BBM 201
Data structures

Lecture 11:
Trees

Dad, dad, look at that tree!!

It's just a tree, son...
But is a binary tree...

Daniel Sturi (turnoff.us)

2018-2019 Fall
Content

• Terminology
• The Binary Tree
• The Binary Search Tree
## Terminology

- Trees are used to represent relationships
- Trees are hierarchical in nature
  - “Parent-child” relationship exists between nodes in tree.
  - Generalized to ancestor and descendant
  - Lines between the nodes are called edges
- A **subtree** in a tree is any node in the tree together with all of its descendants
Terminology

• Only access point is the root
• All nodes, except the root, have one parent
  – like the inheritance hierarchy in Java
• Traditionally trees are drawn upside down

![Tree Diagram]
Terminology

(a) A tree;
(b) a subtree of the tree in part a
Terminology

FIGURE 15-2 (a) An organization chart; (b) a family tree
Properties of Trees and Nodes

- **siblings**: two nodes that have the same parent
- **edge**: the link from one node to another
- **path length**: the number of edges that must be traversed to get from one node to another

Path length from root to this node is 3
General Tree

– A general tree is a data structure in that each node can have infinite number of children
– A general tree cannot be empty.
Binary Tree

• A Binary tree is a data structure in that each node has at most two nodes left and right.

• A Binary tree can be empty.
n -ary tree

– A generalization of a binary tree whose nodes each can have no more than n children.
Example: Algebraic Expressions.

FIGURE 15-3 Binary trees that represent algebraic expressions.
Level of a Node

• Definition of the level of a node $n$:
  – If $n$ is the root of $T$, it is at level 1.
  – If $n$ is not the root of $T$, its level is 1 greater than the level of its parent.
Height of Trees

- The height of a node is the number of edges on the longest downward path between that node and a leaf.

Height of tree = 2
The Height of Trees

Binary trees with the same nodes but different heights.
Depth of a Tree

• The path length from the root of the tree to this node.

The depth of a node is its distance from the root
- a is at depth zero
- e is at depth 2

The depth of a binary tree is the depth of its deepest node
- This tree has depth 4
Full, Complete, and Balanced Binary Trees
Full Binary Trees

• Definition of a full binary tree
  – If T is empty, T is a full binary tree of height 0.
  – If T is not empty and has height h > 0, T is a full binary tree if its root’s subtrees are both full binary trees of height h – 1.
  – Every node other than the leaves has two children.
Facts about Full Binary Trees

- You cannot add nodes to a full binary tree without increasing its height.
- The number of nodes that a full binary tree of height $h$ can have is $2^{(h+1)} - 1$.
- The height of a full binary tree with $n$ nodes is $\log_2(n+1) - 1$.
- The height of a complete binary tree with $n$ nodes is $\text{floor}(\log_2 n)$.
Complete Binary Trees

Every level, except possibly the last, is completely filled, and all nodes are as far left as possible
Full, Complete or Other?

not binary
Full, Complete or other?
Full, Complete or other?
Full, Complete or other?
Full, Complete or other?
Full, Complete or Other?
Full, Complete or other?
Full, Complete or Other?
- A balanced binary tree has the minimum possible height for the leaves.
Number of Nodes in a Binary Tree

<table>
<thead>
<tr>
<th>Level</th>
<th>Number of nodes at this level</th>
<th>Total number of nodes at this level and all previous levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$1 = 2^0$</td>
<td>$1 = 2^1 - 1$</td>
</tr>
<tr>
<td>2</td>
<td>$2 = 2^1$</td>
<td>$3 = 2^2 - 1$</td>
</tr>
<tr>
<td>3</td>
<td>$4 = 2^2$</td>
<td>$7 = 2^3 - 1$</td>
</tr>
<tr>
<td>4</td>
<td>$8 = 2^3$</td>
<td>$15 = 2^4 - 1$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$h$</td>
<td>$2^h$</td>
<td>$2^{h+1} - 1$</td>
</tr>
</tbody>
</table>

depth: $h$  
level: $h+1$
Traversals of a Binary Tree

• General form of recursive traversal algorithm

1. **Preorder Traversal**
   Each node is processed before any node in either of its subtrees

2. **Inorder Traversal**
   Each node is processed after all nodes in its left subtree and before any node in its right subtree

3. **Postorder Traversal**
   Each node is processed after all nodes in both of its subtrees
Preorder Traversal

1. Visit the root
2. Visit the left subtree
3. Visit the right subtree
Algorithm TraversePreorder(n)

Process node n

if n is an internal node then

    TraversePreorder( n -> leftChild)

    TraversePreorder( n -> rightChild)
Inorder Traversal

1. Visit Left subtree
2. Visit the root
3. Visit Right subtree
Algorithm TraverseInorder(n)

    if n is an internal node then
        TraverseInorder( n -> leftChild)
    Process node n

    if n is an internal node then
        TraverseInorder( n -> rightChild)
Postorder Traversals

1. Visit Left subtree
2. Visit Right subtree
3. Visit the root
Algorithm TraversePostorder(n)

    if n is an internal node then
        TraversePostorder( n -> leftChild)
        TraversePostorder( n -> rightChild)
    Process node n
Traversals of a Binary Tree

(a) Preorder: 60, 20, 10, 40, 30, 50, 70
(b) Inorder: 10, 20, 30, 40, 50, 60, 70
(c) Postorder: 10, 30, 50, 40, 20, 70, 60

(Numbers beside nodes indicate traversal order.)
The 3 different types of traversal

Pre-order Traversal
FBADCEGIH

In-order Traversal
ABCDEFGHI

Post-order Traversal
ACEDBHI GF
Binary Tree Operations

• Test whether a binary tree is empty.
• Get the height of a binary tree.
• Get the number of nodes in a binary tree.
• Get the data in a binary tree’s root.
• Set the data in a binary tree’s root.
• Add a new node containing a given data item to a binary tree.
Binary Tree Operations

• Remove the node containing a given data item from a binary tree.
• Remove all nodes from a binary tree.
• Retrieve a specific entry in a binary tree.
• Test whether a binary tree contains a specific entry.
• Traverse the nodes in a binary tree in preorder, inorder, or postorder.
A binary tree can be represented using
- Linked List
- Array

Note: Array is suitable only for full and complete binary trees
struct node
{
    int key_value;
    struct node *left;
    struct node *right;
};

struct node *root = 0;
```c
void inorder(node *p)
{
    if (p != NULL)
    {
        inorder(p->left);
        printf(p->key_value);
        inorder(p->right);
    }
}

void preorder(node *p)
{
    if (p != NULL)
    {
        printf(p->key_value);
        preorder(p->left);
        preorder(p->right);
    }
}

void postorder(node *p)
{
    if (p != NULL)
    {
        postorder(p->left);
        postorder(p->right);
        printf(p->key_value);
    }
}
```
void destroy_tree(struct node *leaf)
{
    if (leaf != 0)
    {
        destroy_tree(leaf->left);
        destroy_tree(leaf->right);
        free(leaf);
    }
}
The Binary Search Tree

• Binary tree is ill suited for searching a specific item
• Binary search tree solves the problem
• Properties of each node, \( n \)
  – \( n \)'s value is \textcolor{red}{
  greater} than all values in the left subtree \( T_L \)
  – \( n \)'s value is \textcolor{red}{
  less} than all values in the right subtree \( T_R \)
  – Both \( T_R \) and \( T_L \) are binary search trees.
The Binary Search Tree

FIGURE 15-13 A binary search tree of names.
The Binary Search Tree

FIGURE 15-14 Binary search trees with the same data as in Figure 15-13
The Binary Search Tree

FIGURE 15-14 Binary search trees with the same data as in Figure 15-13
The Binary Search Tree

Binary search trees with the same data as in Figure 15-13
Binary Search Tree Operations

• Test whether a binary search tree is empty.
• Get height of a binary search tree.
• Get number of nodes in a binary search tree.
• Get data in binary search tree’s root.
• Insert new item into the binary search tree.
• Remove given item from the binary search tree.
Binary Search Tree Operations

- Remove all entries from a binary search tree.
- Retrieve given item from a binary search tree.
- Test whether a binary search tree contains a specific entry.
- Traverse items in a binary search tree in
  - Preorder
  - Inorder
  - Postorder.
Searching a Binary Search Tree

• Search algorithm for binary search tree
struct node *search(int key, struct node *leaf) {
    if( leaf != 0 ) {
        if(key==leaf->key_value) {
            return leaf;
        } else if(key<leaf->key_value) {
            return search(key, leaf->left);
        } else {
            return search(key, leaf->right);
        }
    } else return 0;
}
Creating a Binary Search Tree

Empty subtree where the search algorithm terminates when looking for Frank
void insert(int key, struct node **leaf)
{
    if( *leaf == 0 )
    {
        *leaf = (struct node*) malloc( sizeof( struct node ) );
        (*leaf)->key_value = key;
        /* initialize the children to null */
        (*leaf)->left = 0;
        (*leaf)->right = 0;
    }
    else if(key < (*leaf)->key_value)
    {
        insert(key, &(*leaf)->left );
    }
    else if(key > (*leaf)->key_value)
    {
        insert(key, &(*leaf)->right );
    }
}
# Efficiency of Binary Search Tree Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Average case</th>
<th>Worst case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retrieval</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Insertion</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Removal</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Traversal</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>

The Big O for the retrieval, insertion, removal, and traversal operations of the ADT binary search tree