Data Structure

Part 14

Inserting in AVL Tree

We are going to talk about insertion in AVL trees. As you all remember, when do we do insertion into the AVL trees? We need to check the balance factor and if balance factor does not satisfy the AVL property, the tree property, then we have to rebalance. For doing this rebalance, we are also going to talk about the different types of rotation. In this lecture, we are basically going to talk about the AVL Insertion. Again as we already know, the AVL trees are balanced trees and we are going to look at how insertion will cause rebalance and how you detect the tree balance and how we balance it. So, insertion at the leaf. First, we look at insertion. In this lecture, we are going to only talk about the insertion and we are going to talk about the insertion in AVL trees. Please remember that there are two aspects of this: one is the definition of the AVL tree; remember the balance factor in the AVL tree, that is, a difference between the height of the right sub tree and left sub tree should be no more than 1. That is one of our conditions and the second is AVL tree is basically binary search tree; it obeys all the properties of the binary search tree. Let's look at the insertion at the leaf. The insertion at the leaf is, as is the case for all Binary Search trees, only nodes on the path from insertion point to the root node have possibly changed in height. This is the way the insertion is done because of the binary search property. After the insertion, what we have to do is we have to go back up to the root node by node; we have to update the heights. If a new balance factor, (the difference is h left –h right or h right or –h left) is 2 or -2, then we have to adjust the tree by rotation. This is the insertion. If you do the insertion at the leaf, in the case of binary search tree, the insertion at the leaf is very simple and you have to search for where you have to insert and then you have to insert; that remains the same. The problem is when you do the insertion at the leaf, it is possible that the height of other nodes of the tree will change and the balance factor may become 2 or -2. In which case we have to adjust the tree by rotation around the node.

This rotation is the operation that we use in order to height-balance the tree or rebalance the tree. Insertion is similar to the regular binary search tree. You keep going left or right in the tree until the null child is reached. Insert a new node in this position. And inserted node is always a leaf, to start with. This is how you search for the node; if you have the node that is greater than the root, you have to search in the right sub tree; if it is less than the root, you have to search in the left sub tree and in that sub tree root again, you check and go on till you find a position for the new. The major difference between the binary search tree and the AVL tree is after you do the insertion, you must check if any of the sub trees in the tree have become unbalanced. Search from the inserted node to root, you have to go backwards looking for any node with has the balance factor of + or - 2. If you have a node of balance factor with + or – 2, it means that we have to do the height balancing at that particular node. And a few points about tree insert; the insert must be done recursively. The insert call will return true if the height of the sub tree has changed. Since we are doing the insert, the height of the sub tree can only increase obviously. The height of the sub tree can increase during the insert operation and only can decrease during the delete operation. Inserting into the AVL tree if insert returns true, the balance factor…. that is the way we have written the algorithm the balance factor of current node needs to be adjusted. That is, if we are doing the insertion, then we have to do a balance factor recalculation. The balance factor = height right – height left. If the left sub tree increases, the balance factor will decrease by 1. If the right sub tree increases, the balance factor will increase by 1. So this is the way the balance factor has been done. If you do the insertion and the left sub tree increases, the balance factor will decrease. If you have done right sub tree, the balance factor will increase by 1. In doing so, (now you have to do this at each and every node right up to the node), if the balance factor equals + or - 2 for any node (this is very important), that particular sub tree where you have found the balance factor to be + or – 2, that has to be rebalanced. Now let's look at insertion and rotation in AVL trees. In order to do that tree balancing or height balancing, as it is called in an AVL tree, you have to do the rotation operation. So, insert operation may cause balance factor to become 2 or -2 for some nodes which we have detected, as we said previously. Only the nodes on the path from that insertion point to the root node could have possibly changed. This again we have discussed. That means, if you insert the node on the left-hand side of a node, only the path from that left sub tree up to the root will change in height and the other path of the tree does not change in height. After the insert, we have to go back to the root node by node and then update heights and if the new balance factor, as I told you before, is 2 or -2, adjust tree by rotation.

Rotations

Now let us look at what we mean by rotation operation; we have been telling rotation. What do we mean by rotation? To keep the balance of an AVL tree correct, rotations must be performed when the tree becomes unbalanced. The operation that is performed to change an unbalanced tree back to a balanced tree is the rotation operation. We call the rotation operation only when the tree becomes unbalanced. Otherwise, the insertion operation we do not call necessary rotation operation. We have already said that it will be performed when the tree become unbalanced.

There is a specific type of rotation to perform for each different way that the tree can become unbalanced. This is very important. It is not that the rotation operation is a single type of rotation; you have four different types of rotation operation. And the four different types of operation are called for, depending upon the different ways in which the tree becomes unbalanced. Since an insertion /deletion involves adding/deleting a single node, this can only increase or decrease the height of some sub tree by 1. Thus, if the AVL tree property is violated at a node A, it means that the heights of left (A) and right (A) differ by exactly 2. Rotations will be applied to this node A to restore the AVL tree property. And again I repeat. When an AVL property is violated at node A, it means that the height of the left sub tree of A and the height of the right sub tree of A now differs by more than 1. Which differs exactly by 2? There is no question because every time it goes beyond 2, it is rebalanced to 1, and it differs exactly by 2 and the rotations must be applied at this node A to restore the balance property. When the tree structure changes, we need to transform the tree to restore. This is done using single….. Now, we are going to talk about rotations. First, we divide the rotation into 2 types; 2 categories; then within that, we can see the different categories. First we talk about single rotations and double rotations. This is the case of the single rotation; this is the common methodology that we use to define a sub tree. This doesn't mean this sub tree contains only a node A and it can have any number of nodes. This is sub tree A; this is the sub tree B; this is Y, this is X and this is the sub tree. This is what happens before rotation.

If you look at this before rotation, the height of this tree and the height of this tree have the height of more than 1. That is the height difference of more than 2. The height is actually calculated like this; this is the height; the height difference between these two is 1. The height difference between this node and whatever node is 2. This is the reason why this is an unbalanced tree. Now it is unbalanced at X. We have to do some rotation. What we do is this is at A node we are talking about; this is rotated like this and Y comes here; now if you look, it is height balance; here there is height imbalance; here it is perfectly balanced in this case; I mean the height difference is balance factor 0 for this. If we look here what we have done is we brought X here; that is one change; it is rotated and Y has come here like this in the rotation, if you look and in addition, the right sub tree of Y has now become the left sub tree of X. Let us assume this node is X. This node is Y, and X has become unbalanced; you will rotate it such that X comes here and Y comes here and the right sub tree of B now becomes the left sub tree of X. This is called single rotation.

Just to give an idea of rotation, I have explained that. Now there are four types of case that we have to consider for the unbalanced nodes and consequently there are four types of rotations. As I already told you, there are two categories of rotation and one is single rotation and another double rotation. Within single rotation, you have single right rotation which is called Right-Right Case. Use left rotation to balance the node. Then, you have the Single Left Rotation or the Left – Left Case where we use Right rotation to balance the node. Then you have Double Left Rotation which is called Left –Right Case and use left right rotation to balance the node. Then you have the double Right Rotation or the Right –Left Case, and use right-left rotation to balance the node.

So these are the four different types; two single, two double. This is Right and this is Left rotation; this is Left-Right Rotation and this is Right-Left Rotation; so you have four cases. We are going to look at each of these four cases separately. And then I will explain how this balancing is done.

Single Right Rotation

Now let us take the first case which is the single right rotation. Let us revisit the definition. In the single right rotation you use the left rotation to balance the node. In the Right-Right case, you balance the node using the left node notation. Now the sub tree is rooted at 5 and if you look here, the balance factor is 0. If you look here, the balance factor is -1 and if you look here, the balance factor is -2. It is a Left-Left Case and you have to do a Right Rotation and this particular node has to be balanced. All nodes are now balanced in this tree. What have we done here, I explain this again to just give you a feel; this node comes here; this node comes here and this is what we have done. If you just look at this, what happens to the sub tree we see later; what we have done is we rotated like this This 3 comes here, this 5 comes here and the B node, the node that has been attached here will be attached. This is what we have done after this rotation. So, A is unbalanced to the left; B's left sub tree is bigger than its right sub tree; it is left heavy; this is the condition for performing the single Right Rotation. I will repeat; A is left unbalanced to the left. Where is the unbalanced? Whether in the left sub tree or in the right sub tree and we also have to know what happens to the B's left sub tree, that is the tree we are going to replace with the bigger than the right sub tree, left heavy. Then, we have to perform single right rotation. These are the two conditions which tell you what type of rotation we have to perform. We are talking about the rotation and we have taken the first case of the single right left rotation. I have already told you the condition under which we do the single right rotation. Now I am going to tell you what happens exactly during the single right rotation. Let us assume that this is your tree that you have and what happens here is if you see, here the balance factor will be 1; here the balance factor will be 1 again because the height difference between these two is 1. Now, you should take the balance factor…. here the height of this tree is 2 and the height of this tree is 3 and the height of this tree is 1. So 3-1. Here you have an unbalanced node A because of the height difference being more than 2 between the left and right sub trees. Now the node has its left child, you take the node and unbalance it at A. When you have the unbalance at A, remember the condition we said A is unbalanced to the left. Here is the unbalance problem. Now you take the left child of A which is B and you have to do the rotation; this is called the unbalanced node, this is called the pivot; this is the pivot node which you choose for doing.

Actually what we are going to do is, we are going to bring B here; we are going to bring A here; so this unbalance is due to the left sub tree of AB heavy and so the left sub tree's root node B is taken and called the pivot. And now N becomes the right child of that pivot, N left. What happens is this node which was unbalanced now becomes the right child of B. And N left's right sub tree becomes the left sub tree of N. I come again to this node; what happens is this is the pivot and what you are doing is the right sub tree of this is taken and becomes the left sub tree of A. That is the single right rotation. I will repeat it again with the example here. What happens; this is the unbalanced node, this is n left which we have described as the pivot, and this is N left, the right sub tree; it need not be a single node, it can be a tree. What we do is we do the rotation. When we do the rotation we bring B here and bring A here. That is the rotation. But in addition to doing that rotation, what we have to do is we have to also take the right sub tree of B and make it the left sub tree of A. This is called the single right rotation. Now let's go to the second case which is single left rotation. As you know the single left rotation; again, let's take a case; the same example we have taken except that now it gets unbalanced on the right side. I think you already know how this is unbalanced and I am not explaining that. But the unbalanced has node 3. After the balancing, the rotation we have done… if you look here, in this direction. In the previous case, we did it like this and now we are doing like this. This is the balance tree, after the left rotation. How this is done? Let's look at that. This is the left rotation shown in the other format; it's heavy here; it is doing this rotation and you get this. This is the left rotation; so the algorithm for doing this is rotating left and node N and now you have to find the pivot; the pivot now will be because it is right heavy and N right. N becomes the left child of N right. Previously what happened N was becoming the right child of N left and now N is becoming the left child of N right and N right left sub tree becomes the right sub tree of N. Just opposite of the previous one; and we repeat that we will see this is unbalanced; this is your N right. Now we are going to do the rotation like this. This comes here and the left sub tree of N right becomes the right sub tree of N right the right sub tree of N… this is your N. Please note these values are the balanced factor; so it is unbalanced at this and you have to find the pivot node where it is heavy; this is rotated; V becomes R, R becomes the left child of B and then the left child of N right now becomes the right child of N. So, this is the rotate left. This is the exact contrast or opposite of what we did in rotating it. Those are the two easiest cases: that is, single right rotation and single left rotation.

<u>Double Left Rotation</u>

Now let us take the next case which is double left Rotation. Please note that in the two cases what we had… this can happen anywhere in the tree; it is not necessary in the root. You consider the root just for simplicity sake. When a node has the balance factor of 2 and the left or right child is heavy, then we did this. Now we have to look at double left rotation. This is just to understand this. Now here you have a balance tree and here you have this and here you have 5. So the sub tree is rooted at 5. And the balance factor of 5 is… this is the unbalanced node. Now what we have to do is we just don't do a rotation like this. Now this is the pivot. In the previous cases this would have been the pivot. Now we are changing the pivot completely and all nodes are now balanced and now pivot goes to the unbalanced node; this goes here; this comes here; it is great. Please, remember here the binary search tree property; 4 come here, 3 come here and 5 come here; so we are doing a rotation in a different way. Actually what happens is this is going here and this is coming here; you are having both directions. This was here that has come up in this direction and 5 here comes in this direction and that's why it is called the left right rotation. If you look at this, this is your pivot node; first you do this rotation, that is, you take this here and rotate it; that' is what we have done here and then what you do is again you do the right notations; what you do is you take the k3 and put it here and k2 comes here. That's why it is called double; we are doing the one left rotation and one right rotation and I repeat again with the previous example. What you did here is you took the 4 here and 3 here and so this direction is left rotation. And then what you did for that, you again did the right rotation and that's how you got this. I again repeat this was the pivot 4 in the previous example; this is your 3; this comes like this; that's your left and then you have to do a right rotation which is this and that is what happens. You have a left right rotation when you add T7 to this tree A becomes unbalanced. When A becomes unbalanced, you have to do a left-right rotation. This you rotate first and then you rotate, B would have come here and you rotate like this; that's the left right rotation and A is unbalanced towards the left and this is what is here and then B's sub tree is bigger than its right sub tree. B's left sub tree is heavier than its right. True. And C's right sub tree is heavier than it's left; again true. And this means we must perform a double left rotation. After performing the rotation, the tree is balanced as given. That is the double left rotation and now we will see the double right rotation. Again in the previous case, as you know, this would be the pivot. What you do you will do a right rotation first and then does a… what happens 4 will come here. Then you will do a left rotation and so it is called the right rotation and you can guess now this is your pivot. With the pivot and its parent node we will do a right rotation. And this comes like this and this comes here and so you have 4 here. Then you do a left rotation. And the 4 comes up and 3 moves down and that is what happens. And this is the right left rotation and again, as in the previous case, we look at it in this way and what you will have to do is first you will rotate in the right direction. So k2 comes here and k3 comes here and then you will do a left rotation. When this was k2, k2 will come here and k1 will come here. Right rotation on k3 and left rotation on k1 and that is how you get this balanced. If you take this, suppose T7 is added, A becomes unbalanced; B is your pivot and first you have to do a right. This will be like this and then when B comes here, you will do a left. A is unbalanced to the right and again, as in the previous example, I will explain. A is unbalanced towards the right, as seen and B's right sub tree is bigger than its left sub tree. B's right sub tree is heavier than its left. Why B's? Because B is the pivot. C's left sub tree is bigger than its right. Again lets go back. C's left sub tree is bigger than its right. C is the middle node and that's how you keep track of that. This means we must perform a double right rotation. Please note these conditions are when you have to form what? Previously what I have told you was how to perform. What you will have to look at when the unbalanced comes? Look at the node, the pivot and find out which is the pivot node, find out which is the middle node and find whether these conditions are satisfied. If these two conditions are satisfied, then you have to perform a double right rotation. Similarly when we have a…. what are the two conditions? Pivot's right sub tree should be right heavy. Pivot should be right heavy, and I suppose you know what the definition of right heavy by now is. Right heavy means right sub tree is heavier… heavier means more height than the left sub tree and C's the left sub tree is bigger than the right sub tree. Similarly in that other one, we had other conditions which we have to check.