Thursday, October 01, 2009
3:34 PM

- A864...
  - Text's AVL tree works by recalculating the height of a node following insertion
- An alternate approach is to maintain balance info at each node & return the height change following insertion
- Changes to text's AVL implementation:
  - first notice the code can be streamlined somewhat by removal of the double_left & double_right routines (but calls them doubleWithLeftChild, doubleWithRightChild)
  - this with "compact" insertion code

Thursday, October 01, 2009
3:41 PM


```
template <typename T>
int ~~void~~ Tree<T>::insert( const T& x , Node*& t )
{
    int delta_h = 0;
    if( t == NULL ) { t = new Node( x, NULL, NULL ); delta_h = 1; }
    else if( x < t->data ) {
        ~~insert( x, t->left );~~
        ~~if( height(t->left) - height(t->right) == 2 ) {~~
        if( insert( x, t->left ) ) { // height of left subtree increased
            t->bal--;
            if( t->bal == -1 ) delta_h = 1;
            else if( t->bal == -2 ) {
                if( x < t->left->data )
                    rotate_left(t)
                else {
                    ~~double_left(t)~~
                    rotate_right(t->left)
                    rotate_left(t)
                }
                if( t->left->bal == 1 )
                ~~if( x > t->left->data )~~
                    rotate_right(t->left)
                rotate_left(t)
            }
        }
    } else if( x > t->data ) {
        insert( x, t->right )
        || ( height(t->right) - height(t->left) == 2 )
```

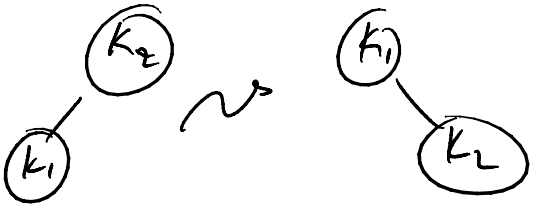
```
    if (x < t->right->data)
        rotate_left(t->right)
    rotate_right(t)
}
```

rewrite

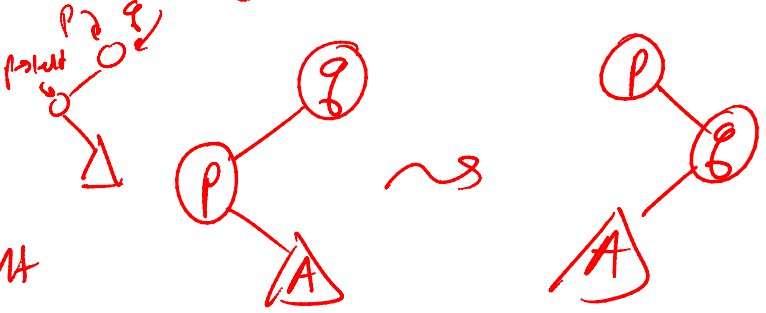
```
else if( x > t->data) {
    if (insert(x, t->right)) {
        t->bal++;
        if (t->bal == 1) delta_h = 1;
        else if(t->bal == 2) {
            if(t->right->bal == -1)
                rotate_left(t->right)
            rotate_right(t)
        }
    }
}
else ; // no-op
t->height = max(height(t->left), height(t->right))+1
return delta_h;
}
```

before:

```
void Tree<T>::rotate_left(Node*& k2)
{
  Node *k1 = k2->left;
  k2->left = k1->right;
  k1->right = k2;
  k2->height = ...
  k1->height = ...
  k2 = k1;
}
```




```
void Tree<T>::rotate_left(Node*& p)
{
  Node *q = p;
  p = p->left
  q->left = p->right
  p->right = q
  q->bal++;
  if (p->bal < 0) q->bal -= p->bal
  p->bal++;
  if (q->bal > 0) p->bal += q->bal
}
```



```
void Tree<T>::rotate_right(Node* &p)
{
  // mirror of rotate_left
}
```

Thursday, October 01, 2009
4:10 PM

Deletion

Alg: if node is a leaf remove it

else replace it with the minimum of the right subtree then erase that node

After deletion, retrace path back up the tree to the root, adjusting the balance factors as needed

e.g.

like insert(x, t→left)

```
{
  t→bal --;
  if (t→bal == 0) delta_h = 1;
  else if (t→bal == -2) {
      if (t→left→bal == 1)
           rotate_right(t→left)
      rotate_left(t)
      if (t→bal == 0) delta_h = 1
  }
```

- the remainder of the delete routine precedes the above in two tests to recurse down to either left or right subtree to perform deletion

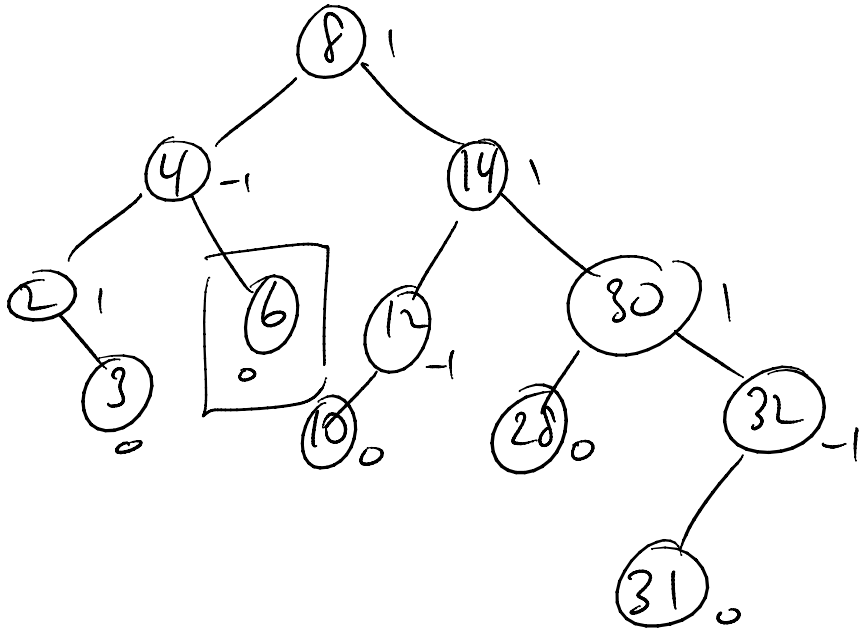
recall the normal BST deletion:

```
if (x < t->data)
    erase (x, t->left)
else if (x > t->data)
    erase (x, t->right)
```

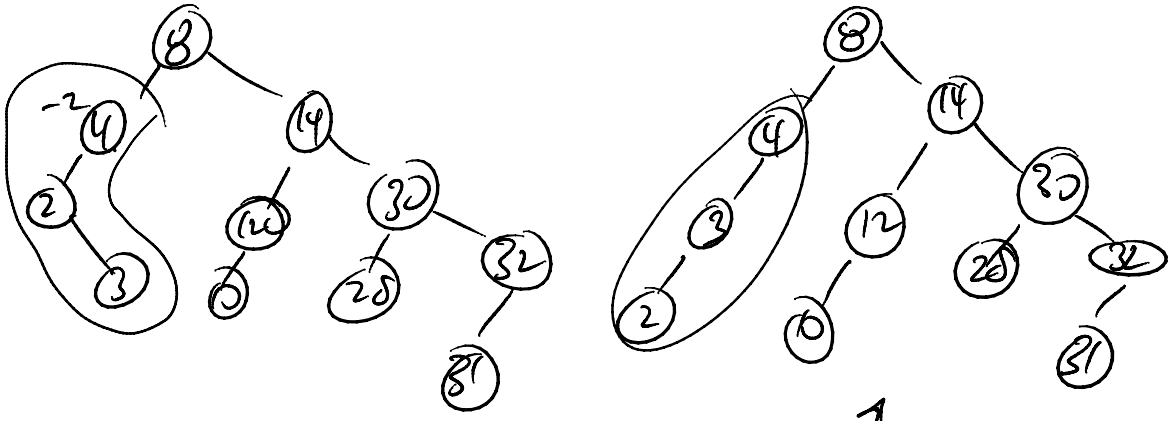
- these now have to be expanded in manner analogous to above deletion of interior node

Thursday, October 01, 2009
4:20 PM

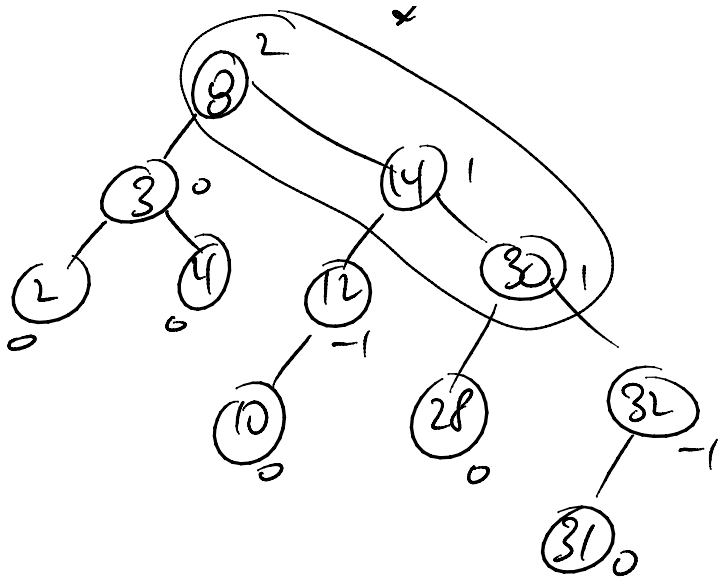
e.g. input: insert 6, 4, 2, 8, 10, 12, 14, 32, 30, 28, 3, 31 :



delete 6:



rotate_right(t->left)
rotate_left(t))



rotate_left(+)