CSE 326
Lecture 14

Spring Quarter 1995

Summary of AVL Insertion

- Iterative Approach

  1. Insert the new node keeping track of the insertion path in a stack.

  2. Pop the stack incrementing the height field of the nodes found until a node n is found which is not balanced or the height field does not need incrementing.

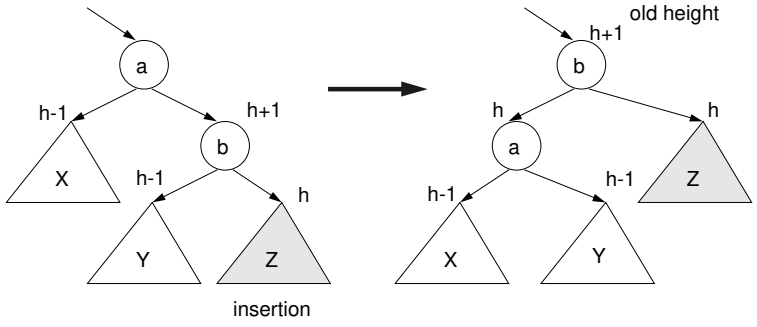
  3. If there is an imbalance then do either a single or double rotation. The rotation will be left or right depending on the direction of insertion.

- Recursive Approach (See Weiss)

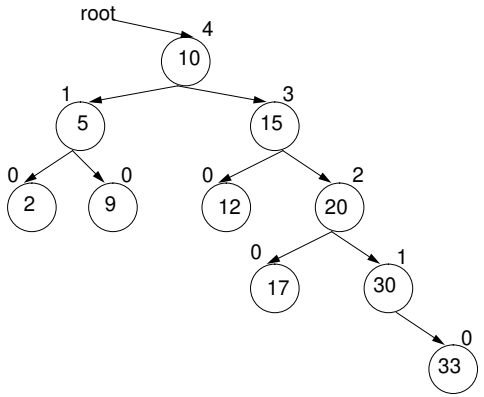
  1. Recursively insert, starting at the root.

  2. Check for imbalance of children, if imbalance then single or double rotate, otherwise update the height field. Such an imbalance will only happen once.

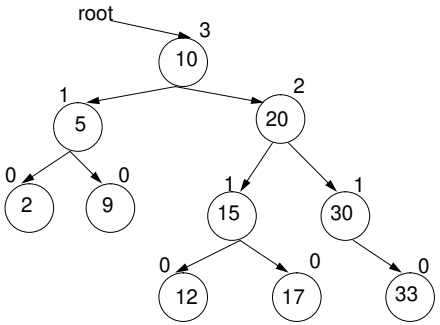
Single Right Rotation in AVL Insertion



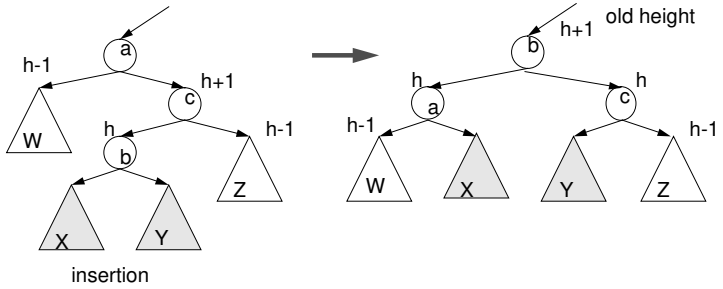
Single Right Rotation Example



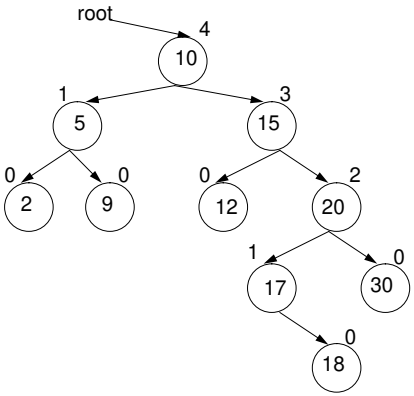
Tree After Single Right Rotation



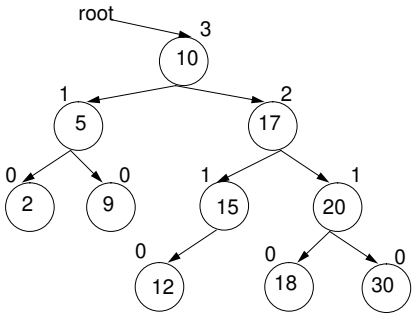
Double Right Rotation in AVL Insertion



Double Right Rotation Example



Tree After Double Right Rotation

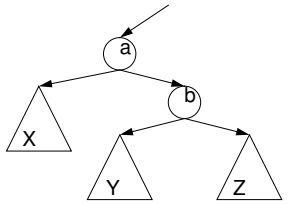


Right Rotation Code

- Single_Rotate_Right(ref n) {

  local node pointer p;

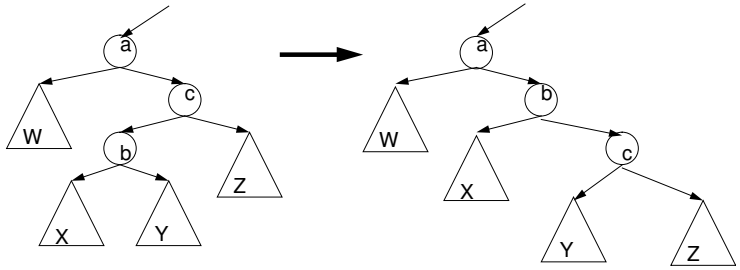
  p := n.right;
  n.right := p.left;
  p.left := n;
  n.height := max(ht(n.left),ht(n.right)) + 1;
  p.height := max(ht(p.left),ht(p.right)) + 1;
  n := p }



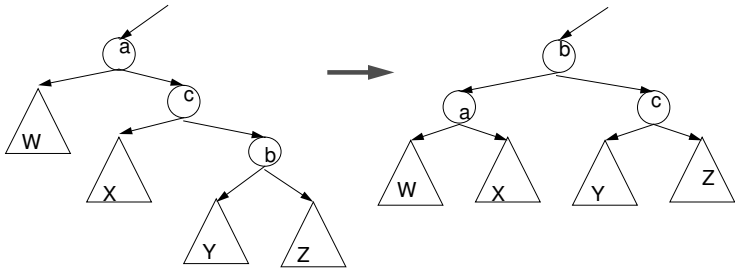
Right Double Rotation

- Double_Rotate_Right(ref n) {

  Single_Rotate_Left(n.right);
  Single_Rotate_Right(n) }



Second Step in Double Rotation



AVL Tree Deletion

- Iterative Approach

  1. Do normal tree deletion all the way to the physical deletion, storing the deletion path on a stack.

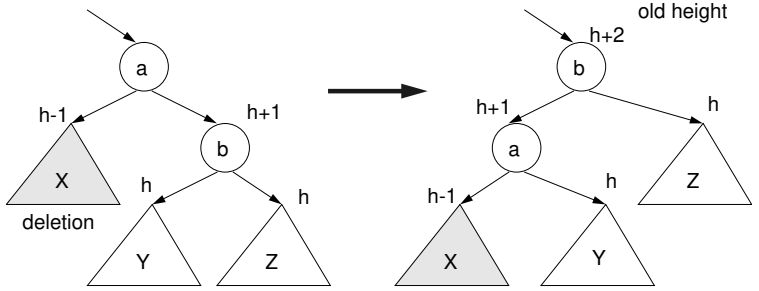
  2. Popping the stack update the height fields looking for an imbalance. If imbalance is found do a rotation. The direction of the rotation will be opposite the direction of deletion. Continue updating height fields if need. Note: more than one rotation may be needed.

- Recursive Approach

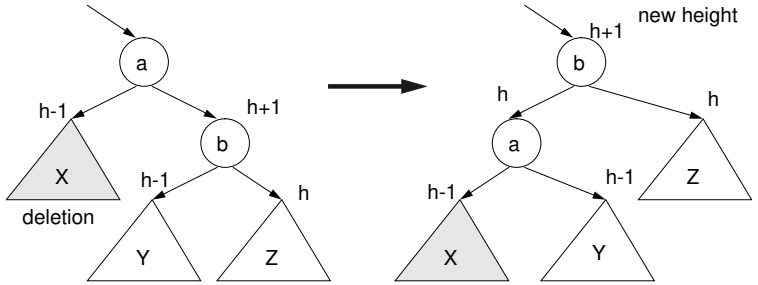
  1. Recursively Delete all the way to the node that is physically deleted.

  2. On return, check for an imbalance. If imbalance do a rotation and update the height fields in all cases.

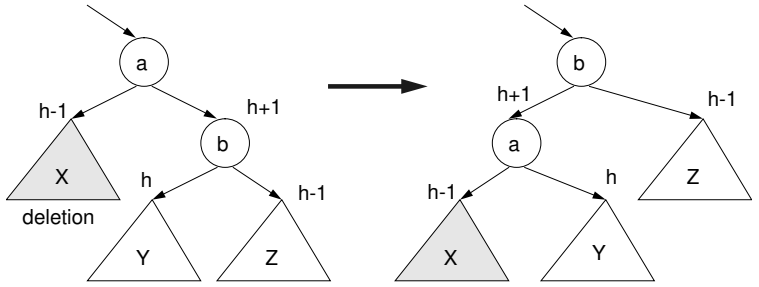
Single Rotation with no Propogation in Deletion



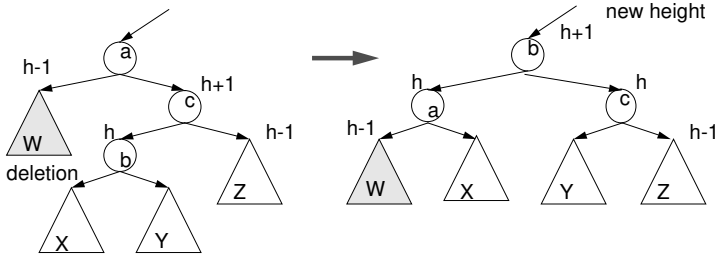
Single Rotation with Propogation in Deletion



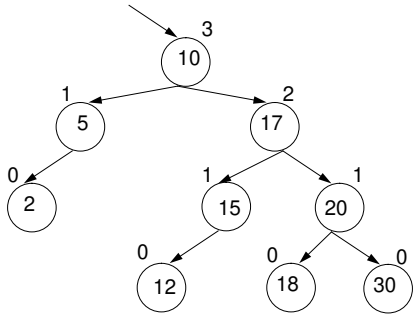
Why Single Rotation is not Enough in Deletion



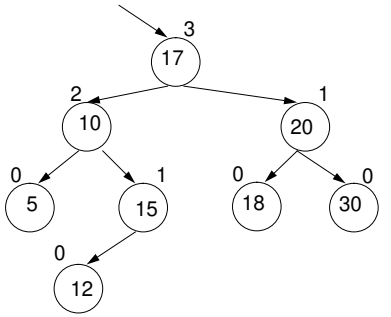
Double Rotation in Deletion



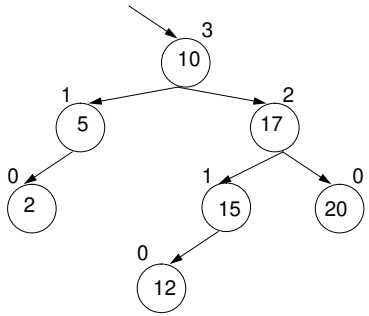
Delete 2



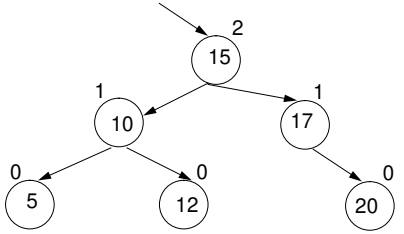
Result of Deleting 2



Delete 2 Again



Result of Deleting 2



Batch Deletion in AVL Trees

- Recall that we simple marked nodes as deleted.
- There are two steps

  1. Do an inorder traversal putting all the unmarked nodes in an array.

  2. Do a divide and conquer construction of a balanced tree, which will have the AVL tree property.
- Suppose we have m deletions in a tree of size n.

  Ordinary AVL deletion time is O(m log n + n)

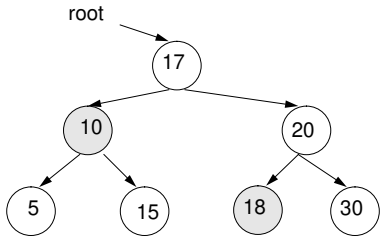
  Batch deletion is O(n)

Listing the Unmarked Items

- List(n,i) puts the items in the tree rooted at n in order in the array A starting at index i.
- List(n, ref i) {

```
if not(n=null) do {
  List(n.left, i);
  if not(n.mark) then {A[i] := n.data; i := i+1};
  List(n.right,i) }}
```

- Call List(root,1)



Creating the AVL Tree

- Assume A[1..N] is a sorted array. The procedure Build(i,j) will return a balanced tree whose nodes have data A[i],A[i+1],...,A[j].
- Build(i,j) {

```
if j < i then return(null)
else {
  k := (i+j)/2;
  return(New_Node(A[k],Build(i,k-1),Build(k+1,j)))  }}
```

- Call Build(1,N) to create the entire tree.
- Fill in the height fields correctly with a postorder traversal.
- Running time is O(n) which is better than inserting items individually, which would be O(n log n).

Example of Creating An AVL Tree

- 5 , 8, 10, 15, 17, 20, 30, 35, 40

Time Complexity of Building a Balanced Tree

$$T(0) \leq b$$

$$T(n) \leq T\left(\left\lceil \frac{n-1}{2} \right\rceil\right) + T\left(\left\lfloor \frac{n-1}{2} \right\rfloor\right) + c$$

- We can prove by induction that T(n) < (b+c)n + b.

$$T(0) \leq b = (b+c)\,0 + b$$

$$T(n) \leq (b+c)\left\lceil \frac{n-1}{2} \right\rceil + (b+c)\left\lfloor \frac{n-1}{2} \right\rfloor + c = (b+c)\,(n-1) + c = (b+c)\,n + b$$