CS 225

Data Structures

March 2 – AVL Analysis
Wade Fagen-Ulmschneider
Insertion into an AVL Tree

Insert (pseudo code):
1: Insert at proper place
2: Check for imbalance
3: Rotate, if necessary
4: Update height

struct TreeNode {
    T key;
    unsigned height;
    TreeNode *left;
    TreeNode *right;
};

_insert(6.5)
template <class T> void AVLTree<T>::_insert(const T & x, treeNode<T> * & t) {
    if (t == NULL) {
        t = new TreeNode<T>(x, 0, NULL, NULL);
    }

    else if (x < t->key) {
        _insert(x, t->left);
        int balance = height(t->right) - height(t->left);
        int leftBalance = height(t->left->right) - height(t->left->left);
        if (balance == -2) {
            if (leftBalance == -1) { rotate___________(t); }
            else { rotate___________(t); }
        }
    }

    else if (x > t->key) {
        _insert(x, t->right);
        int balance = height(t->right) - height(t->left);
        int rightBalance = height(t->right->right) - height(t->right->left);
        if (balance == 2) {
            if (rightBalance == 1) { rotate___________(t); }
            else { rotate___________(t); }
        }
    }

    t->height = 1 + max(height(t->left), height(t->right));
}
AVL Tree Analysis

We know: insert, remove and find runs in: ____________.

We will argue that: h is ____________.
AVL Tree Analysis

Definition of big-O:

...or, with pictures:
AVL Tree Analysis

An **upper** bound on the height $h$ for a tree of $n$ nodes
...is the same as...
A **lower** bound on the number of nodes $n$ in a tree of height $h$
Plan of Action

Since our goal is to find the lower bound on $n$ given $h$, we can begin by defining a function given $h$ which describes the smallest number of nodes in an AVL tree of height $h$: 
Simplify the Recurrence

\[ N(h) = 1 + N(h - 1) + N(h - 2) \]
State a Theorem

Theorem: An AVL tree of height $h$ has at least _________.

Proof:
I. Consider an AVL tree and let $h$ denote its height.

II. Case: ______________

An AVL tree of height ____ has at least ____ nodes.
An AVL tree of height ____ has at least ____ nodes.
Prove a Theorem

IV. Case: ______________
By an Inductive Hypothesis (IH):

We will show that:

An AVL tree of height ____ has at least ____ nodes.
Prove a Theorem

V. Using a proof by induction, we have shown that:

...and inverting:
Summary of Balanced BST

Red-Black Trees
- Max height: $2 \times \lg(n)$
- Constant number of rotations on insert, remove, and find

AVL Trees
- Max height: $1.44 \times \lg(n)$
- Rotations:
Summary of Balanced BST

Pros:
- Running Time:
  - Improvement Over:

- Great for specific applications:
Summary of Balanced BST

Cons:
- Running Time:

- In-memory Requirement:
Iterators

Why do we care?

```cpp
DFS dfs(...);
for ( ImageTraversal::Iterator it = dfs.begin(); it != dfs.end(); ++it ) {
    std::cout << (*it) << std::endl;
}
```
Iterators

Why do we care?

```cpp
DFS dfs(...);
for ( ImageTraversal::Iterator it = dfs.begin(); it != dfs.end(); ++it ) {
    std::cout << (*it) << std::endl;
}
```

```cpp
DFS dfs(...);
for ( const Point & p : dfs ) {
    std::cout << p << std::endl;
}
```
Iterators

Why do we care?

```c++
1 DFS dfs(...);
2 for ( ImageTraversal::Iterator it = dfs.begin(); it != dfs.end(); ++it ) {
3   std::cout << (*it) << std::endl;
4 }
```

```c++
1 DFS dfs(...);
2 for ( const Point & p : dfs ) {
3   std::cout << p << std::endl;
4 }
```

```c++
1 ImageTraversal & traversal = /* ... */;
2 for ( const Point & p : traversal ) {
3   std::cout << p << std::endl;
4 }
```
Iterators

```cpp
ImageTraversal *traversal = /* ... */;
for ( const Point & p : traversal ) {
    std::cout << p << std::endl;
}
```