# Lecture 7b: Trees in C++

CS 70: Data Structures and Program Development

Thursday, March 5, 2020



## Learning Goals

- I can simulate left and right rotations (on paper)
- I can simulate `insertAtRoot` (on paper)
- I can simulate Randomized Binary Tree insertion
- I can represent a tree using C++ classes and structs
- I can implement `insert`, `lookup`, `rotate` in C++



## Recall: BST lookup vs. insert pseudocode

```
lookup(tree, x):
  if tree is empty:
    return false

  else if x == tree's root:
    return true

  else if x < tree's root:
    return lookup(left subtree, x)

  else if tree's root < x:
    return lookup(right subtree, x)
```

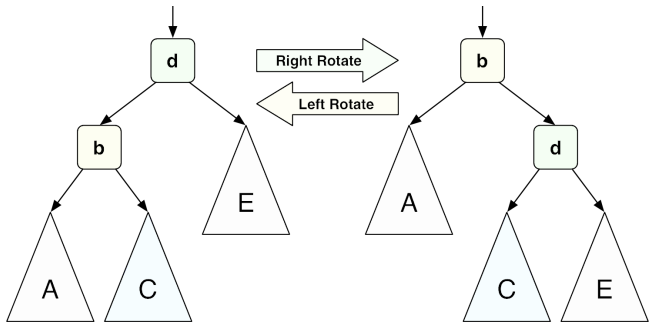
```
insert(tree, x):
  if tree is empty:
    make x its new root.

  else if x < tree's root:
    insert(left subtree, x)

  else if tree's root < x:
    insert(right subtree, x)
```



## Recall: Tree Rotations



## Recall: `insertAtRoot` pseudocode

```
insertAtRoot(tree, x):
   if tree is empty:
      make x its new root.

   else if x < tree's root:
      insertAtRoot(left subtree, x)
      do right rotation at tree's root.

   else if tree's root < x:
      insertAtRoot(right subtree, x)
      do left rotation at tree's root.
```

**Worst-case running time for perfect vs. really terrible tree?**



## Building better trees: Randomized Binary Trees

**Idea:**
**Emulate the nice behavior of pre-shuffled input sequences by inserting each new input at a random level of the tree.**

- maybe the root
- maybe the second level (root of a subtree)
- maybe the third level (root of a subsubtree), etc.

**Algorithm: to randomly insert `x` into a tree already having `n` nodes**

- with probability `1/(n+1)`, do insertAtRoot
- otherwise, *randomly* insert `x` somewhere in the correct subtree.

**Lookup in randomized BSTs doesn't change!**



## BST `randomizedInsert` pseudocode

```
randomizedInsert(tree, x):
   if tree currently has n elements,
   with probability 1/(n+1):
      insertAtRoot(tree, x)

   else if x < tree's root:
      randomizedInsert(left subtree, x)

   else if tree's root < x:
      randomizedInsert(right subtree, x)
```

## Representing Trees

"A binary tree is empty, or has a root and two (possibly empty) subtrees"

```
class IntTree {
 public: ...
 private:
    // "struct" == ("class" + public by default)
    struct Node {
       int value_;
       Node* left_;
       Node* right_;
    };

    Node* root_;
};
```



## Writing `insert`… with a helper function

```
class intTree {
 public:
    void insert(int m);
 private:
    struct Node { ... };
    Node* root_;

    void insertHelper(Node*& nd, int m);
};
```



## Insertion Code

```
void IntTree::insert(int m) {
  insertHelper(root_, m);
}

void IntTree::insertHelper(Node*& nd, int m) {
  if (nd == nullptr)
    nd = new Node{m}; // assume we wrote constructor
  else if (m < nd->value_)
    insertHelper(nd->left_, m);
  else
    insertHelper(nd->right_, m);
}
```



## Exercise: right rotation

## Rotation: Just a few pointer updates

```
void rotateRight(Node*& top) {
  Node* b = top->left_;     // b is d's left child
  top->left_ = b->right_;   // C becomes left child of d
  b->right_ = top;          // d becomes right child of b
  top = b;                  // top is now b
}

void rotateLeft(Node*& top) {
  Node* d = top->right_;    // d is b's left child
  top->right_ = d->left_;   // C becomes right child of b
  d->left_ = top;           // b becomes left child of d
  top = d;                  // top is now d
}
```



## Exercise: How would we add `exists` ?



## Tree Iterator

- **Considerations**
  - Encoding?
  - Which order should iterator yield?