Announcements

MP5 available, due 3/29, 11:59p. EC due 3/15, 11:59p.

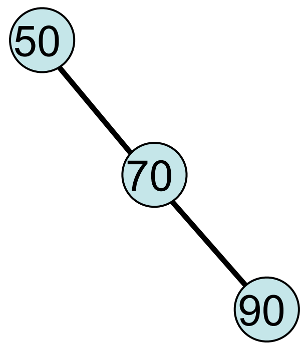
M(n):

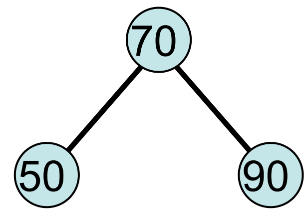
we cleverly observe a recurrence for M(h):

we hypothesize a closed form for the recurrence is: M(h) =

PROVE IT!!

something new… which tree makes you happiest?





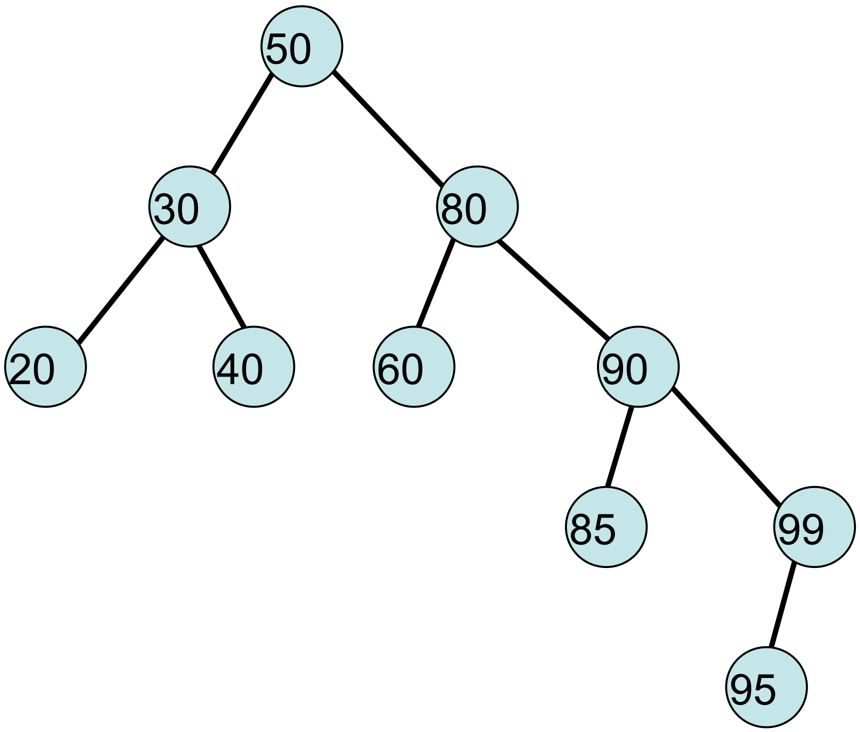
The "height balance" of a tree T is:

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

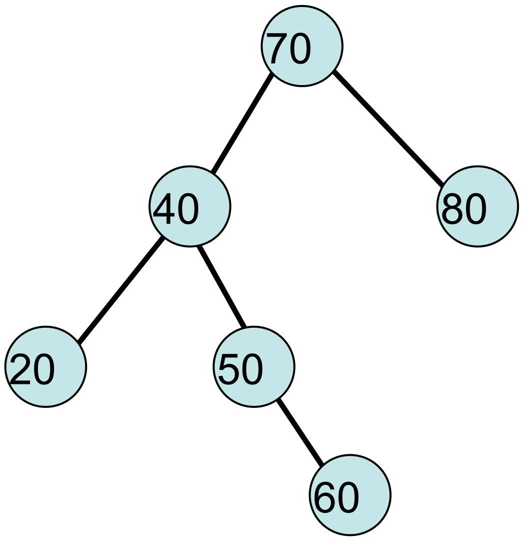
A tree T is "height balanced" if:

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

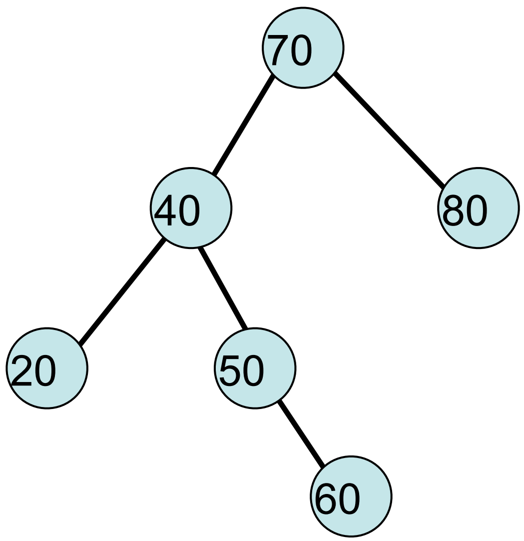
operations on BST - rotations



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:

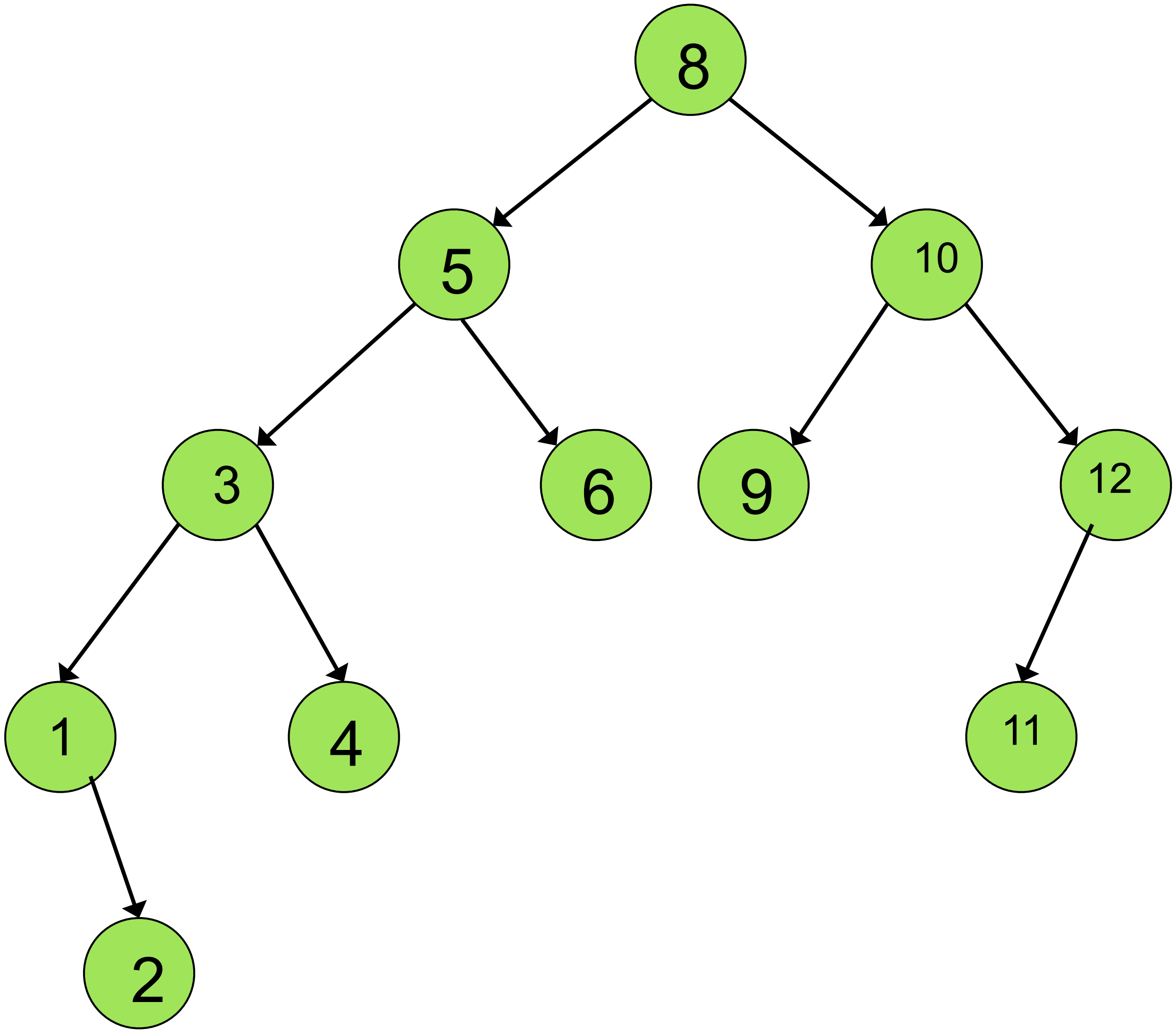
Three issues to consider as we move toward implementation:

Rotating

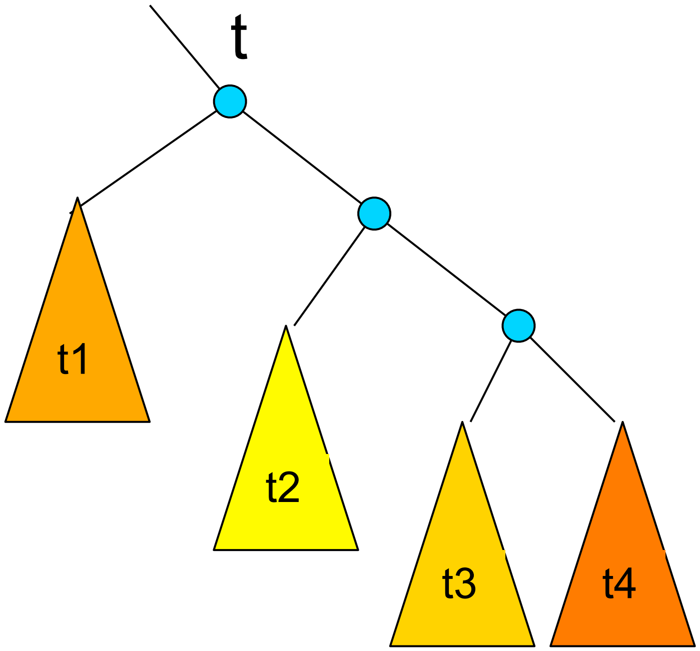
Maintaining height

Detecting imbalance

Maintaining height upon a rotation:



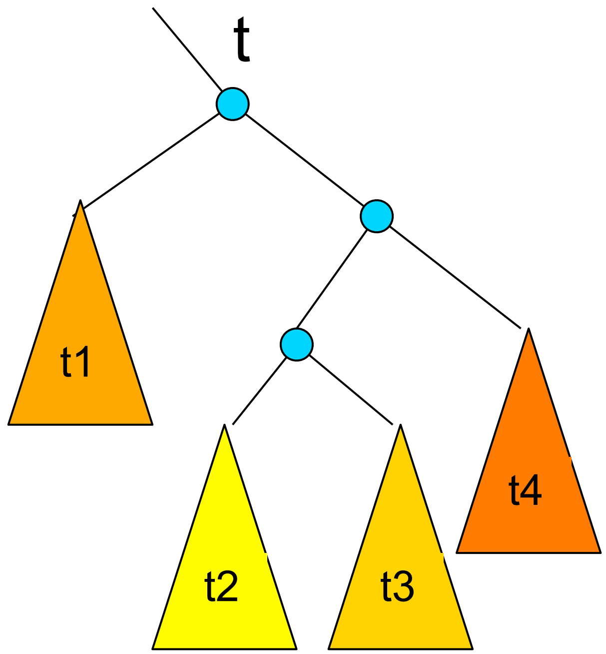
AVL trees: rotations (identifying the need)



If an imbalance is detected at t, and if an insertion was in subtrees t3 or t4, 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)

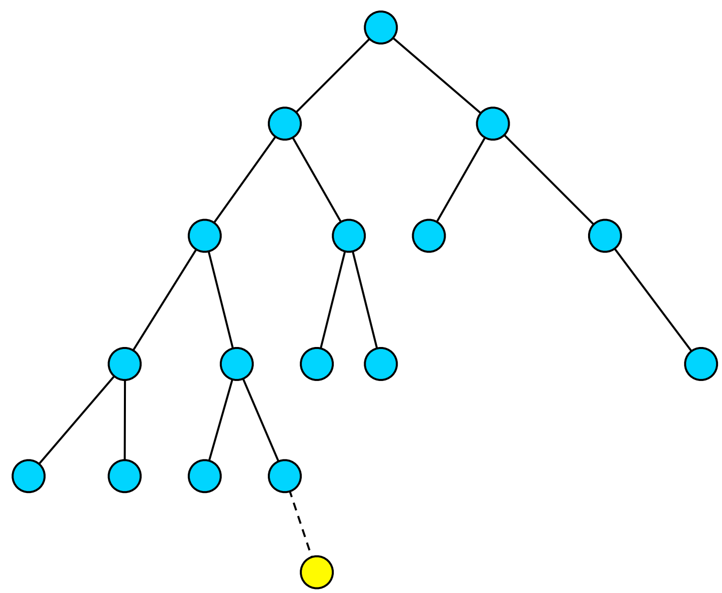


If an imbalance is detected at t, and if an insertion was in subtrees t2 or t3, then a _____________ rotation about t rebalances the tree.

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;
}
```

Warm-ups:

1. Fill in the blanks to give the recursive definition of a BST:

   A binary tree T is a binary search tree (BST) if:

   • T is empty, or

   • T is { __, __, __ } and

   x ___ T_L → key(x) ___ key(r),

   x ___ T_R → key(x) ___ key(r),

   and ____________________

2. Correct the following specification for ADT dictionary:

   insert(key) -> data   traverse() -> data

   find(data) -> key   remove(data) -> void

3. T/F: BSTs have better worst-case performance than SLL

4. How much would you pay for a BST "find" whose running time is O(log n)?

5. What data structure is used to support a level order traversal of a binary tree?