Announcements


TODAY: balanced BST


The “height balance” of a tree $T$ is:

$$b = \text{height}(T_R) - \text{height}(T_L)$$

A tree $T$ is “height balanced” if:

- $T = {}$ OR
- $T = \emptyset, T_L, T_R^3, 161 \quad \text{and } T_L \text{ and } T_R \text{ are height balanced.}$
operations on BST - rotations

```
  50
  /  \
30   80
 /     /  \
20    60   90
  \
    /  \
   85   99
```

95
balanced trees - rotations
balanced trees - rotations
balanced trees - rotations summary:

- there are 4 kinds: left, right, left-right, right-left (symmetric!)
- local operations (subtrees not affected)
- constant time operations
- BST characteristic maintained

GOAL: use rotations to maintain balance of BSTs.

height balanced trees - we have a special name:

Three issues to consider as we move toward implementation:

Rotating
Maintaining height
Detecting imbalance
Maintaining height upon a rotation:
AVL trees: rotations (identifying the need)

if an insertion was in subtrees t3 or t4, and if an imbalance is detected at t, then a __________ rotation about t rebalances the tree.

We gauge this by noting that the balance factor at t->right is _____
AVL trees: rotations (identifying the need)

If an insertion was in subtrees t2 or t3, and if an imbalance is detected at t, then a ________________ rotation about t rebalances the tree.

We gauge this by noting that the balance factor at t->right is _____
AVL trees:

```
struct treeNode {
    T key;
    int height;
    treeNode * left;
    treeNode * right;
};
```

Insert:
insert at proper place
check for imbalance
rotate if necessary
update height
AVL tree insertions:

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