

Binary Trees

Objectives

- Build a binary tree
 - Search a binary tree
 - Find minimum value in a binary tree
 - Find the maximum depth of a binary tree
-

In this recitation, you will be learning basic manipulations of binary search trees.

Trees

Trees can have multiple options (children) to go from one node to another in a structure.

Binary Trees

Having two options (at most) for going to a next node from the current node is a feature of a structure called a binary tree.

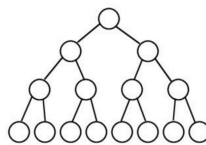
Binary trees are similar to linked lists in that the nodes in the tree can be created dynamically and then linked together to create a structure that can be easily modified to support dynamic data. The next and previous pointers of a doubly linked list are replaced with left and right child pointers in binary trees to make it possible to represent a hierarchical structure in a data set, which is one advantage that trees have over linked lists. Trees can also be searched and modified with minimal computational effort. These advantages mean that binary trees are extremely useful and frequently used over other data structures.

Pointers in a doubly linked list:

- next
- previous

Pointers in a binary tree:

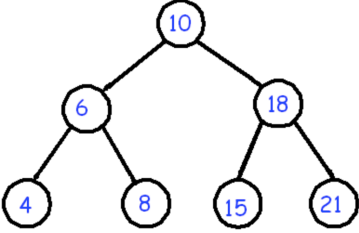
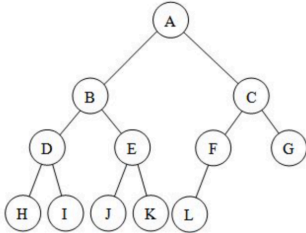
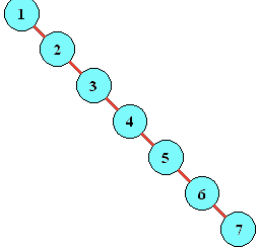
- left child
- right child



Binary Search Trees

A binary search tree (BST) is a special case of a binary tree where the data in the tree is ordered. For any node in the tree, the nodes in the left sub-tree of that node all have a value less than the node value, and the nodes in the right sub-tree of that node all have a value greater than or equal to the node value.

Let x and y be nodes in a binary search tree. If y is in the left sub-tree of x , then $y.key < x.key$. If y is in the right sub-tree of x , then $y.key \geq x.key$.

Full Binary Search Tree	Complete BST	Degenerate BST
 <pre> graph TD 10((10)) --> 6((6)) 10 --> 18((18)) 6 --> 4((4)) 6 --> 8((8)) 18 --> 15((15)) 18 --> 21((21)) </pre>	 <pre> graph TD A((A)) --> B((B)) A --> C((C)) B --> D((D)) B --> E((E)) C --> F((F)) C --> G((G)) D --> H((H)) D --> I((I)) E --> J((J)) E --> K((K)) F --> L((L)) </pre>	 <pre> graph TD 1((1)) --> 2((2)) 2 --> 3((3)) 3 --> 4((4)) 4 --> 5((5)) 5 --> 6((6)) 6 --> 7((7)) </pre>

Using the following struct, create a program with the listed functions below.

```

struct Node
{
    int data;
    Node *left;
    Node *right;
};

```

Functions:

1. **Node * addNode(Node *tree, int value)**
Insert the new value into the BST in the correct position.
2. **int getSize(Node *tree)**
Return the total number of nodes in the BST.
3. **int getMinValue(Node *tree)**
Return the minimal value in the BST.
4. **int getMaxDepth(Node *tree)**
Return the maximum depth of a path in the tree.

To get credit for this lab exercise, show the TA and sign the lab's completion log.