CS 225
Data Structures

Feb. 16 – Tree Proof
Wade Fagen-Ulmschneider
Tree Property: full

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

1. 
2. 

![Diagram of a full tree](image)
Tree Property: perfect

A perfect tree $P$ is defined in terms of the tree’s height.

Let $P_h$ be a perfect tree of height $h$, and:

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 all levels \( k \) in \([0, h-1]\), \( k \) has \( 2^k \) nodes. For level \( h \), all nodes are “pushed to the left”.
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?

If every complete tree full?
Tree ADT

*insert*, inserts an element to the tree.

*remove*, removes an element from the tree.

*traverse*,
 ifndef BINARYTREE_H
#define BINARYTREE_H

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

#endif
Trees aren’t new:
Trees aren’t new:
How many NULLs?

**Theorem:** If there are $n$ data items in our representation of a binary tree, then there are ___________ NULL pointers.
How many NULLs?

Base Cases:

n = 0:

n = 1:

n = 2:
How many NULLs?

Induction Hypothesis:
How many NULLs?

Consider an arbitrary tree $T$ containing $n$ data elements:
Traversals

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

Traversals are a way to visit all nodes within a tree data structure. They are typically used in computer science and mathematics to systematically visit every node in a tree. There are several types of traversals, including preorder, inorder, and postorder, each with a different order of visiting the nodes.

Preorder traversal visits the root node first, then traverses the left subtree, and finally the right subtree.

Inorder traversal visits the left subtree first, then the root node, and finally the right subtree.

Postorder traversal visits the left subtree first, then the right subtree, and finally the root node.

In the given tree, one possible traversal is: a → b → c → d → e.
Traversals

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