CSE 1320 - Intermediate Programming

Binary Trees

Alex Dillhoff

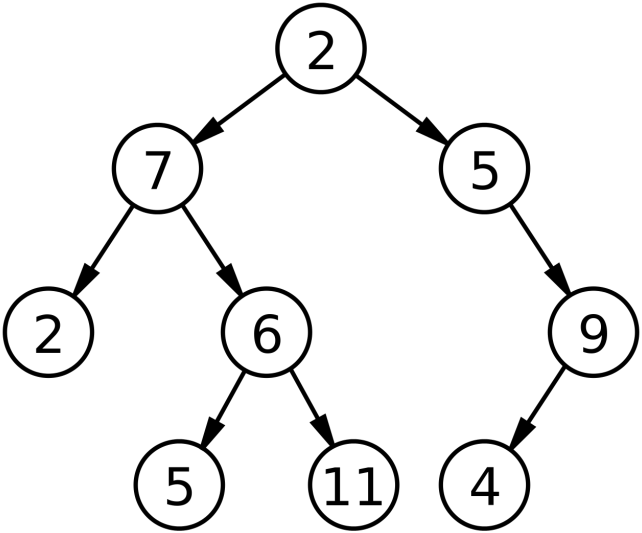
University of Texas at Arlington

Binary Trees

Binary trees are a graph-based data structure.

They are defined as a hierarchical tree of nodes in which each node has at most two sub-nodes.

Binary Trees



Binary Trees

Tree-based data structures share a few common properties and definitions:

- The **size** of a tree T is determined by the total number of nodes in T.
- The **root** of a tree T is the starting point of T.
- A **leaf node** in a tree T is a node that has no sub-nodes.
- The **height** of a tree T is determined by the *length* of the shortest path between the root of T and the lowest leaf node of T.

Binary Trees

Each node in a binary tree contains the following information:

- A reference to the **left** sub-node.
- A reference to the **right** sub-node.
- A **key** representing some value.

Binary Trees

Binary trees are especially useful for sorting and searching data efficiently.

The data can be as simple as a single scalar value or as complex as a multi-membered `struct` as long as a suitable **key** can be chosen to represent each node.

Constructing a Binary Tree

In general, a binary tree can be constructed in any fashion as long as each node has **at most** two sub-nodes.

Although the usefulness of a binary tree comes from utilizing their hierarchical nature, there is no requirement that the data need to be sorted in any particular way.

Binary Search Trees

To take advantage of the benefits of efficient sorting and searching, a Binary Search Tree must be used.

A Binary Search Tree is defined by the following property:

If x is a node in a binary search tree and y is a sub-node of x, then y is a *left sub-node* if $y.key \leq x.key$ and y is a *right sub-node* if $y.key \geq x.key$.

Operations

There are several operations that can be performed on a Binary Search Tree.

1. Traversal
2. Search
3. Insertion
4. Deletion

Operations - Traversal

Given a binary tree T, there are several different ways to traverse the nodes of T.

1. Depth First Search (Preorder)
2. Depth First Search (Inorder)
3. Depth First Search (Postorder)
4. Breadth First Search

Depth First Search (Preorder)

To perform a preorder DFS, the *key* of the current node is printed *before* moving to the sub-nodes.

```
void preorder_dfs(Node* n) {
    if (n != NULL) {
        printf("%d\n", n->key);
        preorder_dfs(n->left);
        preorder_dfs(n->right);
    }
}
```

Depth First Search (Inorder)

To perform a inorder DFS, the *key* of the current node is printed *between* the sub-nodes.

```
void inorder_dfs(Node* n) {
    if (n != NULL) {
        inorder_dfs(n->left);
        printf("%d\n", n->key);
        inorder_dfs(n->right);
    }
}
```

Depth First Search (Inorder)

For a Binary Search Tree, an inorder traversal will print the items out in order from least to greatest, according to they key.

Depth First Search (Postorder)

To perform a postorder DFS, the *key* of the current node is printed *after* moving to the sub-nodes.

```
void postorder_dfs(Node* n) {
    if (n != NULL) {
        postorder_dfs(n->left);
        postorder_dfs(n->right);
        printf("%d\n", n->key);
    }
}
```

Breadth First Search

A Breadth First Search prints each level of the tree in order from top to bottom, left to right.

In this way, the breadth of the layer is explored before moving to the next level in the height of the tree.

Operations - Search

Searching a Binary Search Tree involves looking at each node, starting with the root, until the desired *key* is found.

If the target value is less than the *key*, the left sub-node is traversed. Otherwise, the right sub-node is traversed.

This continues until a leaf node is reached.

Operations - Search

Example: Search through a BST

Operations - Insertion

Besides searching, inserting a node into a Binary Search Tree is one of the greatest benefits of using them.

A new node is inserted depending on its *key* relative to the tree T.

Operations - Insertion

Example: Insert new item into BST.

Operations - Insertion

The **insert** operation is easy to implement and comes with the benefit that the tree T retains the property of a Binary Search Tree after the item is inserted.

Deleting a node is more complicated and may require reorganization of the tree.

Operations - Deletion

If the node is a leaf, the node can simply be set to NULL.

If the node has a single subnode, the subnode is then assigned to the parent of the current node.

If the node has two subnodes, the graph must be restructured depending on the data.