**AVL TREES**

As we had some balancing problems with the binary trees, now we have AVL trees that offer a solution balanced based trees by suitable restructuring during insertion or deletion, so that the basic operations insert, search and delete are efficient with Olog(n) even in worst cases

**Basic principle of self-balancing trees**

1. Degeneration of tree structure in case of unfavourable insertion or deletion   
   operations is avoided by restructuring the trees from time to time
2. There is some specific criterion tat indicates if a tree has to be restructured

For self-Balanced trees, a trade-off ust be made between the effort required to restructure the trees and the impact on the structure and subsequent operations

**Problem**

1. If the trees were restructured optimally each time when a value is inserted or deleted, so that the height is minimal, the large effort required to do so would result in an inefficient overall behavior.
2. If the trees were restructured too little or too infrequently so that the height did not remain within suitable limits, the overall behavior would also become inefficient.

**Using the AVL trees**:  
  
In the case of AVL trees, a solution to the balancing challenge has been found where, on the one hand, it is easy to see if restructuring is required, and, on the other hand, restructuring can be accomplished efficiently while still guaranteeing that the height of the tree (and thus the running time of the operation that depends on it) remains within order of magnitude Θ(log n) even in the worst case

**What is a Balance Factor of Nodes:**So using the balance factor we decide if we have to restructure the tree or not

The balance factor Bx of a node x in a binary tree is the difference between the height of the left subtree and the height of the right subtree of the node. Bx = height (tleft) - height(tright) where tleft is the left subtree and tright the right subtree of x.

Question: Do we have AVL trees only in the case of Binary tree in the exam

A screenshot of a math problem

Description automatically generated

Range balance factor should be from -1 till 1  
Since for all nodes the balance factor is in the range from -1 to +1 and also the search tree property is satisfied, is this an AVL tree  
  
Question: Can there be a AVL tree without the search tree property?  
**Found the answer**

A close-up of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

For example for node 7 its (4-2)=2  
Longest path or height of the tree from left and subtract from the longest path or the height of the right tree

**Height of AVL trees**

For the height of an AVL tree consisting of n nodes always is true:  
**h < 1,44 ld(n+2),**   
i.e. **h = Θ(log n).**

**Search in AVL Trees**

Snice AVL trees are binary trees, searching is done the usual way. The height of the tree has a decisive influence on the search time, snice the number of necessary comparisons is limited by the height of the tree.

The runtime complexity of searching in the AVL tree is of order O(logn)  
In the worst case(and also in the average case)

**Insertion into the AVL Trees**If the structure of an AVL tree is changed by inserting or deleting a value:  
The balance factors must be checked and, if necessary, it must be ensured by restructuring that the tree is still AVL-balanced afterwards.

* 1. To restablish the AVL balance, if necessary, so-called rotation operations will be used. These rotations are local restructurings in a search tree that preserve the search tree property
     1. A diagram of a triangle with arrows and arrows

        Description automatically generated
        1. Question is those rotation operation is only applicable on three references to the subtrees
        2. As far as the Prof. mentioned when you have a balance factor problem or problems in one tree, the way to solve it by start to fix from the lowest level point where a node has a balance factor related issue and then processing and fixing all the nodes linking to that node below which just got fixed or?
     2. A diagram of a tree

        Description automatically generated
     3. A screenshot of a computer

        Description automatically generated
        1. Why is it required to rotate for the second time even though balance factor has all fine with respect to all the nodes or, is it just and example.
  2. Algorithm 7.32 – Insertion into the AVL trees
     1. Search the insertions position as in the binary search trees
     2. Insert new key as a leaf
     3. Perform these step by step from insertion position back to the root node
        1. Update the height of the subtree and check the balance factor of the nodes
        2. At the first node reached with balance factor is 2 or -2 preform a suitable correction by rotation operations (See below) to restore the AVL property
        3. **Exercise 7.34 - Insertion into AVL tree**A screenshot of a computer

           Description automatically generatedUnderstand in the case where we need double rotation, i know its in the case of zig zag  
           “This zig zag rule is confusing sometimes, cause the problem is at node 2  
           Maybe because we added 4 so from the lowest point till the node where the problem exists, we need to do the rotation or?

that is why we generally have double rotation” ----- Verify

**Deletion in AVL Trees**

Not like binary trees deletion, because when deleting nodes with two children, the balance factor may change very much, so that it would not be possible to retore the AVL balance efficiently. Deleting nodes with two children must therefore be done in a different way

1. Search the node k to be deleted and keep track of the path from the root of k.
   1. If node is k leaf or has only one child:
      1. Delete k as in a binary search tree (case 1 or case 2 of Algorithm 7.21)
   2. If node k has two children
      1. Find the maximum v in the left subtree of k (immediate predecessor of k)  
         Copy contents of v to node k. Delete old node with value v
2. From the deleted node go back the complete path to the root, update and check the balance factors of the nodes
   1. If the balance factor of a node is +2 or -2 reestablish AVL balance by single or double rotation, analogous to insertion (Similar to insertion – details see later)
   2. Always go completely back to the root, restructurings at several positions might be necessary.

A diagram of a tree

Description automatically generated

**I mean i understand that if i move 0 and rotate to the right then shouldn’t that solve the problem is some cases, in this case at least doing that is wrong because 1 is greater than 0 so, 1 should not be on the left side, like this, and that why this method is wrong and that is why i should follow those cases, something to do from the lowest to the root node,**

**A screenshot of a computer

Description automatically generated**