CS 225
Data Structures

February 18 – Traversals
G Carl Evans
Binary Tree – Defined

A binary tree $T$ is either:

- OR
- OR
Tree Property: height

*height(T)*: length of the longest path from the root to a leaf

Given a binary tree T:

*height(T) =*

![Binary Tree Diagram]

- C
- S
- 2
- 2
- 5
- X
- A
Tree Property: full

A tree $F$ is **full** if and only if:

1. 
2. 

```
    C
   /|
  /  |  
S   X
  /|
 C C
  /|
A  2
  /|
2  2 5
```
Tree Property: perfect

A perfect tree $P$ is:

1. 

2. 
Tree Property: complete

**Conceptually:** A perfect tree for every level except the last, where the last level is “pushed to the left”.

**Slightly more formal:** For any level $k$ in $[0, h-1]$, $k$ has $2^k$ nodes. For level $h$, all nodes are “pushed to the left”.
Traversals

```
+  
/  |  *  
|  |  |  
|  |  |  
|  |  |  
b a c d e
```
Traversals

template<class T>
void BinaryTree<T>::__Order(TreeNode * cur)
{
}
template<class T>
void BinaryTree<T>::___Order(TreeNode * cur) {
    if (cur != NULL) {
        __________________;
        ___Order(cur->left);
        __________________;
        ___Order(cur->right);
        __________________;
    }
}
template<class T>
void BinaryTree<T>::___Order(TreeNode * cur) {
    if (cur != NULL) {
        _________________;
        ___Order(cur->left);
        _________________;
        ___Order(cur->right);
        _________________;
    }
}
A Different Type of Traversal

```
+  *
-  /
 a  b  c  d  e
```
A Different Type of Traversal

template<class T>
void BinaryTree<T>::levelOrder(TreeNode * root) {

}
Traversal vs. Search

Traversal

Search
Search: Breadth First vs. Depth First

Strategy: Breadth First Search (BFS)

Strategy: Depth First Search (DFS)
Dictionary ADT

Data is often organized into key/value pairs:

- UIN ➔ Advising Record
- Course Number ➔ Lecture/Lab Schedule
- Node ➔ Incident Edges
- Flight Number ➔ Arrival Information
- URL ➔ HTML Page

...
#pragma once

class Dictionary {
public:

private:
    // ...
};
Binary Tree as a Search Structure
Binary Tree (BST)

A BST is a binary tree $T$ such that:
```cpp
#pragma once

template <class K, class V>
class BST {
public:
    BST();
    void insert(const K key, V value);
    V remove(const K & key);
    V find(const K & key) const;
    TreeIterator traverse() const;

private:
};
```
template<class K, class V>

TreeNode *& root, const K & key) const {

}}
template<class K, class V>

TreeNode *& root, const K & key) {
}
template<class K, class V>

__remove(TreeNode * & root, const K & key) { 

}
remove(40);
remove(25);
remove(10);
remove(13);