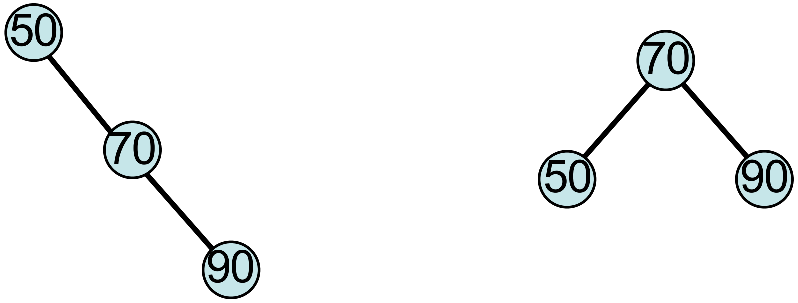
Announcements

MP5 available, due 10/30, 11:59p. EC due 10/23, 11:59p.

TODAY: balanced BST

http://www.qmatica.com/DataStructures/Trees/AVL/AVLTree.html



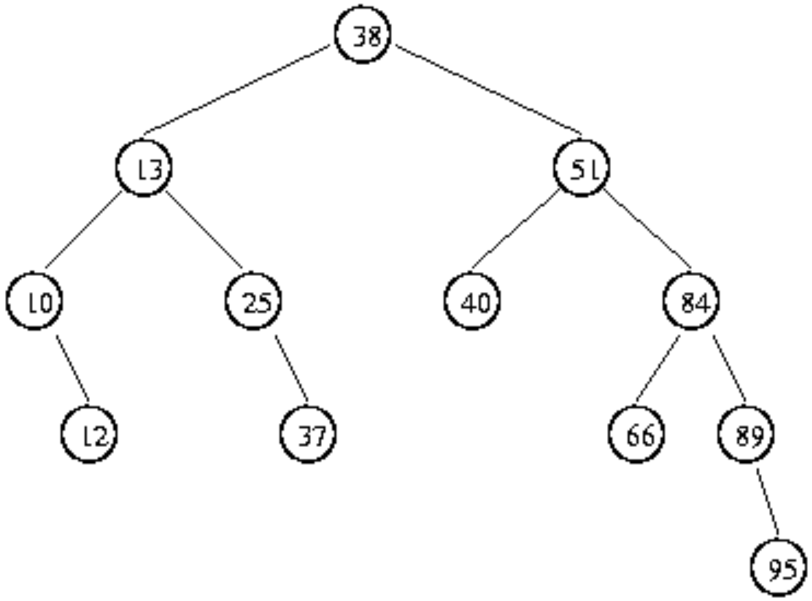
The “height balance” of a tree T is:

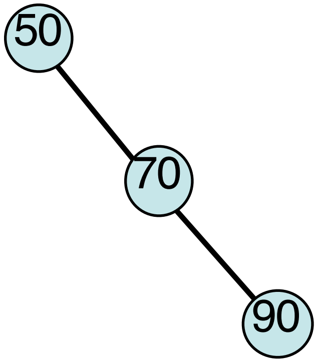
$$b = height(T_R) - height(T_L)$$

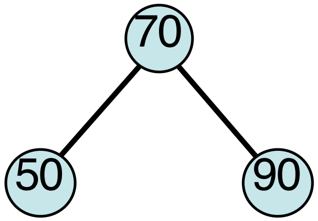
A tree T is “height balanced” if:

- $T = \{\}$ OR
- $T = \{r, T_L, T_R\}$, ______, and T_L and T_R are ht balanced.

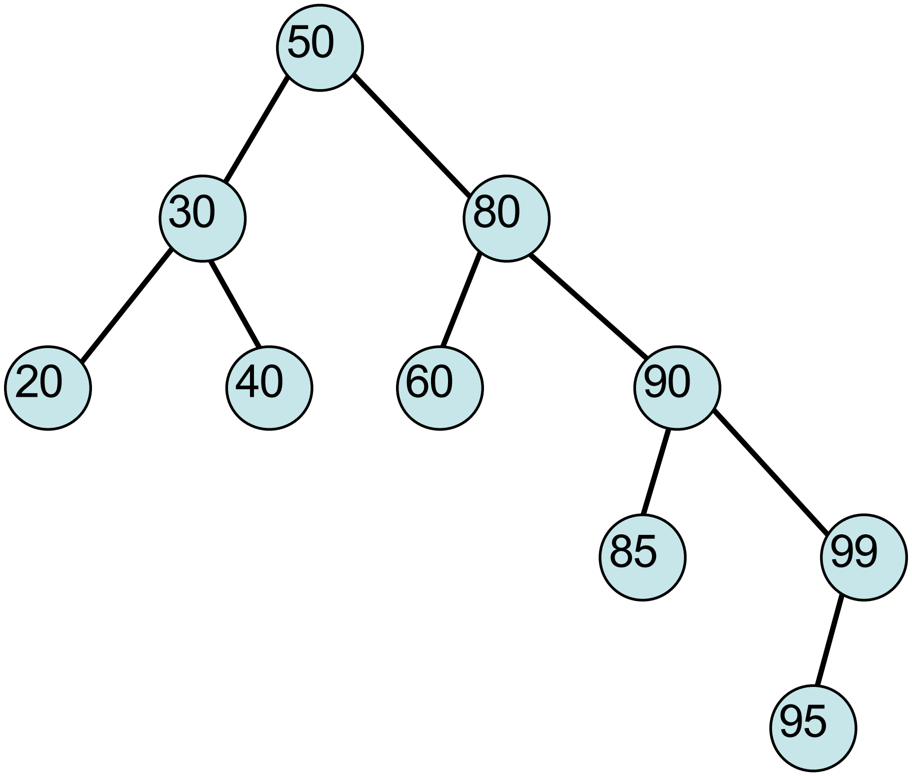
Binary Search Tree - is this tree “height balanced”?



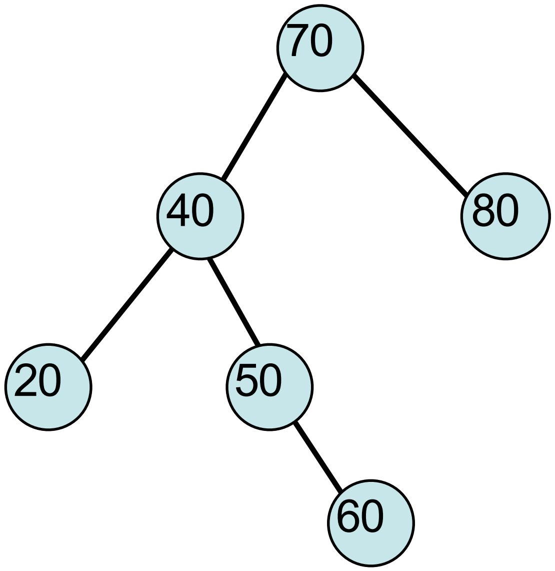




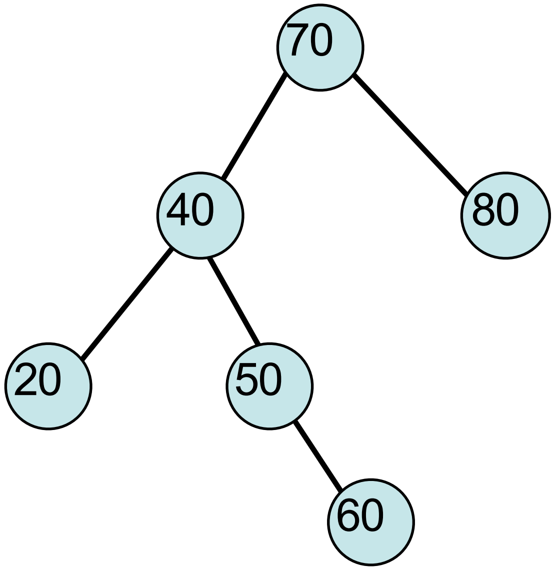
operations on BST - rotations



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:

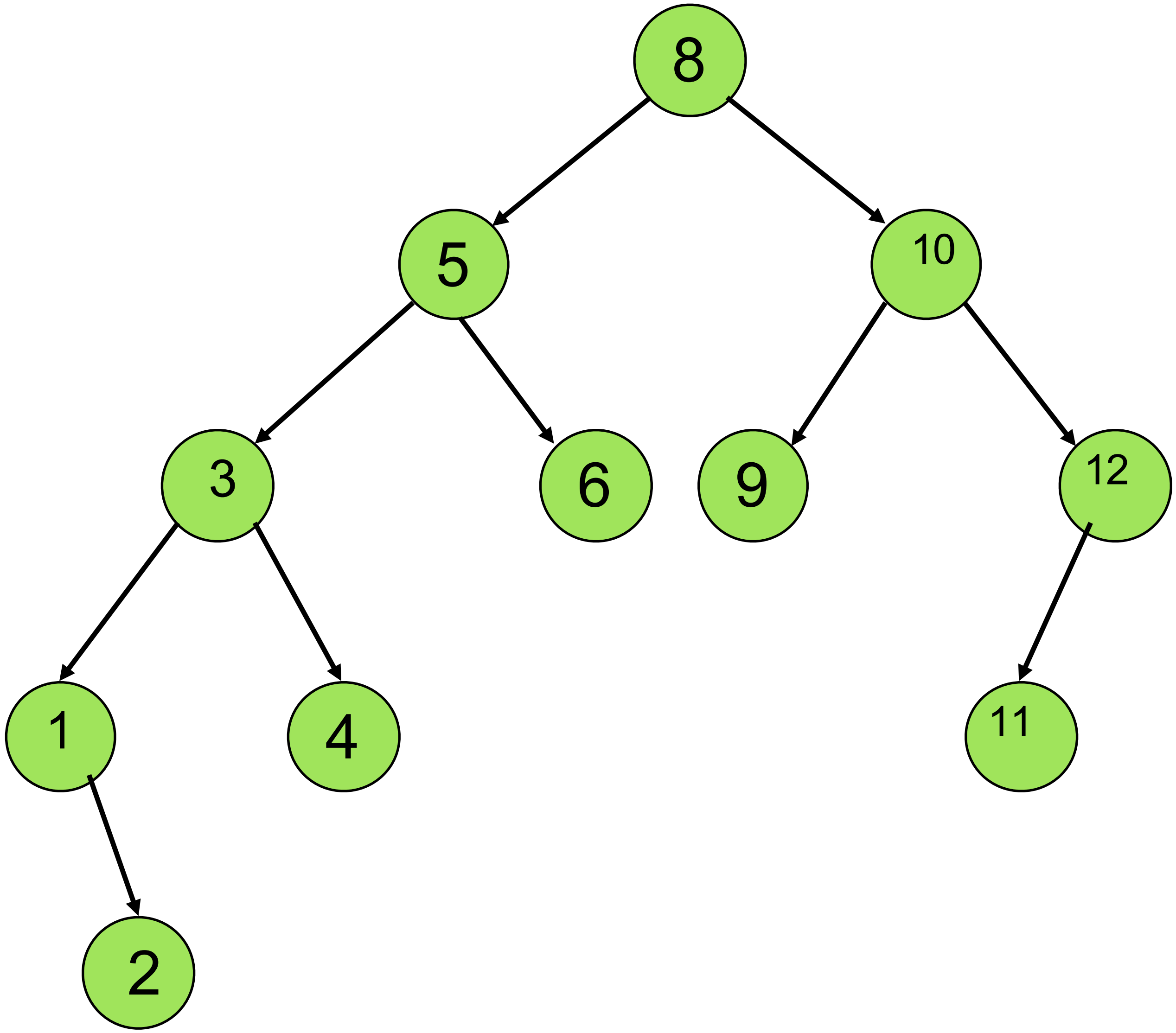
Three issues to consider as we move toward implementation:

Rotating

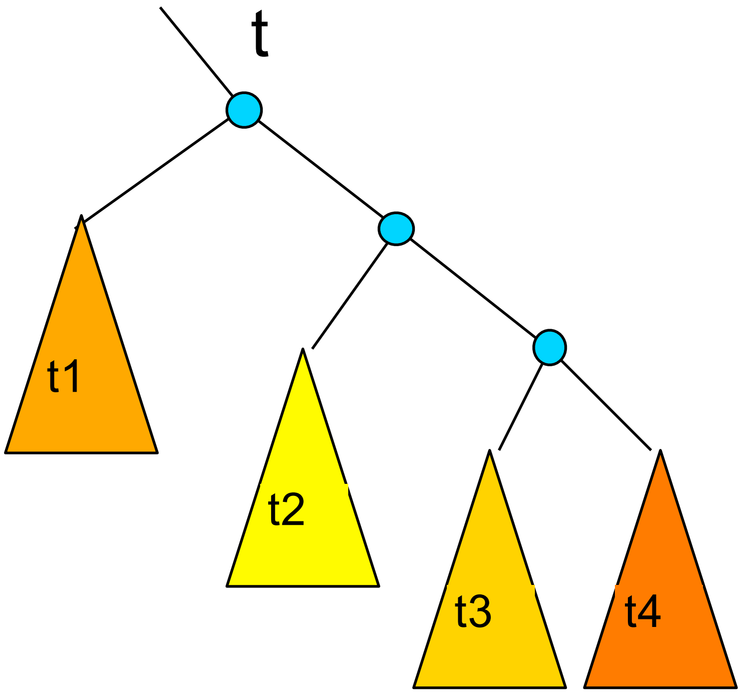
Maintaining height

Detecting imbalance

Maintaining height upon a rotation:



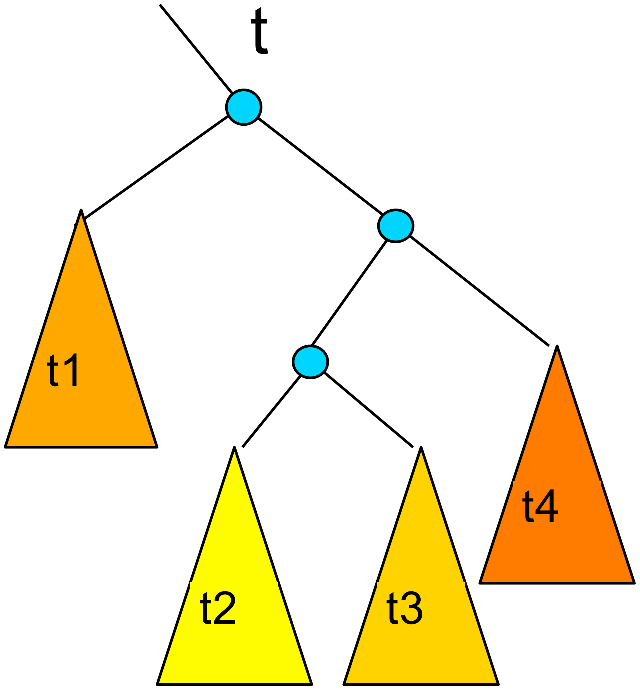
AVL trees: rotations (identifying the need)



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)

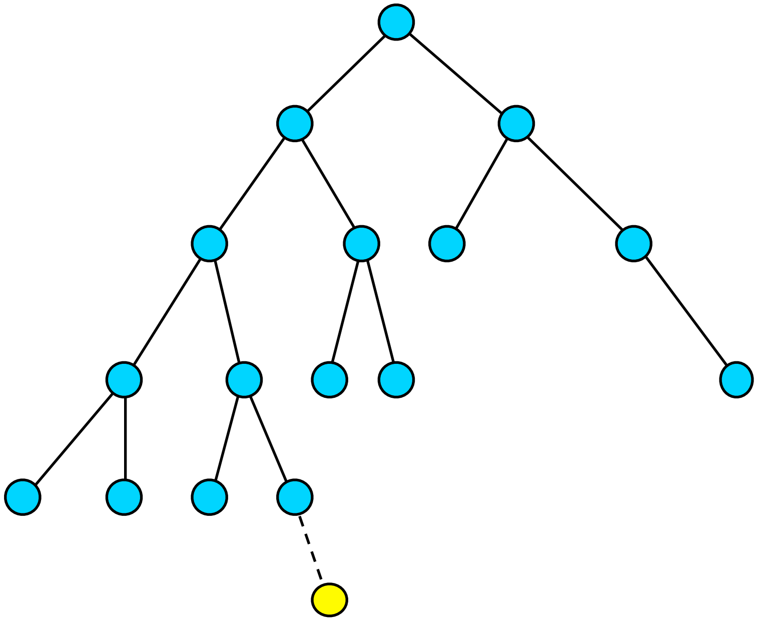


If an insertion was in subtrees t2 or t3, and if an imbalance is detected at t, then a ______________________ rotation about t rebalances the tree.

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;
}
```