**COMPSCI 130**

Design and Analysis of Algorithms

5 – Binary Search Trees

---

**Binary Search Trees**

- Data structure for holding sorted elements
  - Efficient searching, insertion, deletion
  - Efficiency depends on tree structure

---

**Why Trees?**

- Sorted list arrays → fast searching, but:
  - Slow inserts
  - Slow deletes

- Binary search trees → fast for search and update
  - Variants of BSTs are very important in database applications, e.g., for indexing
  - BSTs often used within other algorithms

---

**Binary Trees**

**Terminology**

- μ is the parent of ν
- ν is the left child of μ
- ρ is an ancestor of μ and ν
- μ and ν are descendants of ρ

---

**Binary Trees**

**Recursive Definition**

- a binary tree is:
  - (empty)
  - a root node with a binary tree as left subtree and a binary tree as right subtree

---

**Binary Trees**

**Terminology**

- root node
- edge
- internal nodes
- node
- external nodes / leaves

---
Binary Trees

Items are stored in the internal nodes; leaves typically just null pointers.

The tree is sorted if, for every node, items in its left subtree are smaller than the item in the node, and items in the right subtree are larger.

More terminology:
- The size of the tree is the number of nodes.
- A tree is full if every internal node has two children.
- A path is a sequence of edges between two nodes.
- The length of the path is the number of edges in the path.
- For every node \( \mu \), the length of the unique path between \( \mu \) and the root is the depth of \( \mu \).
- The height of the tree is the maximum depth of any node.
- The path length is the sum of depths over all nodes.

Binary Search Trees

A binary search tree (BST) is a sorted binary tree.

We assume each node is a structure holding
- Data
- A pointer to the (root of the) left subtree
- A pointer to the (root of the) right subtree

In a Java-like language, this would look like:
```java
class Node {
    Item info;
    Node left;
    Node right;
}
```

Typically we store only unique keys in the tree
- Additional data may be stored with each key
- BSTs are good for Sets, Maps, etc.
- E.g., a database index might store keys in a BST together with lists of pointers to rows.

Searching

```java
Node search(Node tree, Item x) {
    if tree = NULL return NOT_FOUND;
    else if x < tree.info return search(tree.left, x);
    else if x = tree.info return tree;
    else return search(tree.right, x);
}
```

Worst case:
- Tree is a linked list \( \Theta(n) \)
Best case:
- Tree is balanced \( \Theta(\log n) \)

Inserting

Complexity is same as for searching!
Deleting

• Follow path to node v containing item
• Multiple cases:
  – v has no internal node children: remove v
  – v has one internal child: make that child the parent of v and remove v
  – v has two internal children: find the rightmost internal node in the left subtree of v, remove it, and substitute it for v (see illustration)

Example: delete(tree, "k")

Final Words

• Basic operations (search, insert, delete) have same complexity = O(height of tree)
• If tree is balanced, height = O(log n)
• Worst case, tree is a linked list, height = O(n)
• Trees can become unbalanced through sequences of inserts and deletes
• Next topic: how to maintain balance (efficiently)