Binary Tree I

DM2233 ADVANCED DATA STRUCTURES & ALGORITHMS

Module Schedule

Week	Lecture	Remarks
1	Overloading and Templates I	
2	Overloading and Templates II	Labour Day (Fri) – Lab 2 Make up on 27-Apr
3	Overloading and Templates III	
4	Overloading and Templates IV	
5	Exception Handling I	
6	Exception Handling II	
7	Standard Template Library / Assignment 1	Vesak Day (Mon)
Week 8 and 9: Mid-Sem Break		
10	Sorting and Searching I	
11	Sorting and Searching II	
12	Sorting and Searching III	
13	Binary Tree I	Hari Raya Puasa (Fri)
14	Lab Test	
15	Binary Tree II	
16	Binary Tree III	SG50 Day (Fri)
17	Preprocessing / Assignment 2	National Day (Mon)

E- Learning in Week 13

- Hari Raya Puasa on Friday
- Lecture as per normal
- No lab sessions for both Lab 1 and 2
- Refer to Blackboard to do your practical and submit via Blackboard

Test (Week 14)

- No Lecture in Week 14
- In-Lab Programming Test.
  - Please be punctual.
  - We will start at 8:30am and 9:30am respectively for Wed/Fri sessions.
- Open Book
  - Can refer to internet but no IM or communication with others
  - Still no phone
- 1 hour and 30 minutes
- Scope:
  - Week 1 till Week 12
  - And even your past C++ and Data modules :-)

Introduction

- Recall we have 2 strategies to sort data:
  - Sorting done at the point of insertion
  - Sorting done only when required
- Both has it's pros and cons
- Today we are going to look at a way to sort data at the point of insertion

Introduction

- When data is to be organized, it is the programmer's highest priority to organize it in such a way that any info item can be inserted, deleted and searched **fast**

- We have done many item insert and delete in an array as well as using binary search on an ordered array for fast item retrieval

- Array has limitation : Item insertion in a large array is time consuming

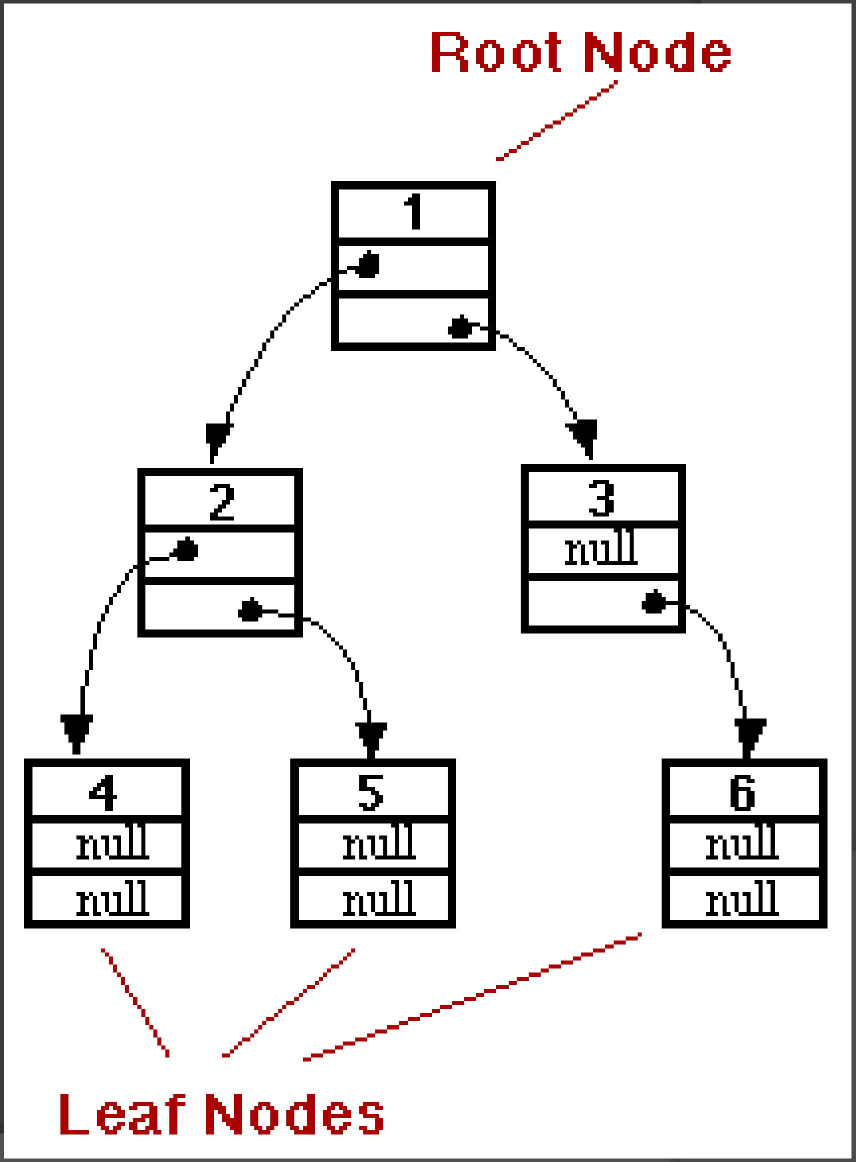
Introduction

- To speed up item insertion and deletion, use linked list with a few manipulation of the pointers

- However, one drawback of using linked list is it has to be processed sequentially; cannot apply binary search

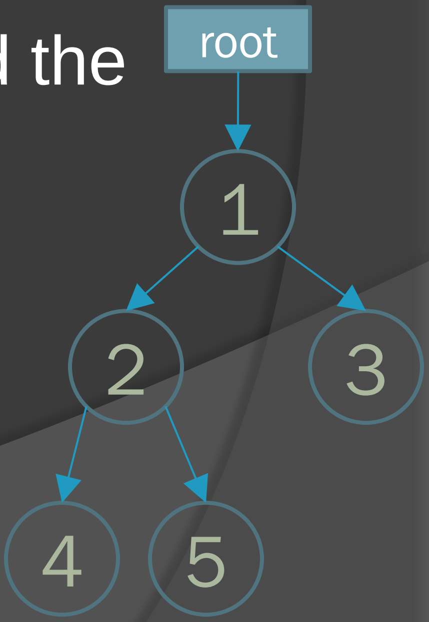
Introduction

- What is a tree?
  - One of the most basic and useful data structure
  - Each of the objects in a binary tree contains two pointers, typically called **left** and **right**



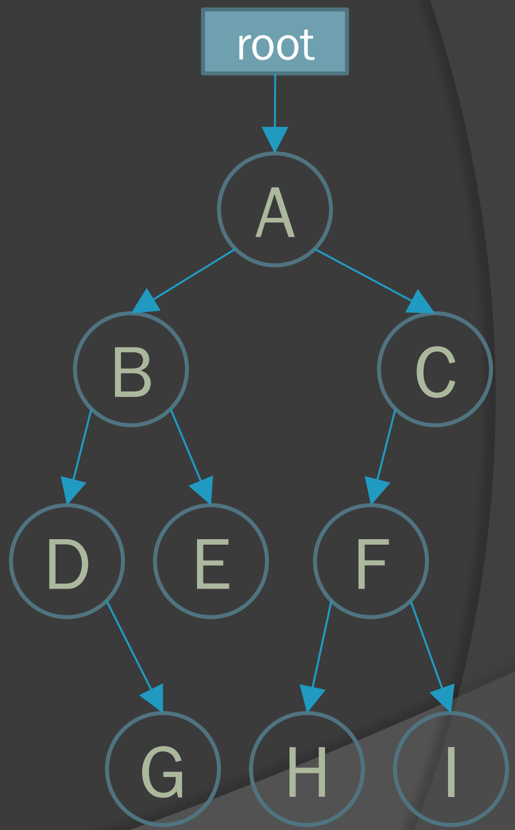
Introduction of Binary Tree

- A binary tree, T, is either empty or such that
  - T has a special node called the root node
  - T has 2 sets of nodes, LT and RT, called the left subtree and right subtree of T, respectively
  - LT and RT are binary trees



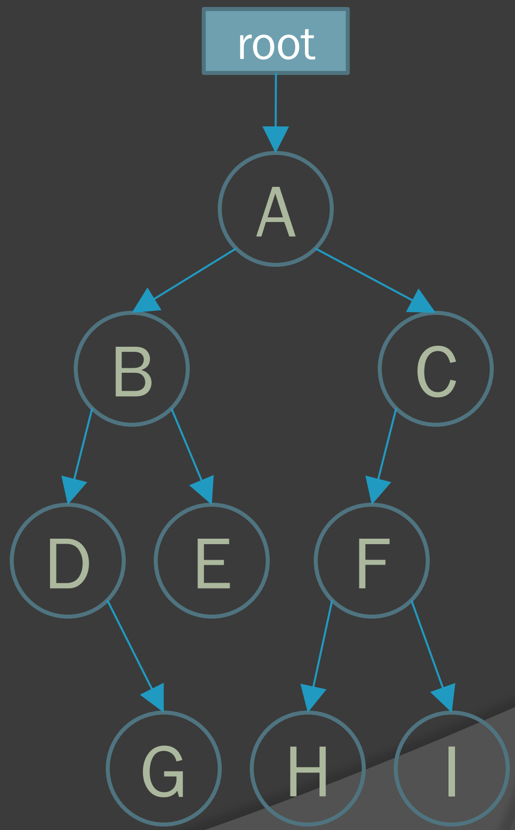
Some terminology

- Root node is A
  - LA and RA are binary trees
- Root node of LA is B
  - LB and RB are binary trees
  - D is called left child of B
  - E is called right child or B
- Which nodes have only 1 child?
- Which nodes are childless?



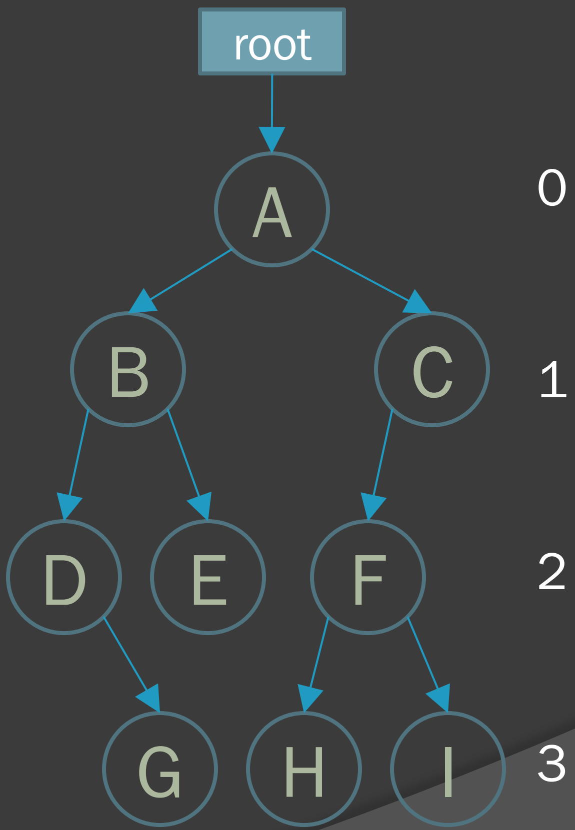
Some terminology

- A node in the binary tree is called a **leaf** if it has no child
- A node is called a **parent** if it has an edge to another node
- A **path** from a node to another is a sequence of nodes
  - E.g. Path from node A to G is A, B, D, G



Some terminology

- The level of a node in a binary tree is the number of branches on the path from the root to the node
- The height of a binary tree is the number of nodes on the longest path from the root to a leaf

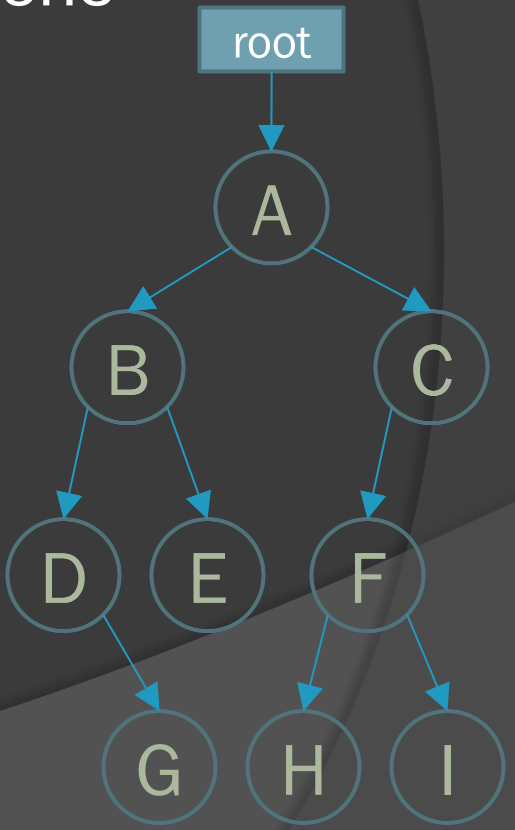


Growing a Tree

1. Insert
   - If data is to be sorted, it has to be done during insertion of the node

2. Traverse the tree

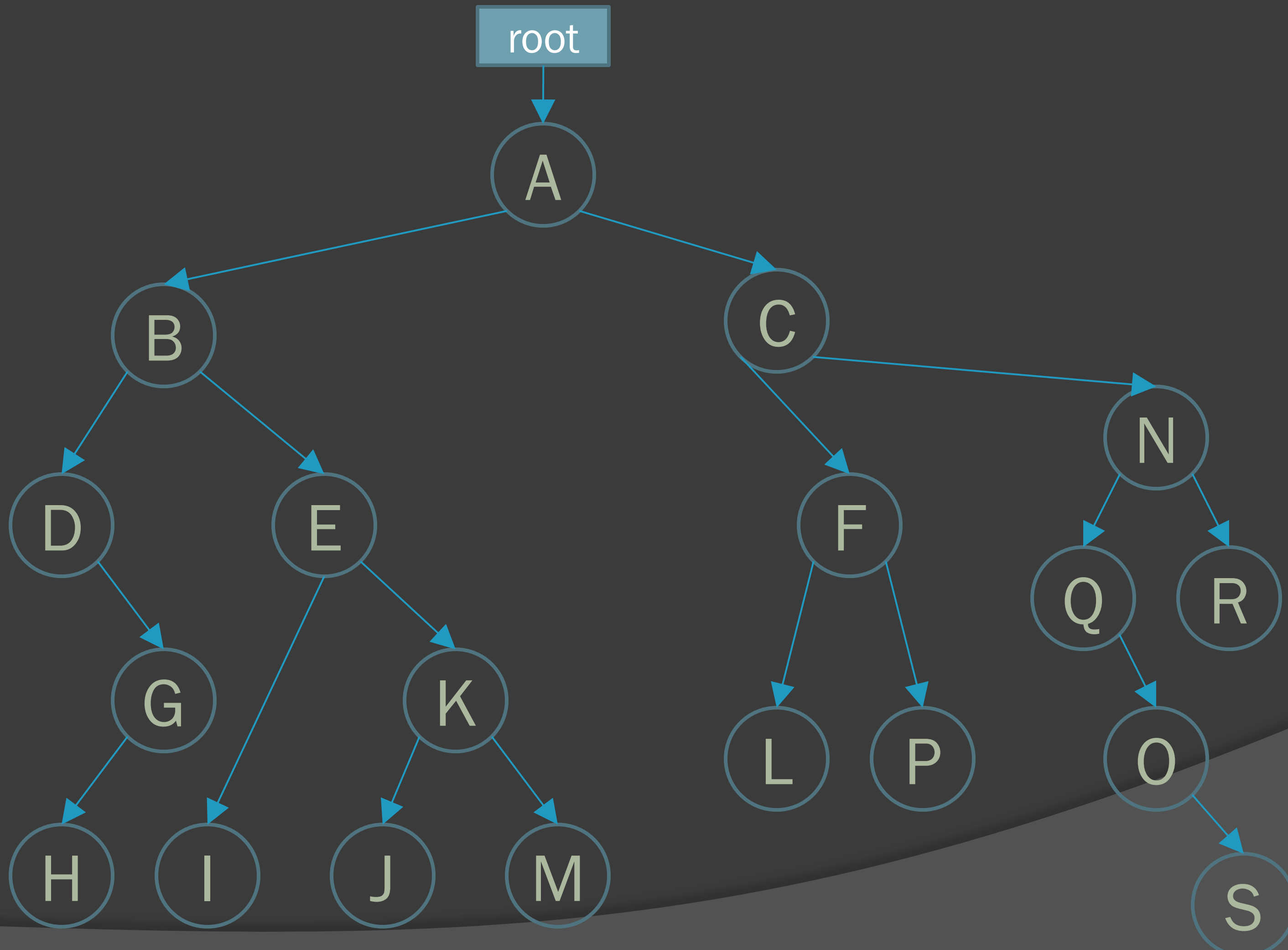
3. Delete a node in the tree



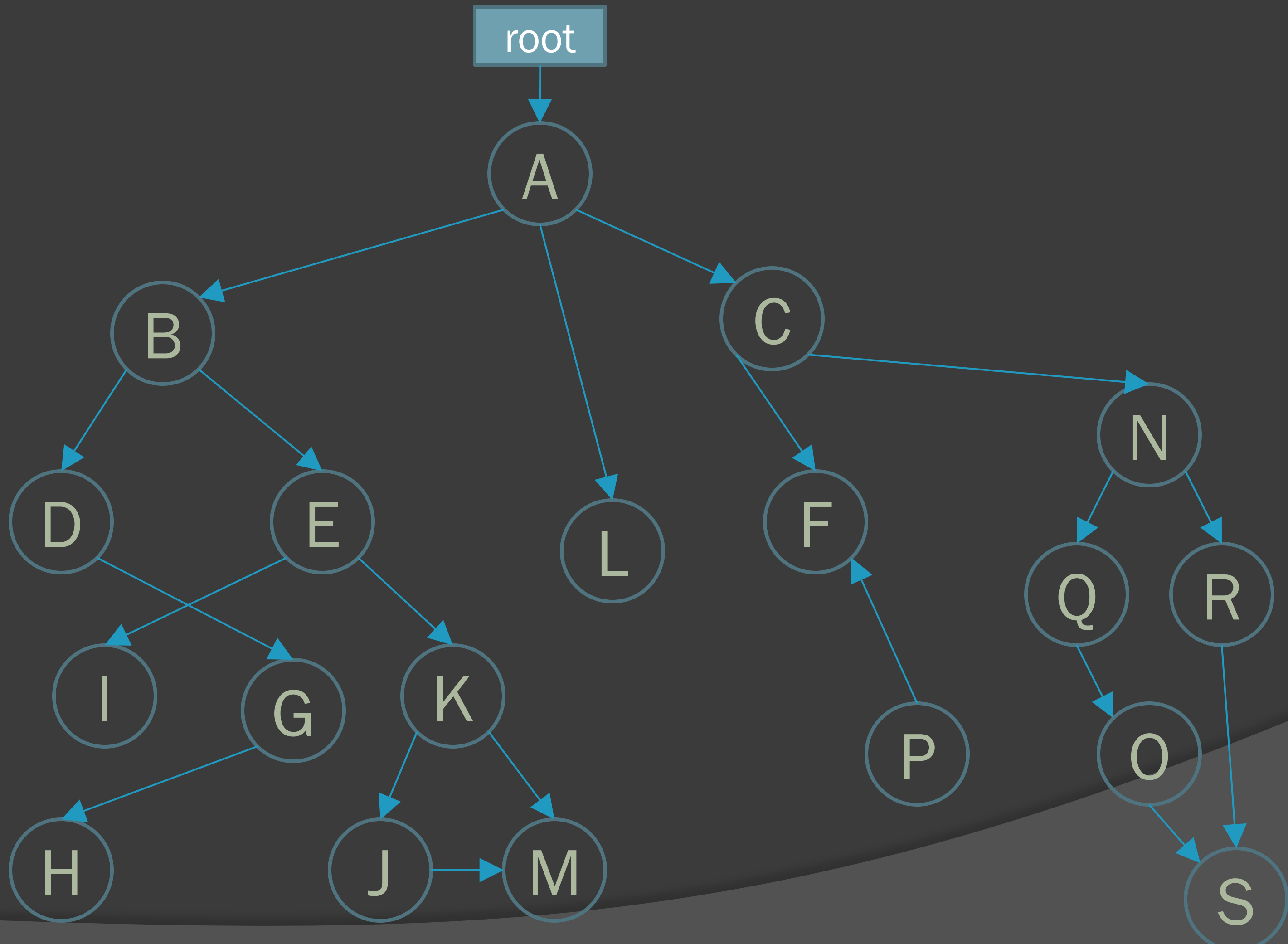
Growing a Tree

- You need 4 pointers to build the tree:
  - One to point to the first node in the tree (**root** pointer)
  - One for traversing (**current** pointer)
  - One to create the new node (**newNode** pointer)
  - One to track previous node (**prev** pointer)

Binary Big Tree Example



Spot the Wrongs



Building a Binary Tree

- Create the new node and initialise it

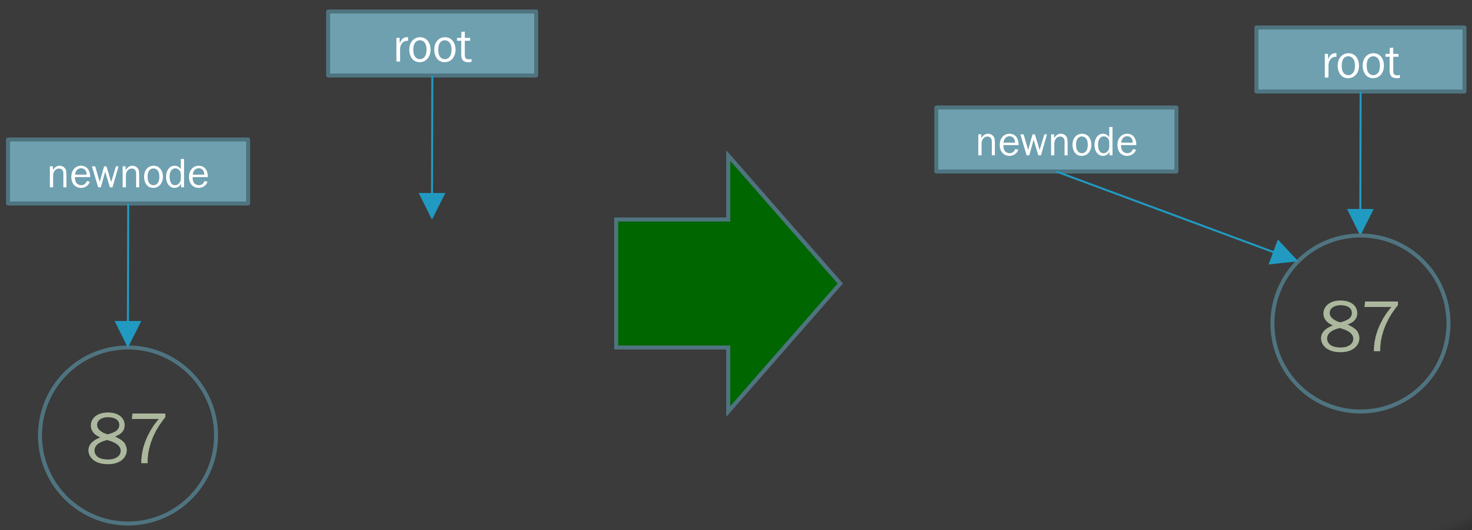
- 2 cases:
  - If root is NULL, append the node to the root
  - Else traversal is required to locate the position to insert the node

- Binary search is employed for the traversal
  - If node value is smaller than current node, traverse to left child
  - Else, traverse to right child

- The resultant is a Binary Search Tree (BST)

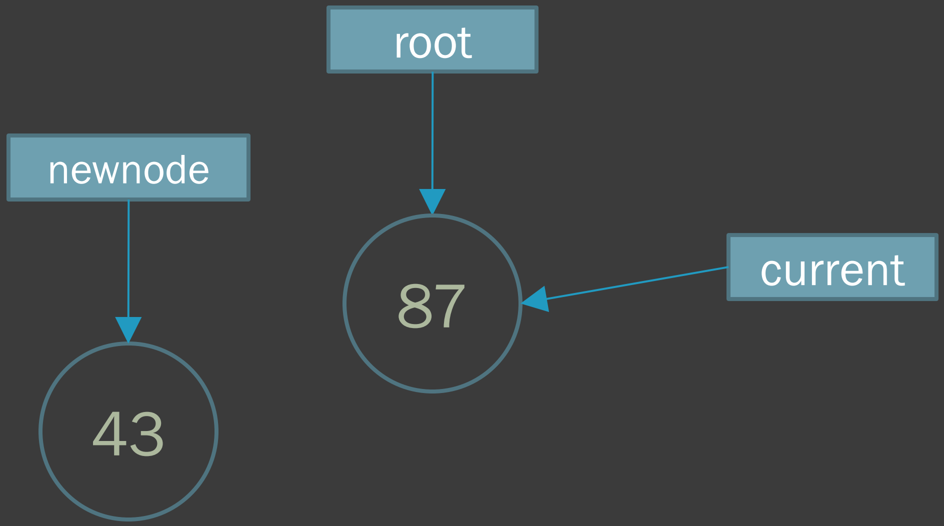
Building a Binary Tree

- Root is NULL (Tree is empty)



Building a Binary Tree

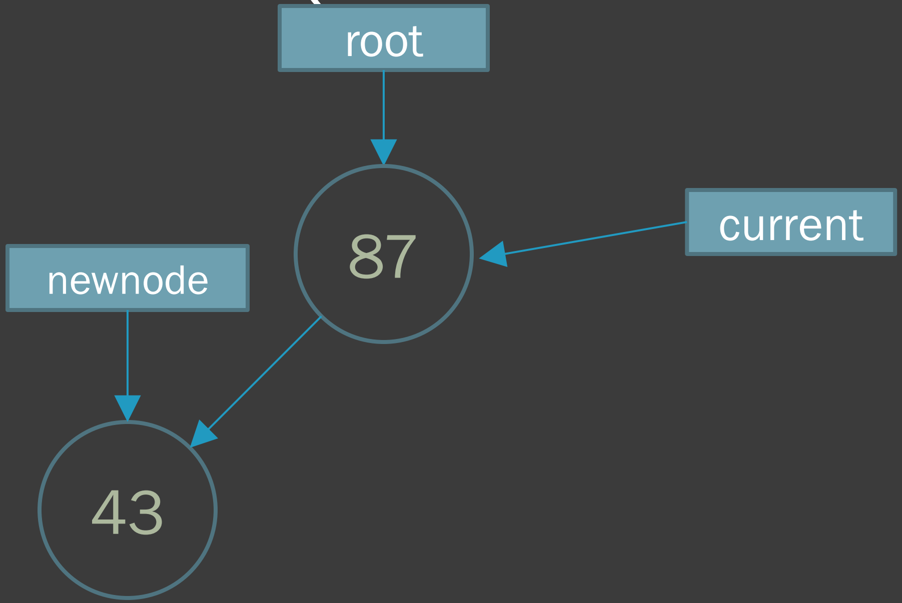
- Root is not NULL (Tree has at least one node)



- Need a pointer: current
  - Init current to start from root

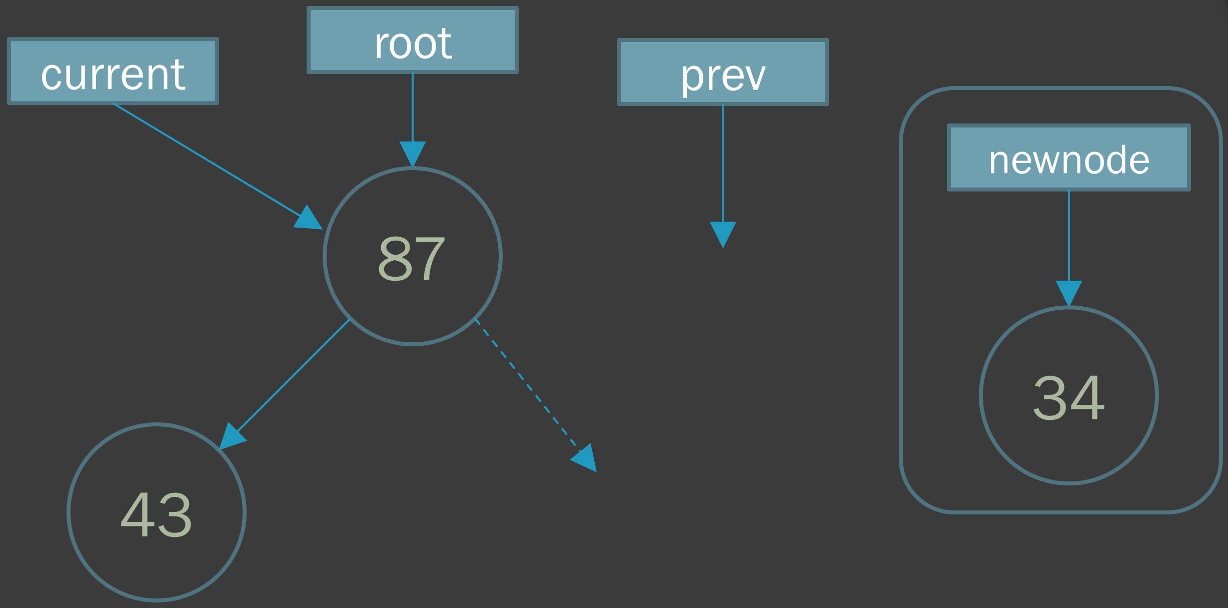
Building a Binary Tree

- Root is not NULL (Tree has at least one node)



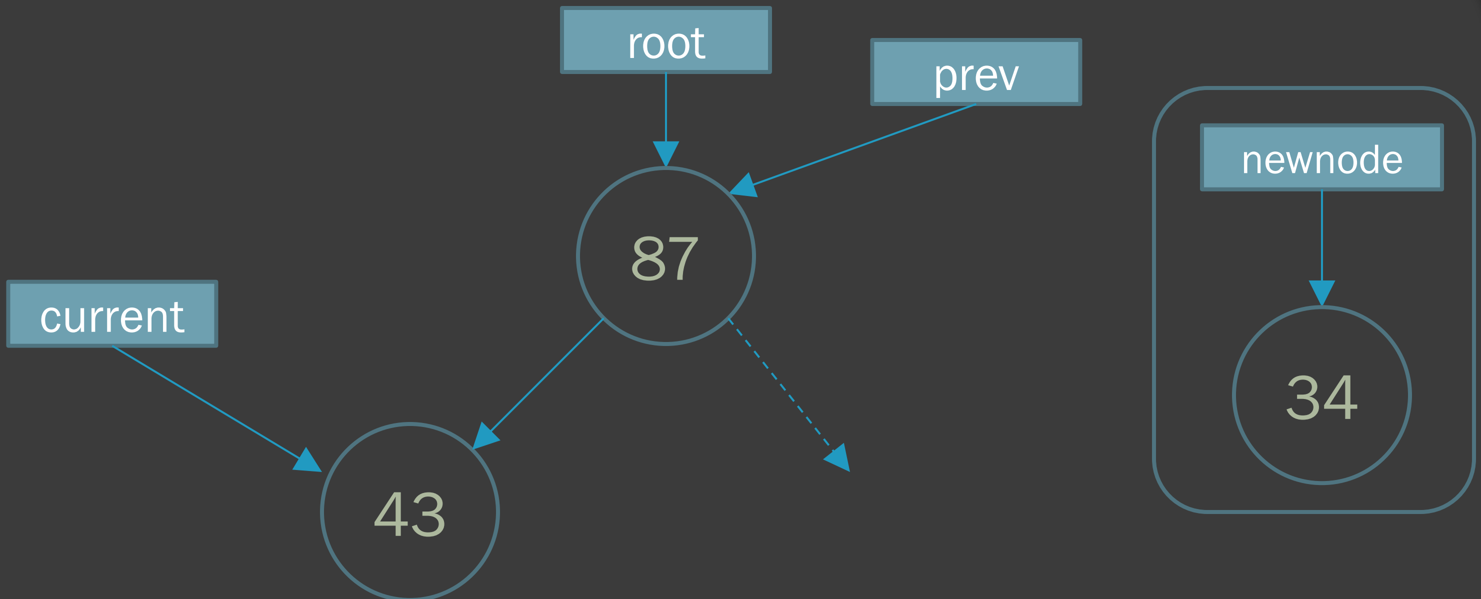
- Check if newnode value is smaller or bigger than the current. Remember if smaller put the newnode to the Left, else put to the right.

Building a Binary Tree



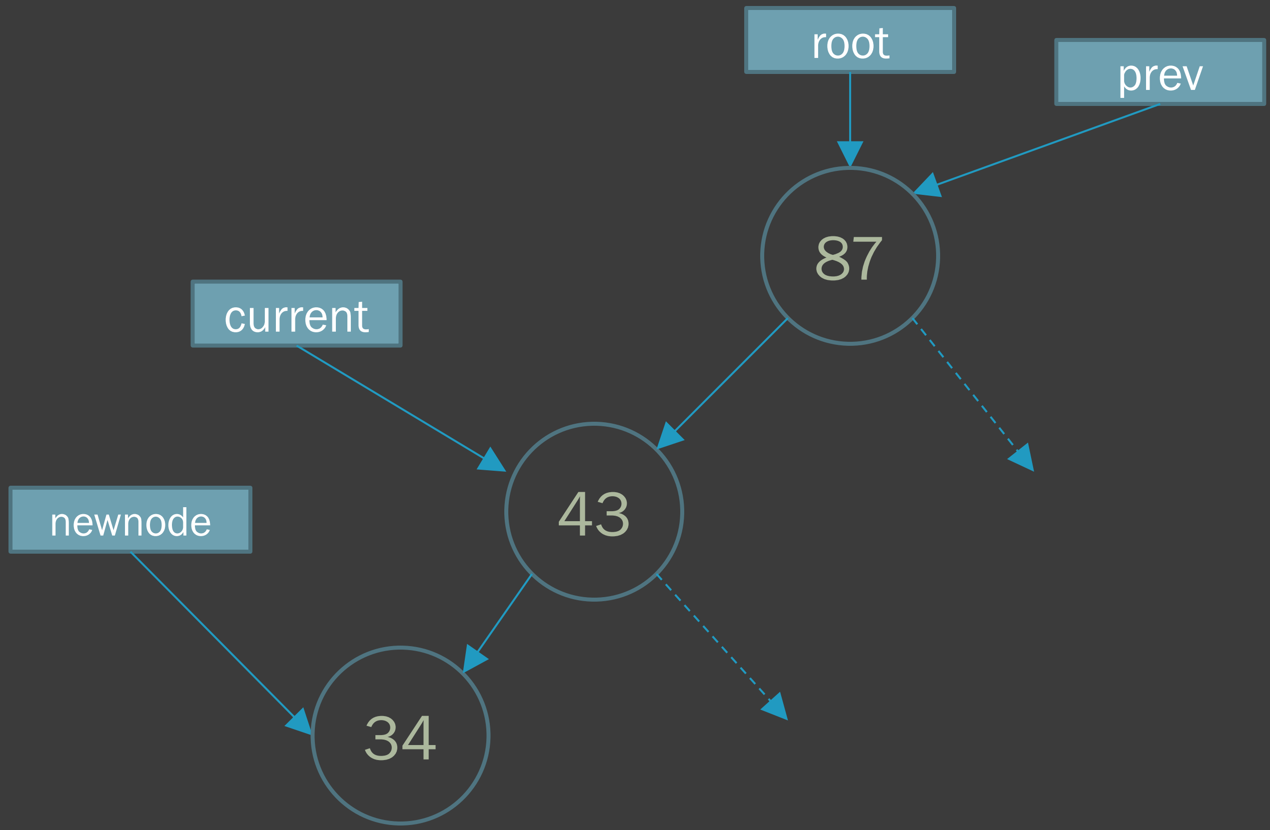
- Need yet another pointer called prev.
- Again start current pointer from root.

Building a Binary Tree



- Since newnode value is smaller than current value, we know that the newnode should be on the left tree.
- Transverse current downwards…

Building a Binary Tree

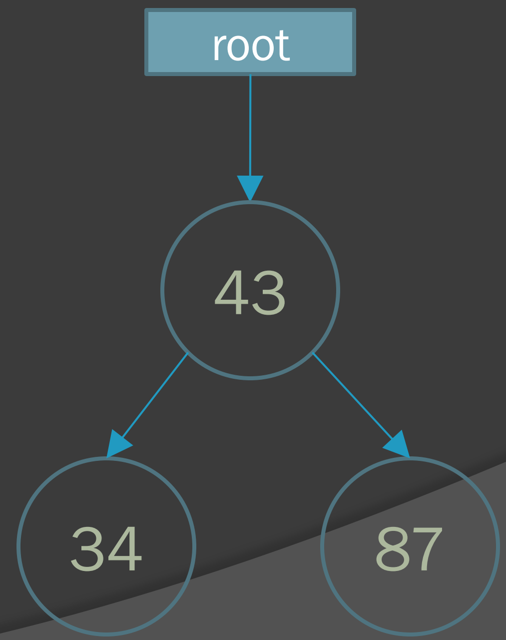


- Since now newnode value is smaller than current node and LT is empty, we know that newnode should now be made LT of current.

Different Sequence of Insertion

- Difference sequence of insertion gives a different tree though they are same values

VS



Binary Tree Node class

```
class CBTnode
{
public: // for ease of access :-)
     int data;
     CBTnode *left; //left child
     CBTnode *right; //right child
}
```

Binary Tree class

```
class CBTree
{
private:
        //pointers to nodes
        CBTnode *root, *current, *prev, *newnode;
        int count;
public:
        CBTree();
        ~CBTree();

        // The following functions assumes this BT
        // stores integers.
        // Modify the para to suit the data type that you
        // wants to store in the tree (don't forget to
        // change the CBTnode class too...
        void insert(int);
        bool search(int);
        bool remove(int);
}
```

Summary

- Understand tree terminology
- Growing a Tree
- Adding nodes to Tree