CSC 212
Data Structures and Abstractions
Spring 2016

Lecture 10: Binary Search Trees I

1. Previously ...

   - Trees
     - definition
     - properties
     - traversals

2. Today ...

   - Binary Trees
     - Binary Search Trees
     - definition
     - search

3. Binary Trees

   - A k-ary tree where \( k = 2 \)

4. Binary Trees

5. Linked Structure for Binary Trees

   - Every node has:
     - data
     - parent
     - left child
     - right child

6. Collections/Dictionaries

7. Computational Cost

   - | what?           | sequential search (unordered sequence) | binary search (ordered sequence) |
     |-----------------|----------------------------------------|-------------------------------|
     | search          | \( O(n) \)                             | \( O(\log n) \)               |
     | insert          | \( O(n) \)                             | \( O(\log n) \)               |
     | delete          | \( O(n) \)                             | \( O(\log n) \)               |
     | min/max         | \( O(n) \)                             | \( O(1) \)                   |
     | floor/ceiling   | \( O(n) \)                             | \( O(\log n) \)               |
     | rank            | \( O(n) \)                             | \( O(\log n) \)               |

8. Computational Cost

9. Computational Cost
Binary Search Trees

Binary Search Tree

A BST is a binary tree

A BST has symmetric order

each node \( x \) in a BST has a key \( \text{key}(x) \)

for all nodes \( y \) in the left subtree of \( x \), \( \text{key}(y) < \text{key}(x) \) **

for all nodes \( y \) in the right subtree of \( x \), \( \text{key}(y) > \text{key}(x) \) **

(**) assume that the keys of a BST are pairwise distinct

```cpp
class BSTNode {
    private:
        int data;
        BSTNode *left;
        BSTNode *right;
    public:
        BSTNode(int d) {
            data = d;
            left = right = NULL;
        }
        ~BSTNode() {}
        friend class BSTree;
};
```

```cpp
class BSTree {
    private:
        BSTNode *root;
    public:
        BSTree();
        ~BSTree();
        void insert(int d);
        bool remove(int d);
        BSTNode *search(int d);
        void traversal(int type);
    };
```

Search

Start at root node

If the search key matches the current node’s key then **found**

If search key is greater than current node’s key

search recursively on right child

If search key is less than current node’s key

search recursively on left child

Stop recursion when current node is NULL (**not found**)

Basic Operations

```cpp
class BSTree {
    private:
        BSTNode *root;
    public:
        BSTree();
        ~BSTree();
        void insert(int d); // internal functions
        bool remove(int d); // constructor & destructor
        BSTNode *search(int d); // basic operations
        void traversal(int type);
    };
```
BSTNode *BSTree::search(BSTNode *p, int d) {
    if (p) {
        if (p->data == d) return p;
        else if (p->data < d) return search(p->right, d);
        else return search(p->left, d);
    }
    return NULL;
}

BSTNode *BSTree::search(int d) {
    return search(root, d);
}