1 Tree

- A tree is a collection of elements with a hierarchical relation over such elements.
- The actual definition is as follows: The empty collection is a tree (the empty tree). A single element is usually called a node. Node is a tree.
- If n is a node and $T_1, T_2, T_3, \ldots, T_n$ are trees, then n related to $T_1, T_2, \ldots, T_n$ is drawn:
  - n is called the root
  - $T_1 \ldots T_n$ are called the subtrees of n.
- A *path* is a sequence of nodes. $< a_1, a_2, a_3, \ldots, a_n >$ when n_{i+1} is a parent of $n_i 0 \leq i < n$.
- The *depth* of a node a is the number of nodes in the path from a to the root.
- the *height* of a tree is the greatest depth of a node in the tree.

Note Every tree has only one root.

Child The root of each subtree T_1, T_n are called the children of n

n Is called the parent of the root of each subtree T_1, T_n

Siblings If two roots have the same parent they are called siblings.

Leaf A leaf is a node with no children.

Degree The degree of a node a is the number of children of a.
- The degree of a tree is the highest degree of a node in the tree.

decedent/ancestor If there is a path from node a to node z then a is called a decedent of z. z is called an ancestor of a. The root is every node's ancestor.

1.1 Binary Search Tree

- Binary: Degree 2
- Search Conditions
  - find(T,x)
  - getMin(T)
  - getMax(T)
  - insert(T)
  - remove(T,x)

```
// Data Structure BinaryTree

template <classname T>
class TreeNode {
    T m_data;
    TreeNode *m_right;
    TreeNode *m_left;
};

// To use recursion, functions cannot be a method of TreeNode
const T& getMin(TreeNode *t) {
    if (t == nullptr) { /* error */ }

    if (t -> m_left == nullptr) {
        return t-> m_data;
    } else {
        return getMin(t -> m_left);
    }
}

const T& getMax(TreeNode *t) {
    if (t == nullptr) { /* error */ }
    TreeNode *p = t;

    while (p -> m_right != nullptr) {
        p = p -> m_right;
    }

    return p -> m_data;
}

bool T& find(TreeNode *t, const T& x) {
    if (t == nullptr) { return false; }
    if (t -> m_data == x) { return true; }

    if (x < t -> m_data) {
        return find(t - > m_left, x);
    } else if (x > t -> m_right) {
        return find(t -> right, x);
    }
}

void insert(TreeNode * &t, const T& x) {
    if (t == nullptr) {
        t = new TreeNode;
        t -> m_right = nullptr;
```

```
        t -> m_left = nullptr;
    } else (x < t -> m_data) {
        insert(t -> m_left);
    } else if (x > t -> m_data) {
        insert(t -> m_right);
    } else {
        return; // This is a duplicate. No duplicates allowed.
    }
}

void remove(TreeNode * &t, const T& x) {
/*
3 Cases:
- No Children
- One Child
- Two Children

To remove, you have choice. Max of Left or Min of right.
*/

    if (t == nullptr) {
        return;
    }
    if (x < t -> m_data) {
        remove(t -> left, x);
    } else if (x > t -> m_data) {
        remove(t -> right x);
    } else {
        // FOUND X!
        if (t -> m_right == nullptr && t -> m_left == nullptr) {
            // No children
            delete t;
            t = nullptr;
        } else if (t -> m_right == nullptr || t-> m_left == nullptr) {
            TreeNode *temporary = t -> m_right;
            if (temporary == nullptr) {
                temporary = t -> left;
            }

            // Now, temporary points to the chlid
            delete x;
            t = temporary;
        } else {
            // X has two children
            t -> m_data = getMin(t -> m_right);
            remove(t -> m_right, t -> m_data);
```

```
        }
    }
}
```

- collection of objects
- repetition is not allows
  - SETS!
- Why? Who cares? WHY NOT VECTORS?
  - find()
    - Find of size 500. $log_2 500 = 8.9$
    - 5000. $log_2 5000 = 12.2$
    - 5 Million. $log_2 5 Million = 22.25$
    - Most important operations.
  - insert()
    - $log_2 n$
  - remove()
    - $log_2 n$

index at i, right$(i) = 2i + 1$, left$(i) = 2i + 2$. The parent of is $\frac{i-2}{2}$