Learning Objectives

Review fundamental tree terminology

Introduce the concept and properties of a binary tree

Conceptualize and code tree traversals

Introduce fundamental tree search strategies
Trees

“The most important non-linear data structure in computer science.”
- David Knuth, The Art of Programming, Vol. 1

A tree is:
- 
-
Tree Terminology Review

Find an **edge** that is not on the longest **path** in the tree.

Which vertex is the **root** of the tree?

How many parents does each vertex have?

Which vertex has the fewest **children**?

Which vertex has the most **ancestors**?

Which vertex has the most **descendants**?

List all vertices in b’s left **subtree**? In a’s?

List all **leaves** in the tree.
There are many *types* of trees
Binary Tree

A **binary tree** $T$ is either:

- OR

-
Tree Property: height

$\text{height}(T)$: length of the longest path from the root to a leaf

Given an arbitrary binary tree $T$, write a recursive equation for height:
A tree \textbf{F} is \textbf{full} iff one of two things is true:

1.

2.
Tree Property: perfect

A tree $P$ of height $h$ is **perfect** iff one of two things is true:

1.

2.
Tree Property: complete

A tree $P$ of height $h$ is **complete** if:

1. For every level except the last the tree is perfect

2. The last level is ‘pushed to the left’

**How many nodes are at level $k$ in a complete tree?**
Tree Property: complete

A complete tree $C$ of height $h$, $C_h$:

1. $C_{-1} = \{\}$

2. $C_h$ (where $h>0$) = \{r, T_L, T_R\} and either:

   $T_L$ is __________ and $T_R$ is __________

   OR

   $T_L$ is __________ and $T_R$ is __________
Tree Property: complete

Is every **full** tree **complete**?

Is every **complete** tree **full**?
Tree ADT
#pragma once

template <class T>
class BinaryTree {
    public:
        /* ... */
    
    private:
};
#pragma once

template <class T>
class BinaryTree {
    public:
        /* ...
   */

    private:

        struct TreeNode {
            T data;
            TreeNode *left;
            TreeNode *right;
        }

        TreeNode *root_;
Trees aren’t new:
Theorem: If there are $n$ objects in our representation of a binary tree, then there are ________ NULL pointers.
“Wasted Overhead” in Binary Tree

**Theorem:** If there are $n$ objects in our representation of a binary tree, then there are $n+1$ NULL pointers.

Induction Step:
Traversal
Traversals

```
template<class T>
void BinaryTree<T>::__Order(TreeNode * root)
{
}
```
template<class T>
void BinaryTree<T>::__Order(TreeNode * root)
{
    if (root != NULL) {
        ____________________;
        __Order(root->left);
        ____________________;
        __Order(root->right);
        ____________________;
    }
}
Traversals

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

```
+  
-  *
  /  d  e
 a  b  c
d
```

A Different Type of Traversal

```
template<class T>
void BinaryTree<T>::lOrder(TreeNode * root)
{
    Queue<TreeNode*> q;
    q.enqueue(root);
    while( q.empty() == False){
        TreeNode* temp = q.head();
        process(temp);
        q.dequeue();
        q.enqueue(temp->left);
        q.enqueue(temp->right);
    }
}
```
Traversal vs Search

Traversal

Search