

Binary Search Trees

15-111

Data Structures

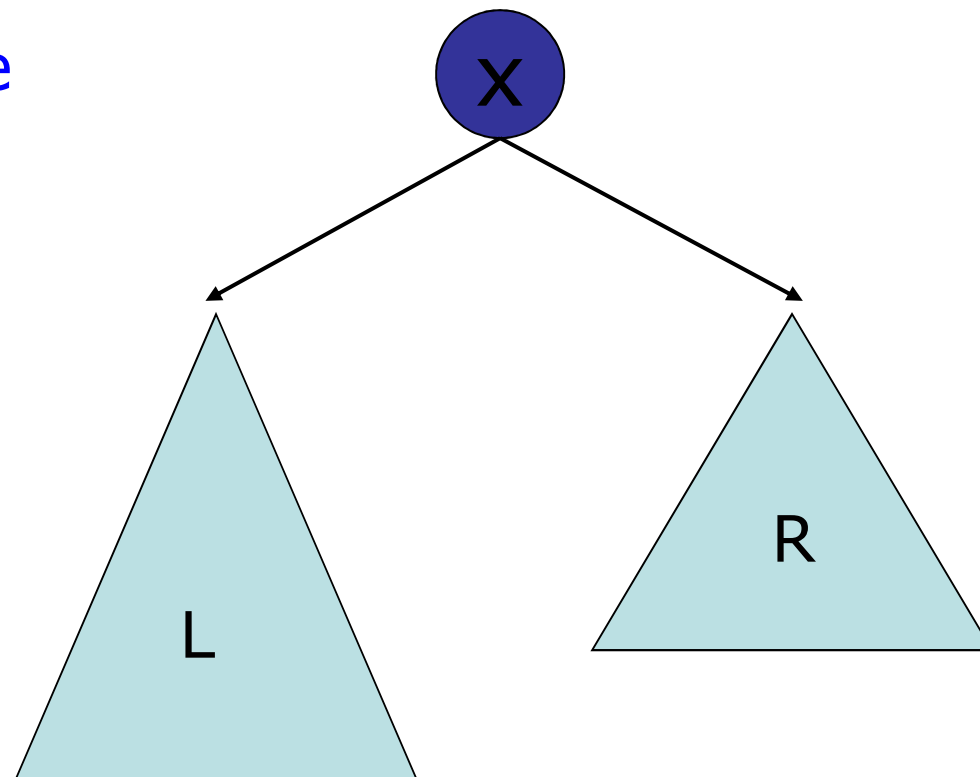
Ananda Gunawardena

Tree Data Structure

- A non-linear data structure that follows the shape of a tree (i.e. root, children, branches, leaves etc)
- Most applications require dealing with hierarchical data (eg: organizational structure)
- Trees allows us to find things efficiently
 - Navigation is $O(\log n)$ for a “balanced” tree with n nodes
- A Binary Search Tree (BST) is a data structure that can be traversed / searched according to an order
- A binary tree is a tree such that each node can have at most 2 children.

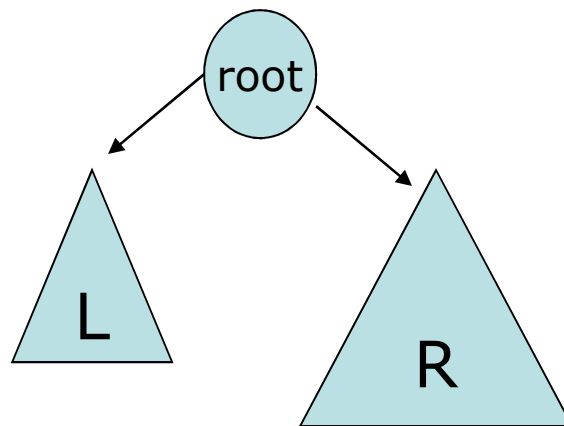
BST In Pictures

Empty Tree

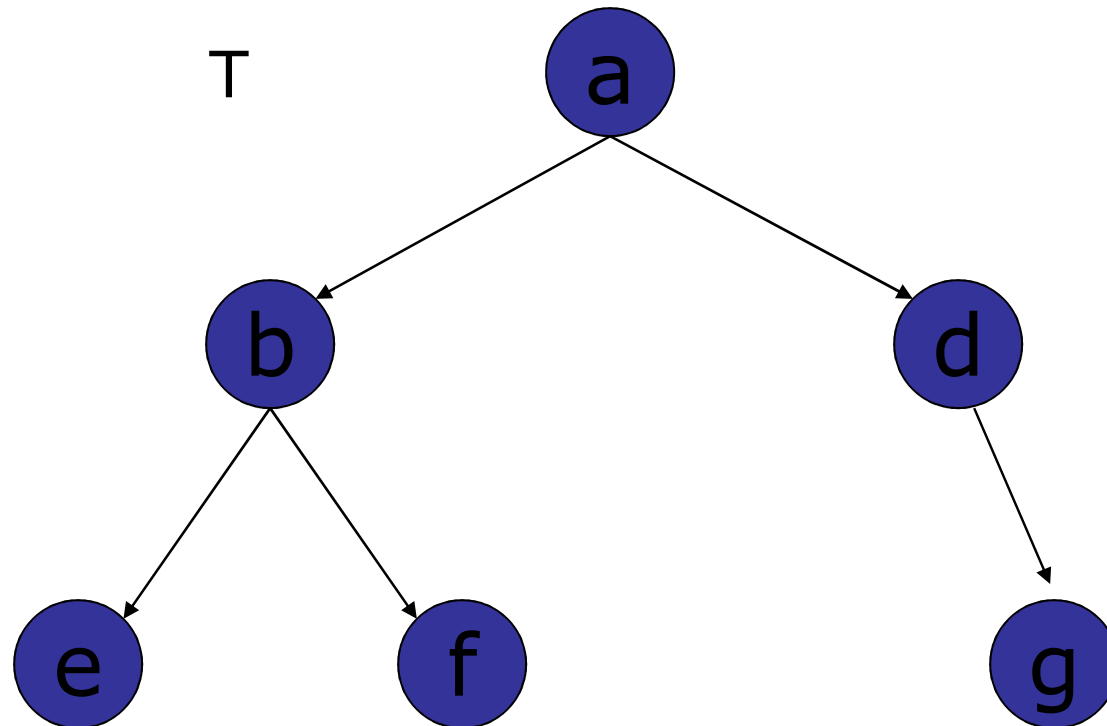


Definition of Flat T

- Given a Binary Tree T, Flat(T) is a sequence obtained by traversing the tree using **inorder** traversal
 - That is for each node, recursively visit
 - Left Tree, Then Root, then Right Tree



Flattening a BT



$\text{flat}(T) = e, b, f, a, d, g$
7/8/2011

Def: Binary Search Tree

A binary Tree is a **binary search tree (BST)**
if and only if

flat(T) is an **ordered** sequence.

Equivalently, in (x, L, R) all the nodes in L are less than x , and all the nodes in R are larger than x .

BT Definitions

Definitions

- A **path** from node n_1 to n_k is defined as a path n_1, n_2, \dots, n_k such that n_i is the parent of n_{i+1}
- **Depth of a node** is the length of the path from root to the node.
- **Height of a node** is length of a path from node to the deepest leaf.
- **Height of the tree** is the height of the root

Definitions

- **Full Binary Tree**
 - Each node has zero or two children
- **Complete Binary Tree**
 - All levels of the tree is full, except possibly the last level, where nodes are filled from left to right
- **Perfect Tree**
 - A Tree is perfect if total number of nodes in a tree of height h is $2^{h+1} - 1$

Binary Tree Questions

- What is the **maximum height** of a binary tree with n nodes? What is the **minimum height**?
- What is the minimum and maximum **number of nodes** in a binary tree of height h ?
- What is the **minimum number** of nodes in a full tree of height h ?
- Is a complete tree a full tree?
- Is perfect tree a full and complete tree?

Binary Tree Properties

- Counting Nodes in a Binary Tree
 - The max number of nodes at level i is 2^i ($i=0,1,\dots,h$)
 - Therefore total nodes in all levels is
$$n = 1 + 2^1 + 2^2 + \dots + 2^h$$
 - Find a relation between n and h .
- A **complete tree** of height, h , has between 2^h and $2^{h+1}-1$ nodes.
- A **perfect** tree of height h has $2^{h+1}-1$ nodes

Binary Tree Questions

- What is the **maximum** number of nodes at level k ? (root is at level 0)

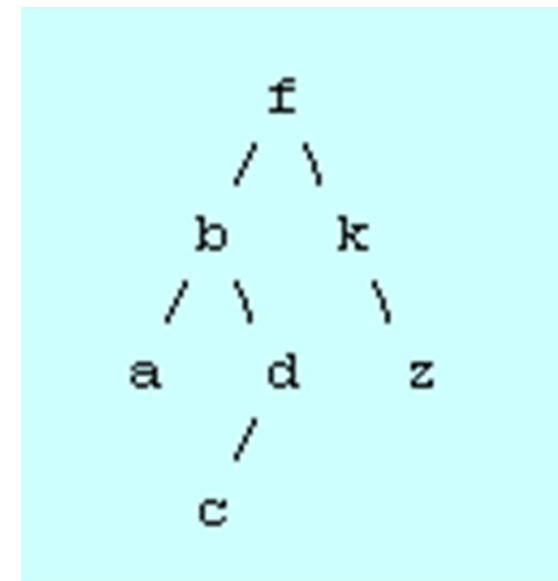
BST Operations

Tree Operations

- Tree Traversals
 - Inorder, PreOrder, PostOrder
 - Level Order
- Insert Node, Delete Node, Find Node
- Order Statistics for BST's
 - Find k^{th} largest element
 - num nodes between two values
- Other operations
 - Count nodes, height of a node, height of a tree, balanced info

Binary Tree Traversals

- **Inorder** Traversals
 - Visit nodes in the order
 - Left-Root-Right
- **PreOrder** Traversal
 - Visit nodes in the order
 - Root-Left-Right
- **PostOrder** Traversal
 - Visit nodes in
 - Left-Right-Root



Inorder Traversal

```
private void inorder(BinaryNode root)
{
    if (root != null) {
        inorder(root.left);
        process root;
        inorder(root.right);
    }
}
```


Preorder Traversal

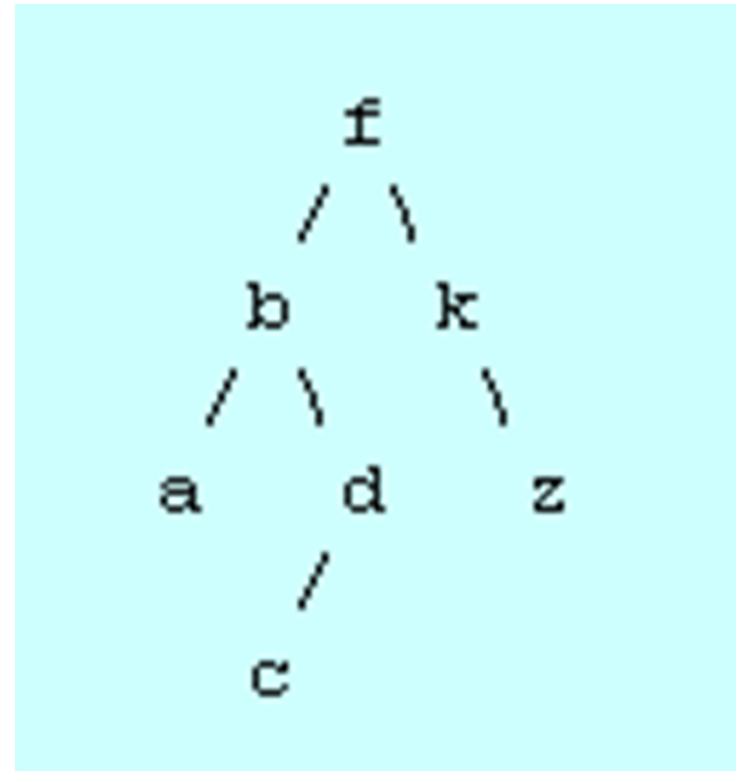
```
private void preorder(BinaryNode root)
{
    if (root != null) {
        process root;
        preorder(root.left);
        preorder(root.right);
    }
}
```

Postorder Traversal

```
private void postorder(BinaryNode root)
{
    if (root != null) {
        postorder(root.left);
        postorder(root.right);
        process root;
    }
}
```

Level order or Breadth-first traversal

- Visit nodes by levels
- Root is at level zero
- At each level visit nodes from left to right
- Called “**Breadth-First-Traversal(BFS)**”



Level order or Breadth-first traversal

BFS Algorithm

```
enqueue the root
while (the queue is not empty)
{
    dequeue the front element
    print it
    enqueue its left child (if present)
    enqueue its right child (if present)
}
```

