Binary Trees

Lecture 31
Section 19.1

Robb T. Koether

Hampden-Sydney College

Fri, Apr 13, 2018

Outline

Binary Trees

Definition (Binary Trees)

A binary tree is a data structure with the following properties.

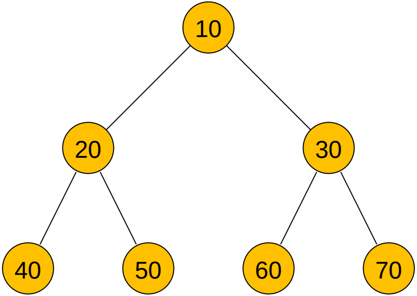
- It is either empty or it has a root node.
- Each node in the binary tree may be linked to up to two other nodes, called the left and right children.
- Each node, except the root node, has exactly one parent. The root node has no parent.

Outline

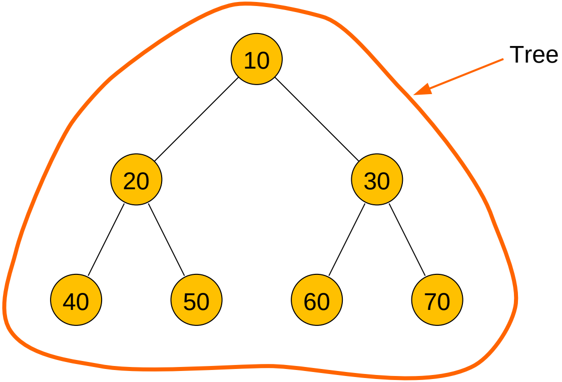
Binary Tree Terminology

- The tree metaphor - tree, root, branch, leaf.
- The family metaphor - parent, child, sibling, ancestor, descendant.

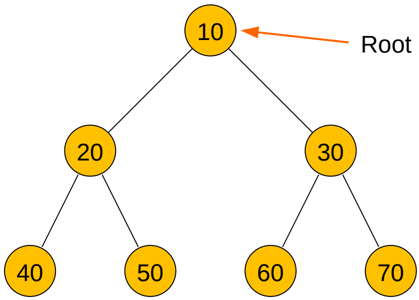
Tree Metaphor



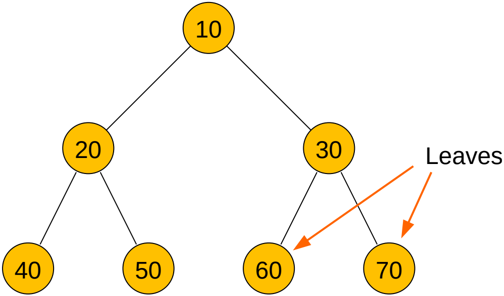
Tree Metaphor



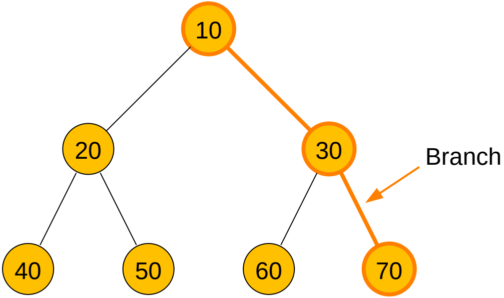
Tree Metaphor



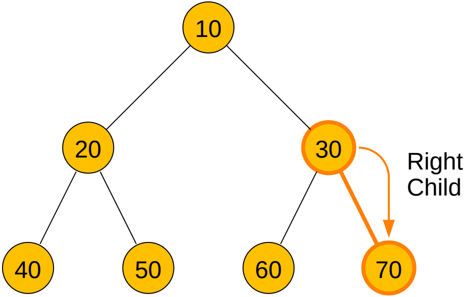
Tree Metaphor



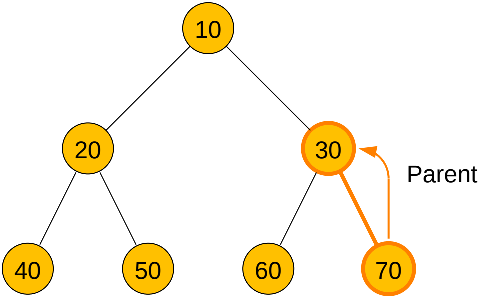
Tree Metaphor



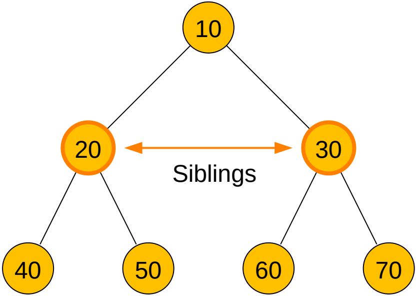
Tree Metaphor



Tree Metaphor



Tree Metaphor



Outline

Outline

Binary Search Tree

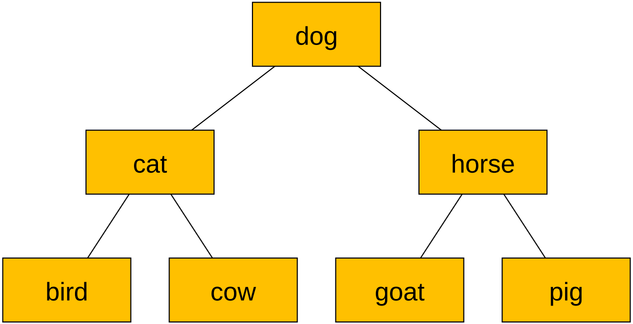
Definition (Binary search tree)

In a binary search tree, at every node,

- Every element in the left subtree is less than or equal to the element at the node.
- Every element in the right subtree is greater than or equal to the element at the node.

- Storing words in alphabetical order in a binary search tree allows for very rapid look-up.

Binary Search Tree



{bird, cat, cow, dog, goat, horse, pig}

Outline

Binary Search Tree

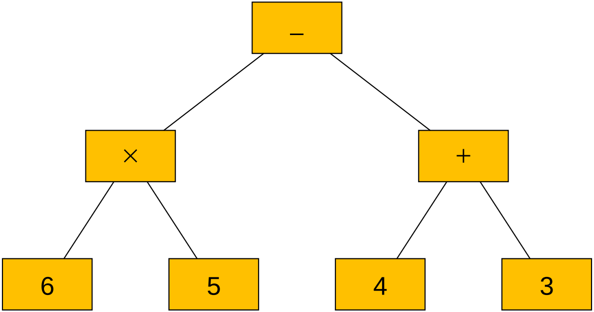
Definition (Binary search tree)

In a binary expression tree, at every interior node,

- The node element is an operator.
- The left subtree represents the left operand of the operator at the node.
- The right subtree represents the right operand of the operator at the node.

- It is very easy to evaluate an expression in a binary expression tree.

Binary Search Tree



$$6 \times 5 - (4 + 3)$$

Outline

Outline

Binary Tree Implementation

Binary Tree Node Data Members

- `T m_value`
- `BinaryTreeNode<T>* m_left`
- `BinaryTreeNode<T>* m_right`

- `m_value` – The value stored in the node.
- `m_left` – A pointer to the left child node or `NULL`.
- `m_right` – A pointer to the right child node or `NULL`.

Binary Tree Implementation

Binary Tree Data Member

- `BinaryTreeNode<T>* m_root`

- `m_root` – A pointer to the root node or `NULL`.

Outline

Insertions and Deletions

- Insertions and deletions in a binary tree are considerably more complicated than they were for linked lists.
- That is because a binary tree is not linear.
- Where should the new node be inserted?
- How would we specify a position?
- When a node is deleted, how is the hole filled?

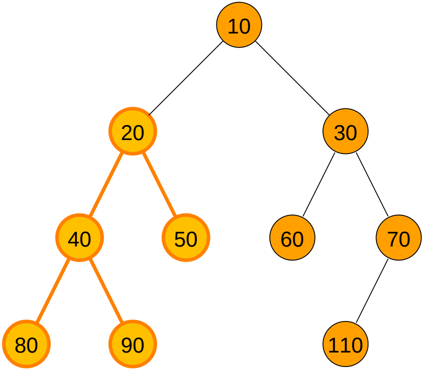
The `size()` Function

- The `size()` function returns the number of nodes in the tree.
- Since the size of the tree is not stored as a data member, we will need to count the nodes.

The `size()` Function

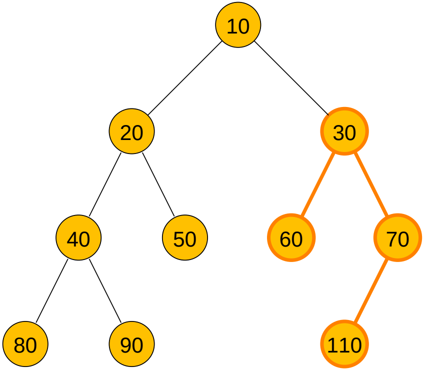
- How do we count the nodes in a binary tree?
- Think recursively.
- There is the root node.
- There are the nodes in the left subtree.
- There are the nodes in the right subtree.

The `size()` Function



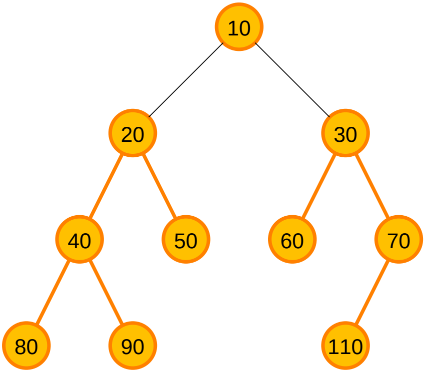
Size of left subtree = 5

The `size()` Function



Size of right subtree = 4

The `size()` Function



Size of tree = 1 + 5 + 4 = 10

The `size()` Function

- Therefore,

$$\text{size(root)} = 1 + \text{size(left)} + \text{size(right)}.$$

The `size()` Function

Example (Public, Nonrecursive `size()` Function)

```
int size() const
{
    return size(m_root);
}
```

The size() Function

Example (Private, Recursive size() Function)

```
int size(BinaryTreeNode* node) const
{
    if (node == NULL)
        return 0;
    else
        return 1 + size(node->m_left) + size(node->m_right);
}
```

The `search()` Function

The `search()` Function

```
BinaryTreeNode* search(T value) const;
```

- The `search()` function has the above prototype
- It returns a pointer to the node where the value was found, or
- It returns `NULL` if the value was not found.

The search() Function

- Write the public and private `search()` functions.
- Write public and private `output()` functions that will print the values of the nodes from “left to right.”

Outline

Assignment

Assignment

- Read Section 19.1.