CSE 3010 – Data Structures & Algorithms

Lecture #42

What will be covered today

- Helper functions
- Balancing an unbalanced tree
    - Cases to consider
- Insertion of a new node in an AVL tree

Some Helper Functions

```
//Returns the maximum of two integers
int maximum(int x, int y) {
    if(x > y)
        return(x);
    else
        return(y);
}

//Sets the balance factor for a node
int getBalanceFactor(AVLNODE* node) {
    return(getHeight(node->left) - getHeight(node->right));
}
```

Displaying a BST

```
//Displays a BST like a tree
void displayBST(AVLNODE* root, int space) {
    int i;
    if(root == NULL)
        return;
    else {
        displayBST(root->right, space + HORIZONTAL_GAP);
        printf("\n");
        for(i = 1; i <= space; i++)
            printf(" ");
        printf("%d\n", root->key);
        displayBST(root->left, space + HORIZONTAL_GAP);
    }
}
```

Height of a node

```
int getHeight(AVLNODE* node) {
    int left_height = 0;
    int right_height = 0;

    if(node == NULL)
        return(-1);
    else {
        left_height = getHeight(node->left);
        right_height = getHeight(node->right);

        return(maximum(left_height, right_height) + 1);
    }
}
```

Righting the unbalanced tree

- On insertion or deletion of a node, a binary can become unbalanced in various ways
- AVL trees implement some operations in order to keep the height small
  - Operations are performed on the **first unbalanced** node when traversing up from the new node

Inserting a new node in an AVL tree

- Insert a node as you would normally do in a BST. Let this node be **`w`**.
- Traverse up from **`w`** to **`root`**. Let **`z`** be the first unbalanced node in this path. In the path from **`w`** to **`z`**, let **`y`** be the child of **`z`** and **`x`** be the grandchild of **`z`**.
- Rebalance the tree by performing one of the following operations depending on the relationship between **`x`**, **`y`**, and **`z`**
  - **Left Left Case**: **`y`** is the left child of **`z`** and **`x`** is the left child of **`y`**
  - **Left Right Case**: **`y`** is the left child of **`z`** and **`x`** is the right child of **`y`**
  - **Right Right Case**: **`y`** is the right child of **`z`** and **`x`** is the right child of **`y`**
  - **Right Left Case**: **`y`** is the right child of **`z`** and **`x`** is the left child of **`y`**