# 5: Trees

CSE326 Spring 2002

April 8, 2002

## Dictionaries

Remember ADTs?

*Dictionary*

- MakeEmpty, IsEmpty
- Find
- Insert, Delete



## Using Dictionaries

Dictionaries are everywhere in computers

- 
- 
- 

## Implementations of Dictionaries

| Implementation | Time to | | |
|---|---|---|---|
| | Find | Insert | Delete |
| Unordered List | | | |
| Unordered Array | | | |
| Sorted Array | | | |



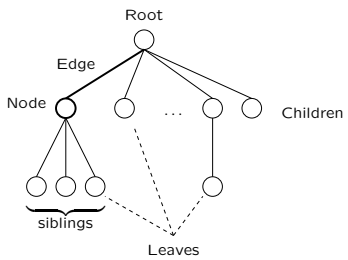
## Trees

Trees in the World

Trees in CS

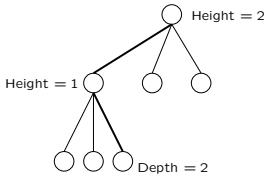


## Definitions




Anatomy of a Tree

## Depth and Height



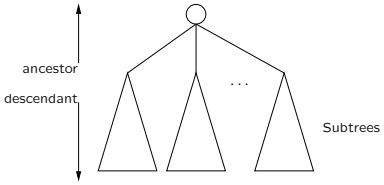



## Recursive Definition

A tree is. . .

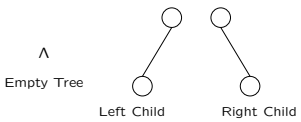
A root

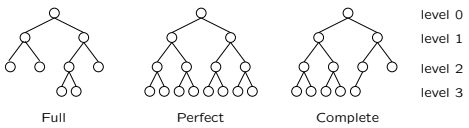
A root with distinct child trees





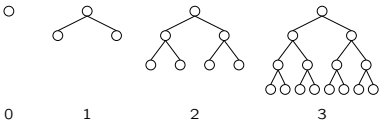

## Binary Trees



## Counting with Perfect Trees

- How many nodes in level $i$ of a perfect tree?



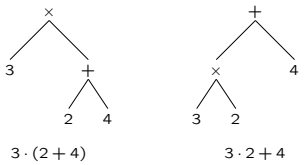

- How many nodes, total?

- What *fraction* of the tree is in the last level?



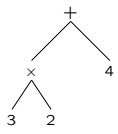
## Binary Tree Traversals

- A *traversal* is an order for visiting all nodes of a tree
- Easy to think of in terms of *expression trees*






## Three Kinds of Traversals



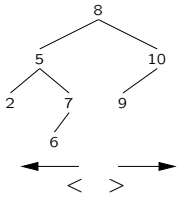

- Preorder: $+ \times 3\ 2\ 4$
- Inorder: $3 \times 2 + 4$
- Postorder: $3\ 2 \times 4 +$
  - For an expression, this is *prefix* order

## Code for Traversals

```
void TreeNode::PreOrder() {
   Visit ();
   if (LeftChild ()) LeftChild()->PreOrder();
   if (RightChild ()) RightChild()->PreOrder();
}

void TreeNode::InOrder() {




}

void TreeNode::PostOrder() {




}
```



## Binary Search Trees

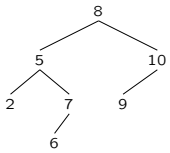



```
Node *Node::Find(Key k)
{
   if (key == k) return this;
   if (k < key
       && LChild())
     return LChild->Find(k);
   if (k > key
       && RChild())
     return RChild->Find(k);
  return NULL;
}
```

```
void Node::Insert(Key k)
{
   if (k <= key) {
     if (LChild())
       LChild()->Insert(k);
     else
       SetLChild(new Node(k));
   }
   if (k > key) {
     if (RChild())
       RChild()->Insert(k);
     else
       SetRChild(new Node(k));
   }
}
```

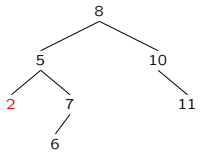


## Traversals on BSTs



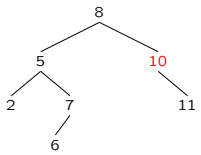

Preorder Inorder Postorder

## BST Deletion I



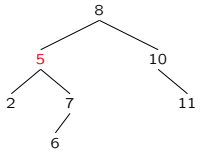



## BST Deletion II

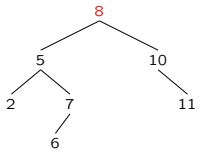





## BST Deletion III



## BST Deletion IV






## BST Deletion

- Locate node to delete
- If no children, delete
- If one child, replace with child
- Otherwise,
  - Locate *successor* (or *predecessor*)
  - Replace with successor (or predecessor)



## BST Analysis

- Perfect Tree

Complete? Full?

- *Unbalanced* Tree

## Making an Unbalanced Tree