
        while (1)
        {
            parent = current;
```

BST Insert / Create Operation



```
            //go to left of the tree
            if (newValue < parent->value)
            {
                current = current->left;
                //insert to the left
                if (current == NULL)
                {
                    parent->left = newNode;
                    break;
                }
            } //go to right of the tree
            else
            {
                current = current->right;
                //insert to the right
                if (current == NULL)
                {
                    parent->right = newNode;
                    break;
                }
            }
        }
    }
    return 1;
}
```

BST Insert/Create Operation using Recursion

```
int recursiveInsert(struct node **root, int newValue)
{
    struct node *newNode;
    newNode = (struct node *)malloc(sizeof(struct node));
    newNode->value = newValue;
    newNode->left = NULL;
    newNode->right = NULL;

    if (*root == NULL)
    {
        *root = newNode;
    }
    else
    {
        if (newValue < (*root)->value)
        {
            insert(&(*root)->left, newValue);
        }
        else
        {
            insert(&(*root)->right, newValue);
        }
    }
    return 1;
}
```

BST Traversal Operation

```
void printInorderTraversal(struct node *root)
{
    if (root == NULL)
    {
        return;
    }
    printInorderTraversal(root->left);
    printf("value: %d\n", root->value);
    printInorderTraversal(root->right);
}
```

BST Search Operation

```c
struct node *search(struct node *root, int data)
{
    struct node *current = root;
    while (current != NULL && current->value != data)
    {
        //go to left tree
        if (current->value > data)
        {
            current = current->left;
        } //else go to right tree
        else
        {
            current = current->right;
        }
    }
    return current;
}
```

BST Search Operation using Recursion

```
struct node *recursiveSearch(struct node *root, int data)
{
    if (root == NULL || data == root->value)
    {
        return root;
    }
    else if (data < root->value)
    {
        return recursiveSearch(root->left, data);
    }
    else
    {
        return recursiveSearch(root->right, data);
    }
}
```

BST Delete Operation

```
struct node *findMin(struct node *root){
    if (root == NULL || root->left == NULL){
        return root;
    }else{
        return findMin(root->left);
    }
}
struct node *delete (struct node *root, int deleteValue){
    struct node *temp;
    if (root == NULL){
        return NULL;
    } else if (deleteValue < root->value) {
        root->left = delete (root->left, deleteValue);
    }else if (deleteValue > root->value){
        root->right = delete (root->right, deleteValue);
    }else {
          if (root->left == NULL){
            temp = root->right;
            root->right = NULL;
            free(root);
            return temp;
          }
```

BST Delete Operation

```
        else if (root->right == NULL){
            temp = root->left;
            root->left = NULL;
            free(root);
            return temp;
        }else{
            temp = findMin(root->right);
            root->value = temp->value;
            if (temp == root->right){
                free(root->right);
                root->right = NULL;
            }else{
                struct node *parentMin;
                parentMin = root->right;
                while (parentMin->left != temp){
                    parentMin = parentMin->left;
                }
                parentMin->left = NULL;
                free(temp);
            }
        }
    }
    return root;
}
```

Summary

- A tree is a widely 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 with a parent node, represented as a set of linked nodes.
- A binary tree is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child.
- Binary Search Trees (BST), sometimes called ordered or sorted binary trees, are a particular type of container: data structures that store "items”.
- BST supports searching, adding and deleting elements in binary trees easily.

VTC Academy
Think Ahead, Beyond Boundaries

Thank you!