Binary Tree

Definition: A binary tree is either empty or it consists of a node called the root together with two binary trees called the left subtree and the right subtree of the root.
Traversal of Binary Tree

- Preorder: VLR
- Inorder: LVR
- Postorder: LRV

Expression Tree

<table>
<thead>
<tr>
<th>Expression</th>
<th>Preorder</th>
<th>Inorder</th>
<th>Postorder</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a + b$</td>
<td>$+ a b$</td>
<td>$a + b$</td>
<td>$a b +$</td>
</tr>
<tr>
<td>$\log x$</td>
<td>$\log x$</td>
<td>$\log x$</td>
<td>$x \log n$</td>
</tr>
<tr>
<td>$n!$</td>
<td></td>
<td>$n!$</td>
<td></td>
</tr>
<tr>
<td>$a - (b \times c)$</td>
<td>$- a b c$</td>
<td>$a - b c$</td>
<td>$a b c -$</td>
</tr>
<tr>
<td>$(a &lt; b)$ or $(c &lt; d)$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Expression Tree**

\[ x = \frac{-b + (b + 2 - 4 \times a \times c) \uparrow 0.5}{2 \times a} \]

**Linked List Implementation**

```c
typedef struct treenode TreeNode;

typedef struct treenode {
    TreeEntry entry;
    TreeNode *left;
    TreeNode *right;
} TreeNode;
```
Inorder Traversal Routines

```c
void B(TreeNode *root,
       void (*Visit)(TreeEntry x))
{
    if (root) {
        A(root->left, Visit);
        Visit(root->entry);
        A(root->right, Visit)
    }
}
```

Quiz: How the Postorder and Preorder Traversal Routines will be?

Binary Search Tree
Idea

• Dilemma
  – In a Linked list there is no obvious way of moving into the middle point of list other than sequential traversal.
  – On the other hand, in a contiguous list it is difficult to add/delete entries!

Can we find an implementation for ordered lists in which we can search quickly (as with binary search on a contiguous list) and in which we can make insertions and deletions quickly (as with a linked list)?

• Solution
  – Binary Search Tree!

Binary Search Tree

**Definition:** A *binary search tree* is a binary tree that is either empty or in which every node contains a key and satisfies the conditions:
1. The key in the left child of a node (if it exists) is less than the key in its parent node.
2. The key in the right child of a node (if it exists) is greater than the key in its parent node.
3. The left and right subtrees of the root are again binary search trees.

• Assumption: No two entries in a binary search tree may have equal keys.
Op-1: Searching into a BST

```c
TreeNode *TreeSearch(TreeNode *root, KeyType target) {
    if (root) {
        if (LT(target, root->entry.key))
            root = TreeSearch(root->left, target);
        else if (GT(target, root->entry.key))
            root = TreeSearch(root->right, target);
        return root;
    }
}
```

Op-2: Insertion into a BST

```c
TreeNode *InsertTree(TreeNode *root, TreeNode *newnode) {
    if (!root) {
        root = newnode;
        root->left = root->right = NULL;
    } else if (LT(newnode->entry.key, root->entry.key))
        root->left = InsertTree(root->left, newnode);
    else
        root->right = InsertTree(root->right, newnode);
    return root;
}
```

Quiz: insert e, b, d, f, a, g and c
Example: Insert e, b, d, f, a, g, c

Uniqueness of BST
Tree Sort

- Quiz: How to get a sorted list from a BST?

```
     e
    / \
   b   f
  /   / \
 a   d   g

Tree Sort & Quick Sort

- In BST
  - The first key becomes the root of the tree.
  - During search the key is compared with the root.
- In Quick Sort
  - The first key is compared with the first pivot...

**Theorem 9.1** TreeSort makes exactly the same comparisons of keys as does quicksort when the pivot for each sublist is chosen to be the first key in the sublist.

**Corollary 9.2** In the average case, on a randomly ordered list of length $n$, treesort does $2n \ln n + O(n) \approx 1.39n \ln n + O(n)$ comparisons of keys.

- One advantage of Tree sort: It does not require all the key to be present throughout the sorting process. But, Quick Sort requires access to all the items.
Op-3: Deletion from BST

Example
Today’s Math
Rabbit Counting

Evaluation of Fibonacci Series

- Let us consider an infinite series:
  
  \[ F(x) = F_0 + F_1 x + F_2 x^2 + \ldots + F_n x^n + \ldots \]

  \[ F(x) = F_0 + F_1 x + F_2 x^2 + \ldots + F_n x^n + \ldots \]

  \[ x \cdot F(x) = F_0 x + F_1 x^2 + \ldots + F_n x^n + \ldots \]

  \[ x^2 \cdot F(x) = \quad + F_0 x^2 + \ldots + F_{n-2} x^n + \ldots \]

  Therefore:

  \[ (1 - x - x^2) F(x) = F_0 + (F_1 - F_0) x = x \]

  \[ F(x) = \frac{x}{(1 - x - x^2)} = \frac{1}{\sqrt{5}} \left( \frac{1}{1 - Ax} - \frac{1}{1 - Bx} \right) \]

  \[ A = \frac{1}{2} (1 + \sqrt{5}) = 1.618034 \quad B = \frac{1}{2} (1 - \sqrt{5}) = -0.618034 \]
Evaluation (continued..)

- From the original definition:
  \[ F(x) = F_0 + F_1 x + F_2 x^2 + \ldots + F_n x^n. \]
  \[ = \frac{1}{\sqrt{5}} (1 + A x + A^2 x^2 + \ldots) - \frac{1}{\sqrt{5}} (1 + B x + B^2 x^2 + \ldots) \]
  \[ F_n = \frac{1}{\sqrt{5}} (A^n - B^n) \]

- B is always smaller than 1, therefore:
  \[ F_n = \frac{1}{\sqrt{5}} A^n = \frac{1}{\sqrt{5}} \left[ \frac{1 + \sqrt{5}}{2} \right]^n \]
  rounded to nearest integer

F(n) is always integer!

The number A has been studied since the time of the Greeks- is called golden mean-the ratio of A:1 gives the most pleasing shape of rectangle.

The Parthenon and many other Greek buildings have sides with this ratio!