How would you create a data structure where you needed fast insertion, removal and lookup?
- e.g. Create a phone book structure

None of our known data structures are really good for this type of thing...
- Fast lookup/search and fast insertion/removal are at odds with each other
  - Insertion/Removal are slow if you have to sort
  - Search is slow if data is not sorted
- Need some remembrance of random access
  - Stacks/Queues/Deques can’t do this
A binary tree, T, is either empty or is constructed such that:
- T has a special node called the root
- T has two sets of nodes, L and R, called the left subtree and the right subtree of T
- L and R are also binary trees

Just like Linked Lists...
- Binary Trees are made up of Nodes
- Nodes point to other nodes
- Binary Trees are a recursive data structure
Linked Lists

In linked lists, each Node pointed to one other Node and looked something like this:

```cpp
template<typename T>
class Node {
    public:
        //The data section
        T data;

        //The link
        Node<T>* next;
};
```
In binary trees, each Node can point to two other Nodes and looks something like this:

```cpp
template <typename T>
class BTNode {
    public:
        // The data section
        T data;

        // The links
        BTNode<T>* left;
        BTNode<T>* right;
};
```

Binary Tree
Binary Tree
Binary Tree Terminology

- Trees grown *downward* in computer science
- Each BTNode has 0, 1 or 2 *Child Nodes* that are below it in the tree
- A BTNode that has a child is called that child’s *Parent Node*
- Parent nodes point to child nodes via a *branch* (child nodes don’t need to point to parents, but they can if they want)
Binary Trees

- A is the Parent of B and C
- B is the Left Child of A
- C is the Right Child of A
Binary Tree Terminology

- The **Root Node** is the only BTNode in the tree that has no parent node.
- A BTNode is a **Leaf Node** if it has no children.
- The **Level** of a BTNode in the tree is the number of branches from the root to the node.
- The **Height** of the tree is the number of nodes on the longest path from root to leaf.
Binary Trees

- A is the **Root Node**
- C, D, E are **Leaf Nodes**
- Level(A) = 0
  - Level(B) = 1
  - Level(D) = 2
- The **Height** of the tree is 3
Binary Trees

What kind of operations do we need to perform on a binary tree?

- height()
- size()
- search()
- insert()
- delete()
- print()

NOTE: Pretty much every operation requires a way to traverse the tree
template<typename T>
class BinaryTree {
private:
    BTNode<T>* root;

public:
    BinaryTree();
    ~BinaryTree();

    int height() const;
    int size() const;
    void print() const;
    void insert(const T& value);
    bool remove(const T& value);
    bool search(const T& value) const;
};
If you had to calculate the height of a binary tree, how would you do it?
Binary Tree Height

\[ \text{Height} = 1 + \max(\text{Height(Root->left)}, \text{Height(Root->right)}) \]
Binary Tree Height

//public helper function
int BinaryTree<T>::height() {
    return height(this->root);
}

int BinaryTree<T>::height(BTNode<T>* node) {
    if (node == NULL) { // Base Case
        return 0;
    } else { // Recursive Case
        return 1 +
            max(height(node->left),
                height(node->right));
    }
}
Alternate Notation

Root → A

B → D

C → E

F → G

A B C

D E F

G H J

Short-hand Notation

A

B C

D E F

G H J
If you had to print out all the nodes in a binary tree, how would you do it?
If you had to print out all the nodes in a binary tree, how would you do it?

At each node in the tree, you have three choices:
- Print the current node
- Go to the left child
- Go to the right child

These choices leads to different kinds of traversals:
- **In Order**: print left, print current, print right
- **Pre Order**: print current, print left, print right
- **Post Order**: print left, print right, print current
Binary Tree Traversal

**In Order Traversal:**
- Traverse A's left subtree
- Print A
- Traverse A's right subtree

Order of output:
G D B H E I A C F
Binary Tree Traversal

In Order Traversal (left, current, right)

//public helper function
int BinaryTree<T>::inOrderTraversal() const {
   inOrder(this->root);
}

void BinaryTree<T>::inOrder(BTNode<T>* node) const {
   if (node != NULL) {
      inOrder(node->left);  // Traverse left subtree
      cout << node->data << " ";  // Print current node
      inOrder(node->right);  // Traverse right subtree
   }
}
Binary Tree Traversal

**Pre Order Traversal:**
- Print A
- Traverse A’s left subtree
- Traverse A’s right subtree

Order of output:
A B D G E H I C F
Binary Tree Traversal

Pre Order Traversal (left, current, right)

```cpp
//public helper function
int BinaryTree<T>::preOrderTraversal() const {
    preOrder(this->root);
}

void BinaryTree<T>::preOrder(BTNode<T>* node) const {
    if (node != NULL) {
        cout << node->data << " "; //Print current node
        preOrder(node->left); //Traverse left subtree
        preOrder(node->right); //Traverse right subtree
    }
}
```
Binary Tree Traversal

- **Post Order Traversal:**
  - Traverse A’s left subtree
  - Traverse A’s right subtree
  - Print A

- Order of output: G D H I E B F C A
Binary Tree Traversal

Post Order Traversal (left, current, right)

```cpp
//public helper function
int BinaryTree<T>::postOrderTraversal() const {
    postOrder(this->root);
}

void BinaryTree<T>::postOrder(BTNode<T>* node) const {
    if (node != NULL) {
        postOrder(node->left); //Traverse left subtree
        postOrder(node->right); //Traverse right subtree
        cout << node->data << " "; //Print current node
    }
}
```
If you had to find a particular value in a binary tree, how would you do it?
Binary Tree Issues

So far binary trees are interesting, but they don’t provide any extra benefit
- Search still requires us to potentially look at every single node in the tree
  - Still $O(n)$
- If you’re going to insert a new Node, where should you put it in the tree?

What’s missing here is some extra organization that makes it easier to find things