DATA STRUCTURES AND ALGORITHMS

LECTURE 14

Marian Zsuzsanna

Babeș - Bolyai University
Computer Science and Mathematics Faculty

2017

In Lecture 13...

- Binary Search Trees
- AVL Trees

Today

1. AVL Trees
2. Final Exam

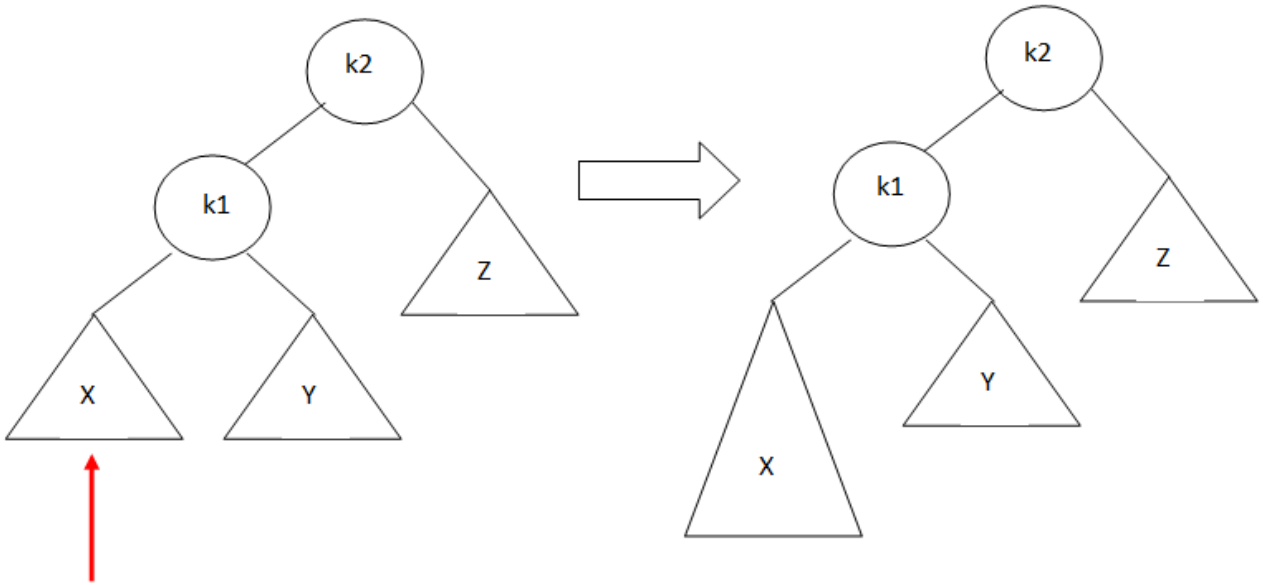
AVL Trees

- Definition: An AVL (Adelson-Velskii Landis) tree is a binary tree which satisfies the following property (AVL tree property):
  - If x is a node of the AVL tree:
    - the difference between the height of the left and right subtree of x is 0, 1 or -1 (balancing information)
- Observations:
- Height of an empty tree is -1
- Height of a single node is 0

AVL Trees - rotations

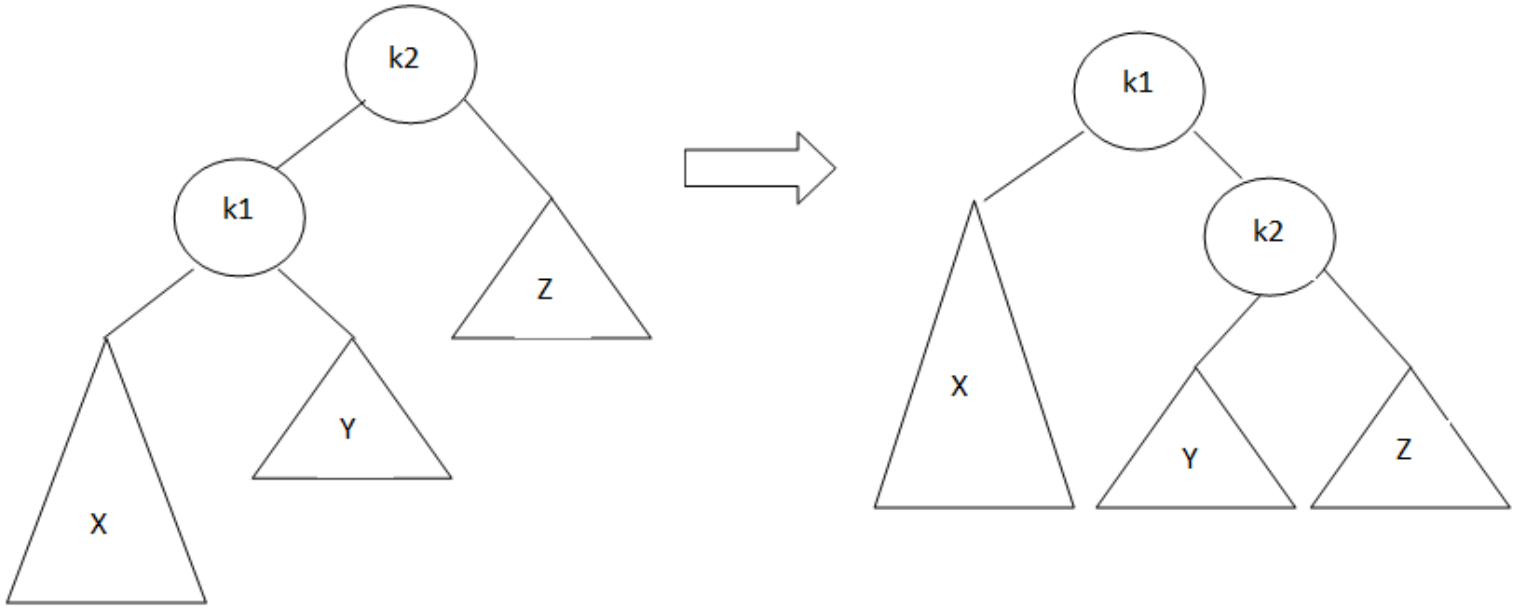
- Adding or removing a node might result in a binary tree that violates the AVL tree property.
- In such cases, the property has to be restored and only after the property holds again is the operation (add or remove) considered finished.
- The AVL tree property can be restored with operations called **rotations**.

AVL Trees - rotations

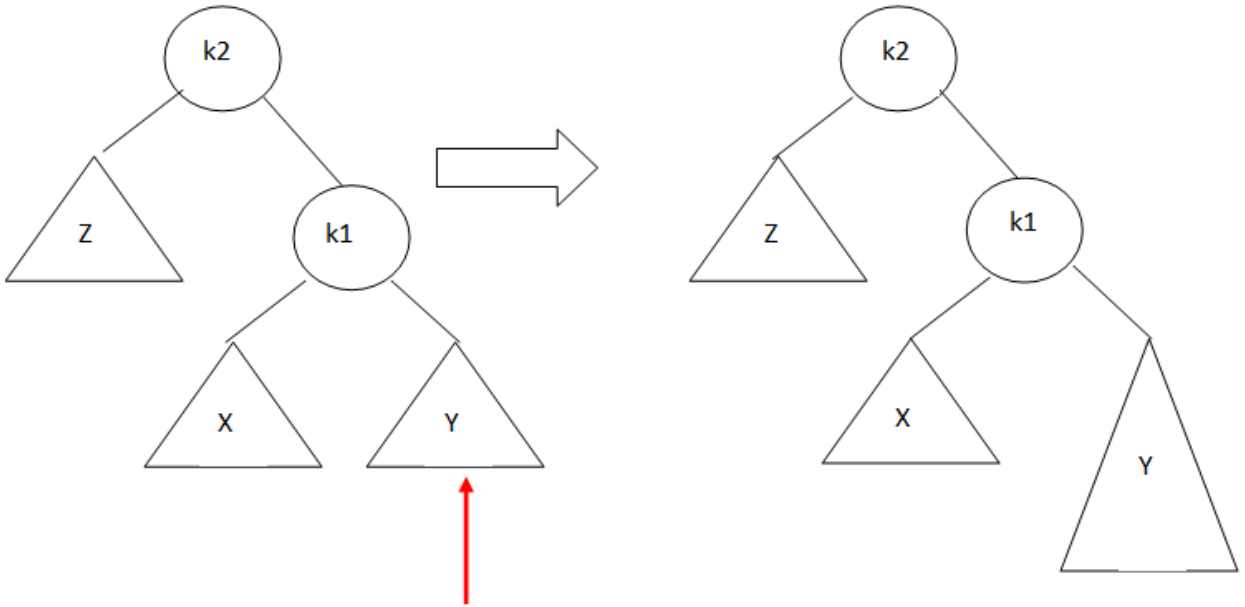


- Solution: **single rotation to right**

AVL Trees - rotation - Single Rotation to Right

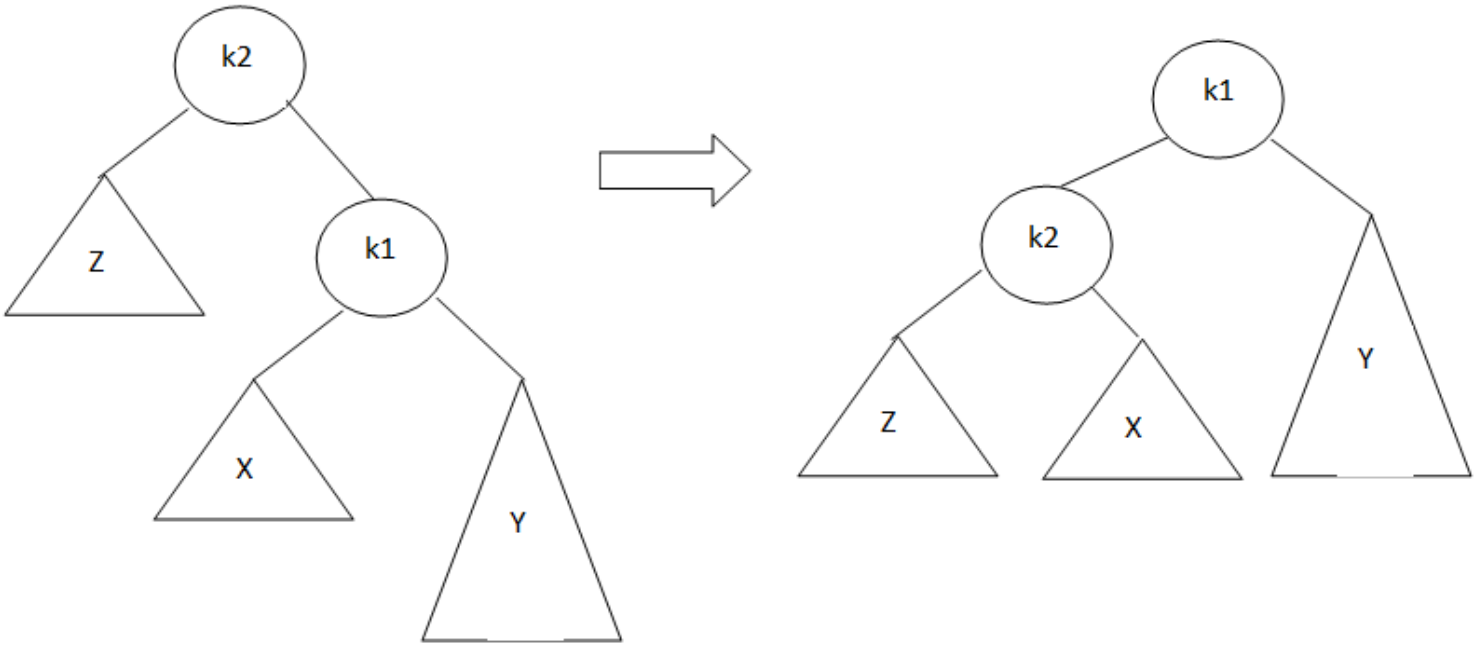


AVL Trees - rotations

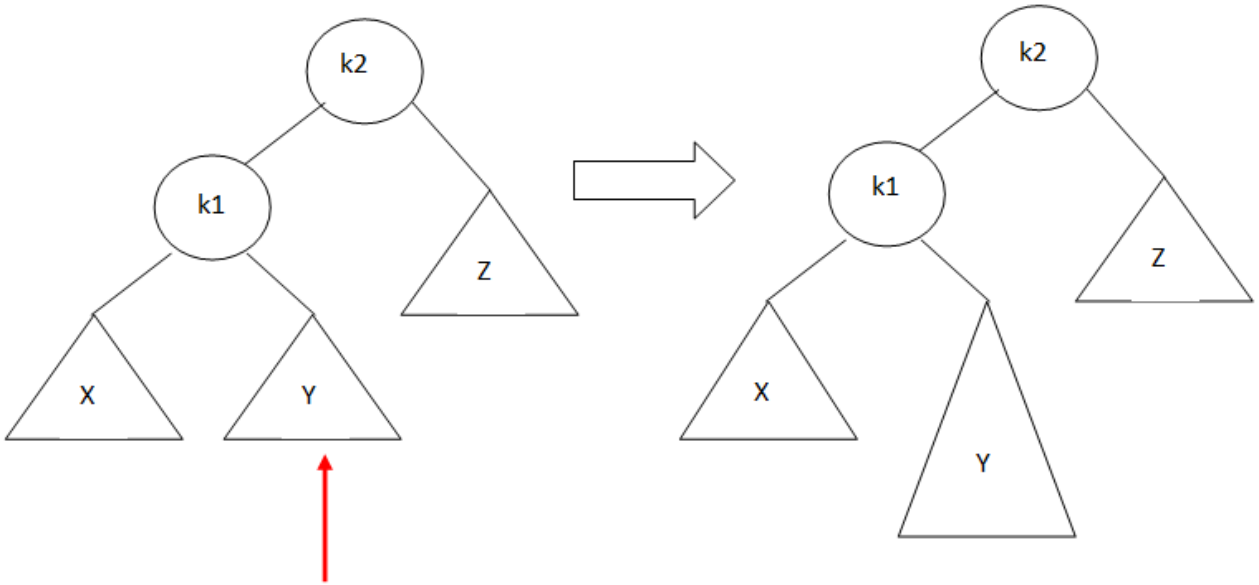


- Solution: **single rotation to left**

AVL Trees - rotation - Single Rotation to Left

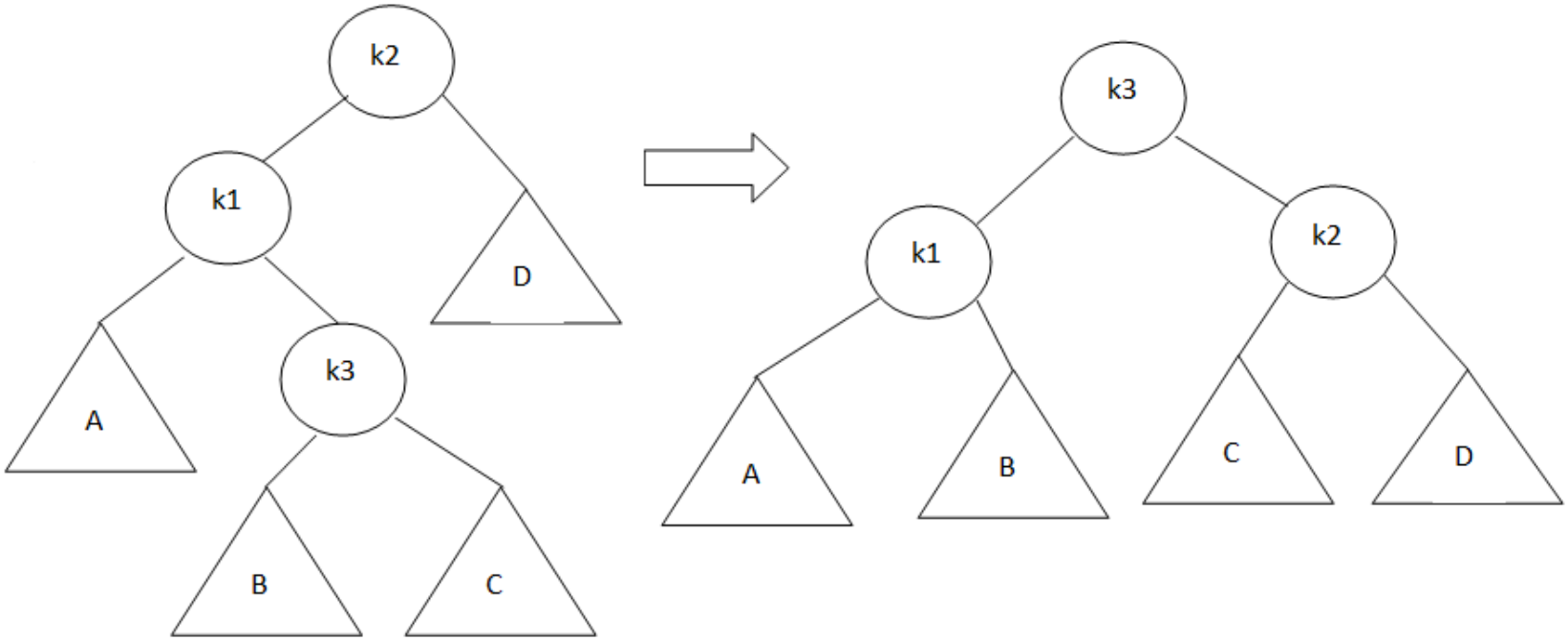


AVL Trees - rotations

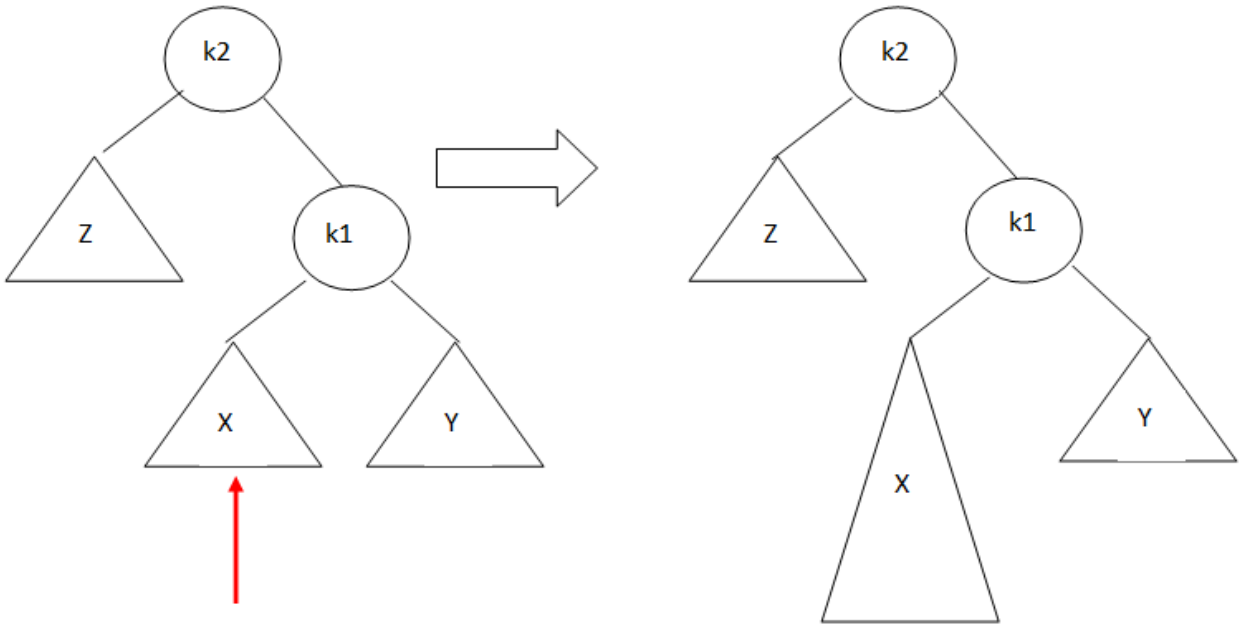


- Solution: **Double rotation to right**

AVL Trees - rotation - Double Rotation to Right

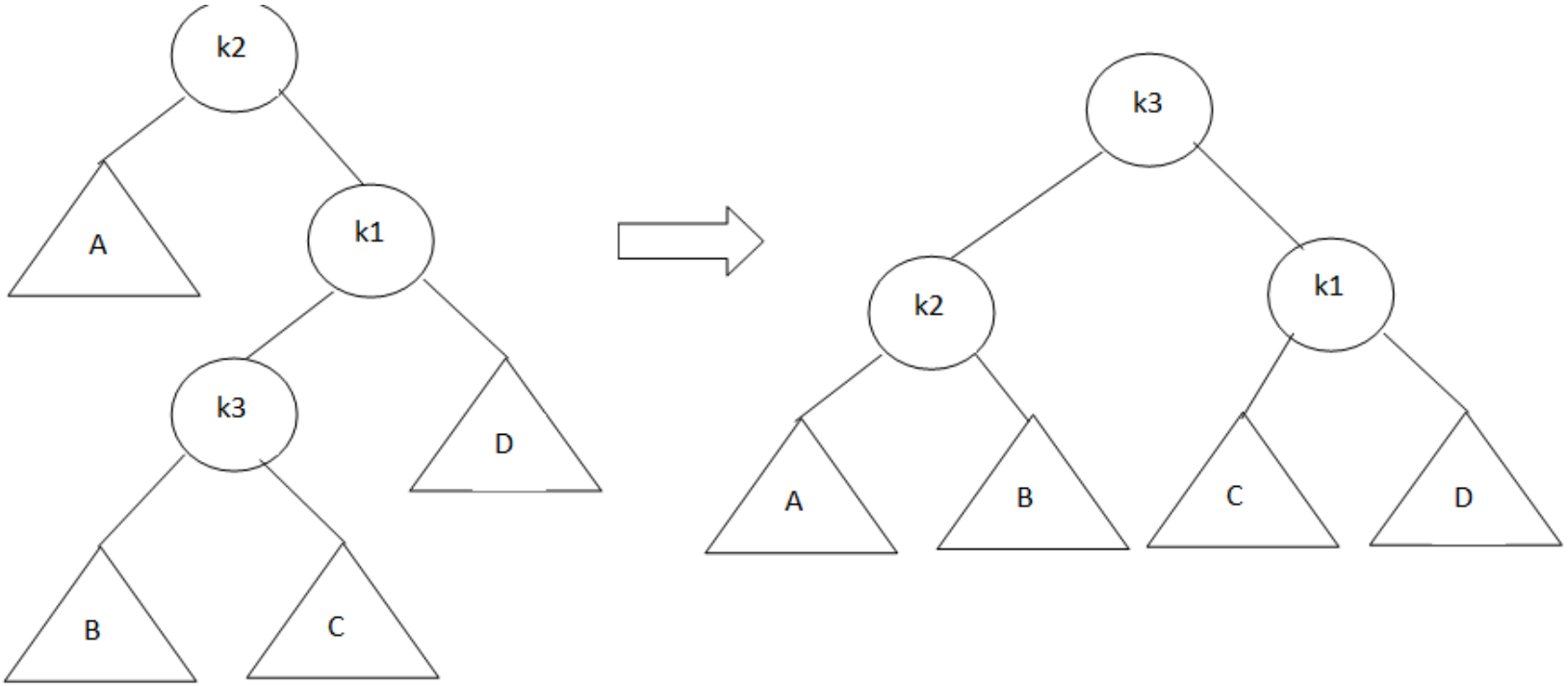


AVL Trees - rotations



- Solution: **Double rotation to left**

AVL Trees - rotation - Double Rotation to Left



AVL rotations example I

- Start with an empty AVL tree
- Insert 2

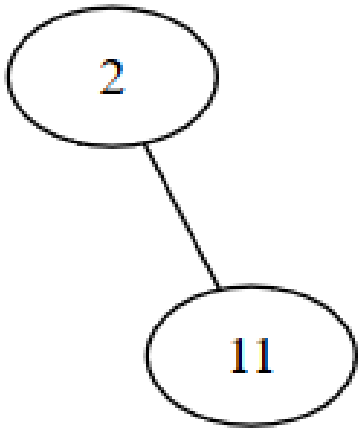
AVL rotations example II

- Do we need a rotation?
- If yes, on which node and what type of rotation?

AVL rotations example III

- No rotation is needed
- Insert 11

AVL rotations example IV

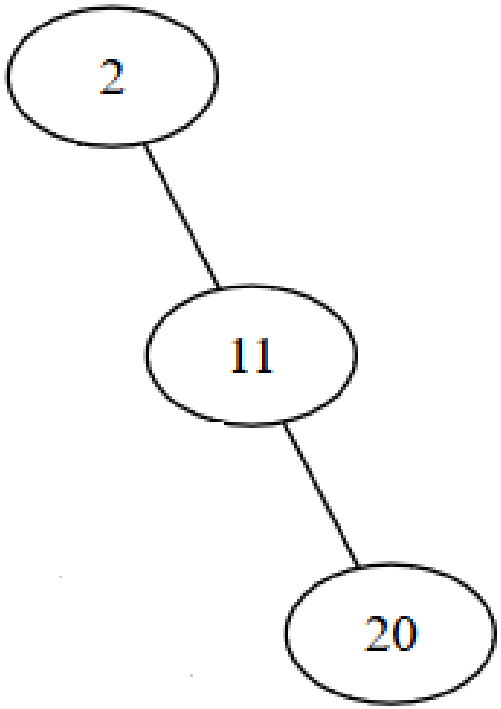


- Do we need a rotation?
- If yes, on which node and what type of rotation?

AVL rotations example V

- No rotation is needed
- Insert 20

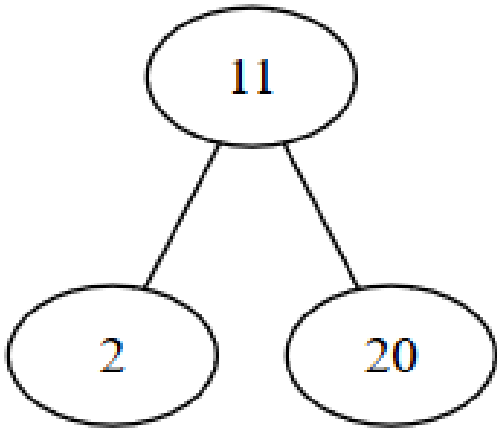
AVL rotations example VI



- Do we need a rotation?
- If yes, on which node and what type of rotation?

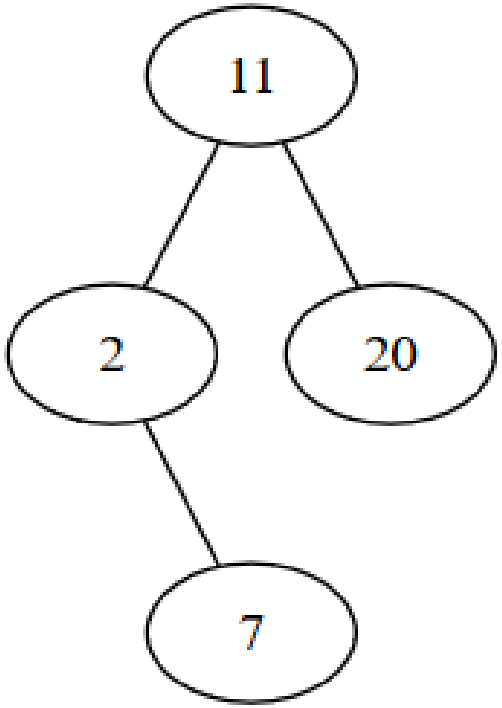
AVL rotations example VII

- Yes, we need a single left rotation on node 2
- After the rotation:



- Insert 7

AVL rotations example VIII

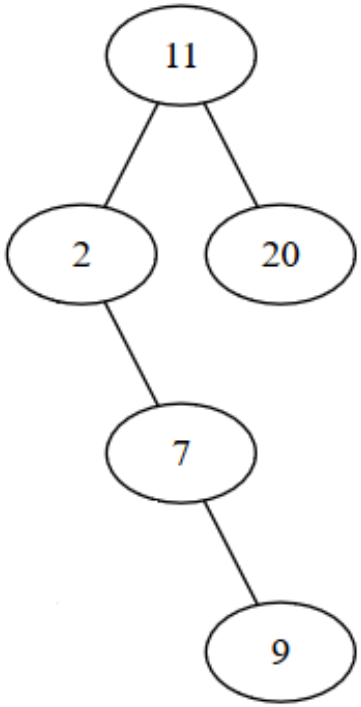


- Do we need a rotation?
- If yes, on which node and what type of rotation?

AVL rotations example IX

- No rotation is needed
- Insert 9

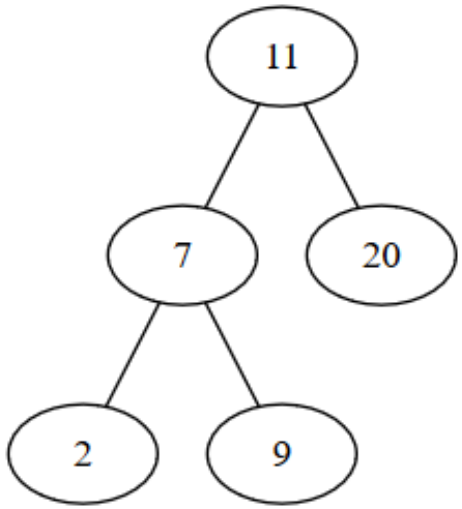
AVL rotations example X



- Do we need a rotation?
- If yes, on which node and what type of rotation?

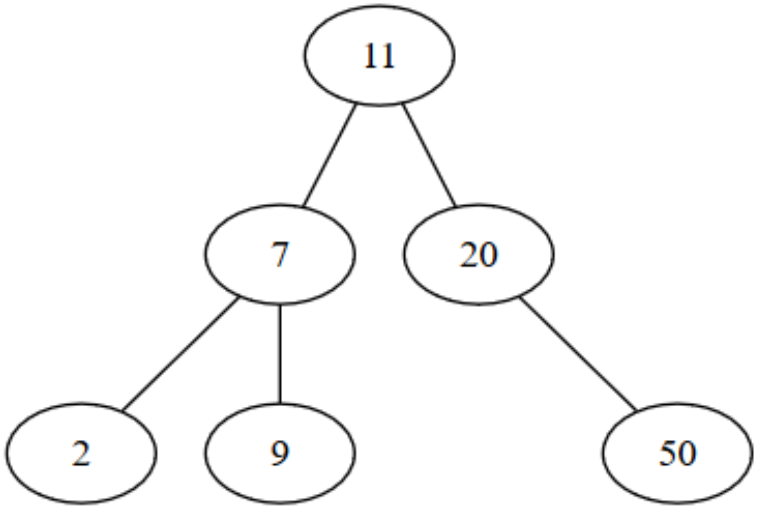
AVL rotations example XI

- Yes, we need a single left rotation on node 2
- After the rotation:



- Insert 50

AVL rotations example XII

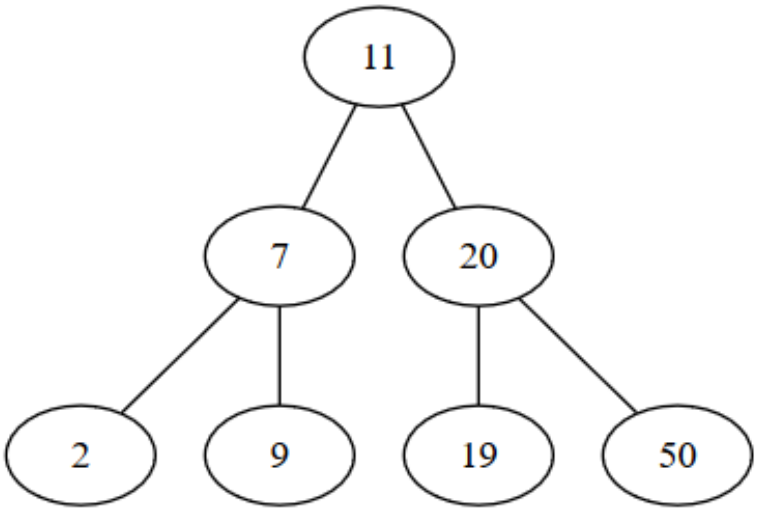


- Do we need a rotation?
- If yes, on which node and what type of rotation?

AVL rotations example XIII

- No rotation is needed
- Insert 19

AVL rotations example XIV

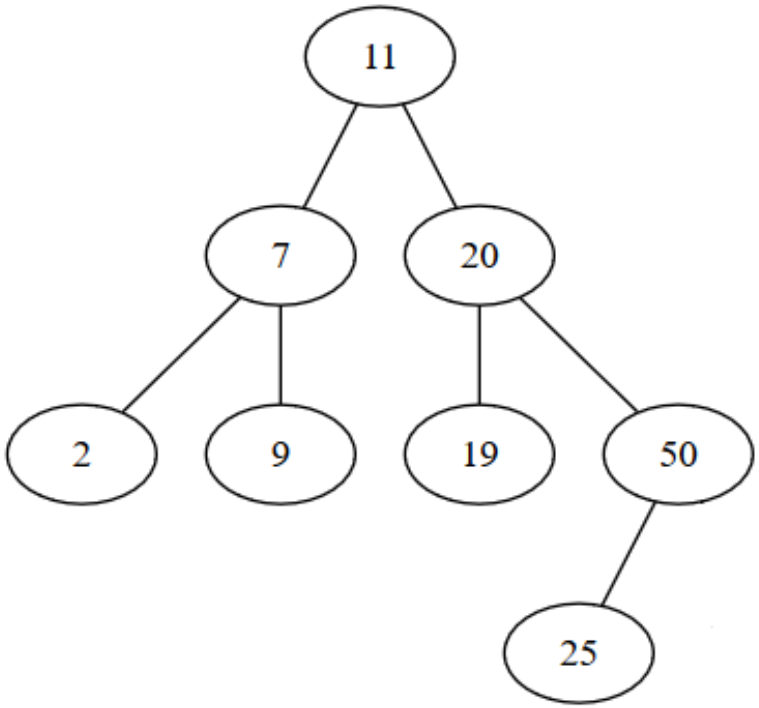


- Do we need a rotation?
- If yes, on which node and what type of rotation?

AVL rotations example XV

- No rotation is needed
- Insert 25

AVL rotations example XVI

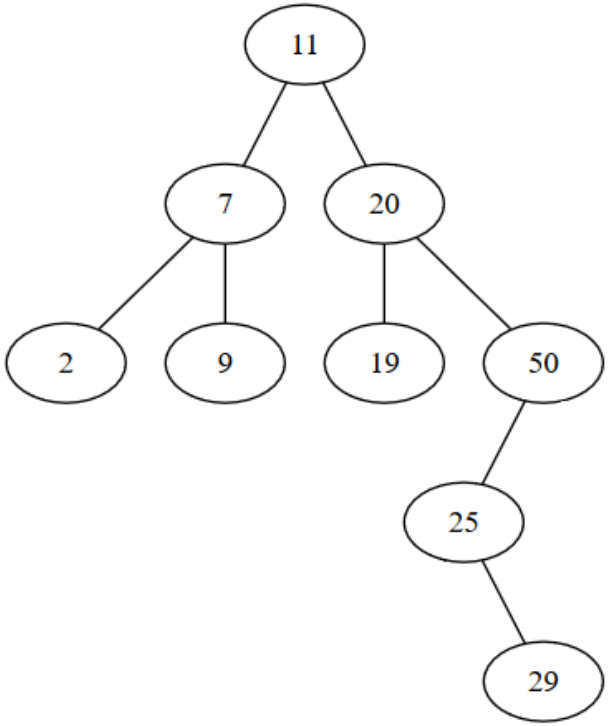


- Do we need a rotation?
- If yes, on which node and what type of rotation?

AVL rotations example XVII

- No rotation is needed
- Insert 29

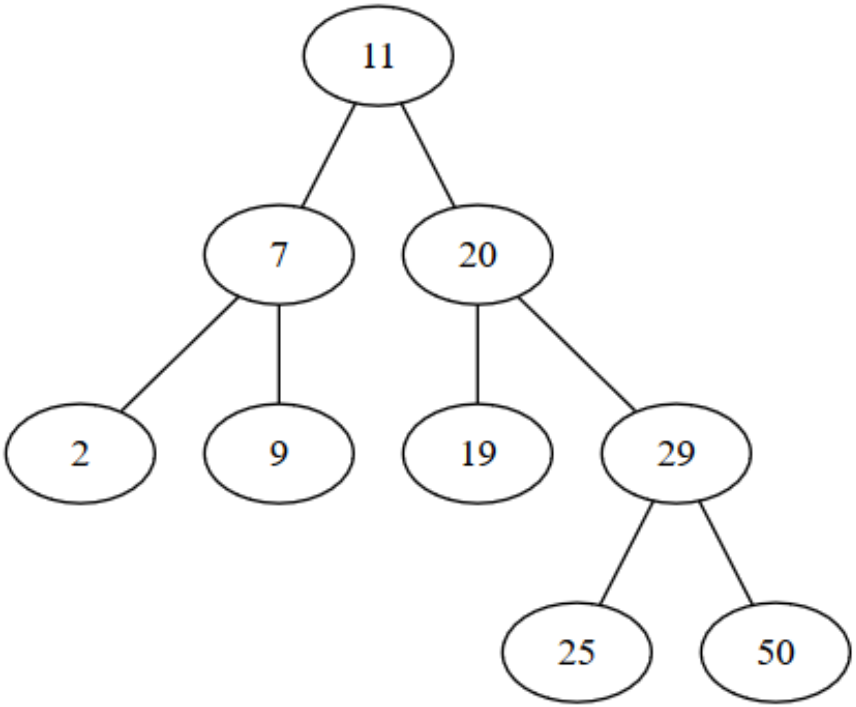
AVL rotations example XVIII



- Do we need a rotation?
- If yes, on which node and what type of rotation?

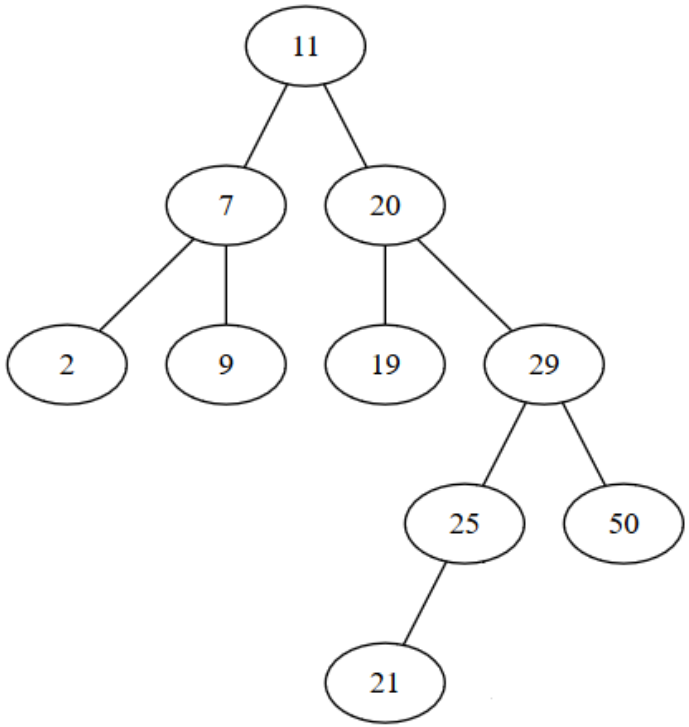
AVL rotations example XIX

- Yes, we need a double right rotation on node 50
- After the rotation



- Insert 21

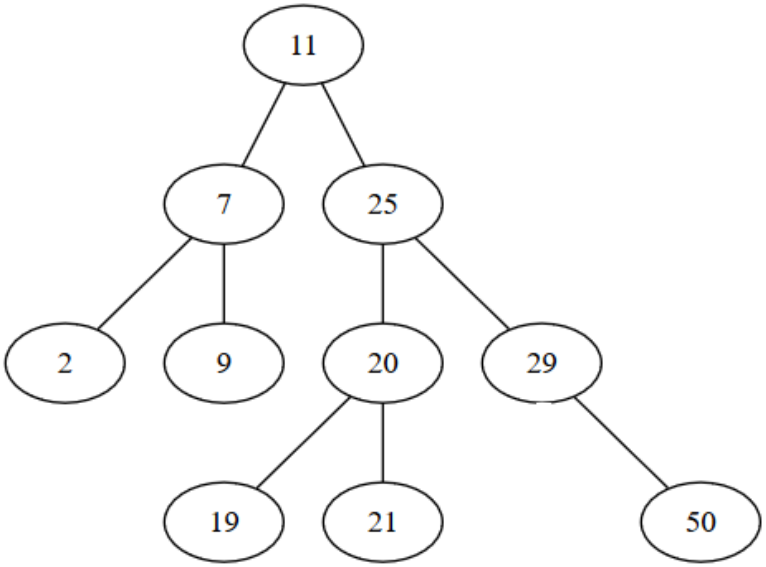
AVL rotations example XX



- Do we need a rotation?
- If yes, on which node and what type of rotation?

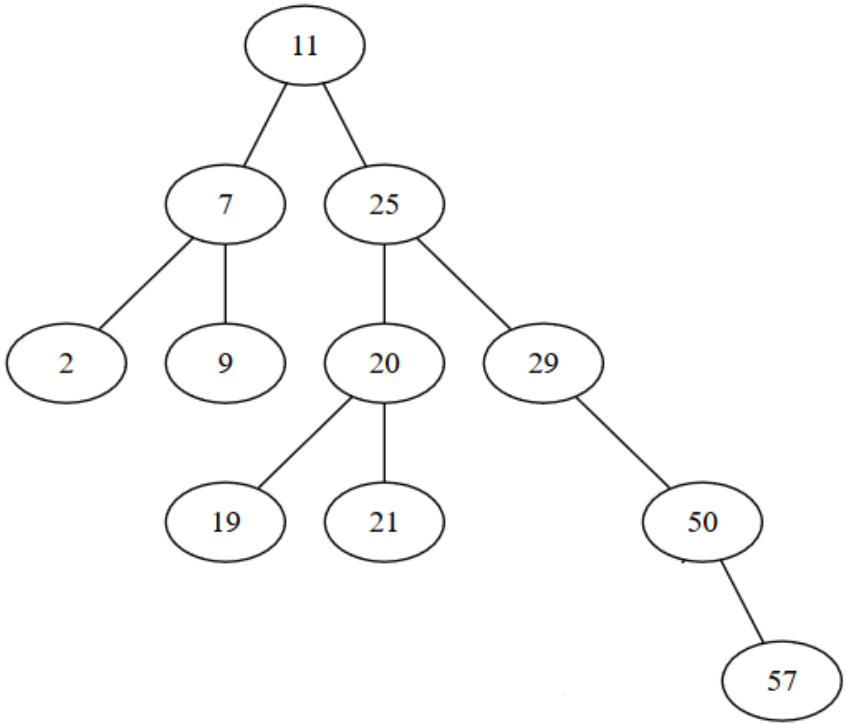
AVL rotations example XXI

- Yes, we need a double left rotation on node 20
- After the rotation



- Insert 57

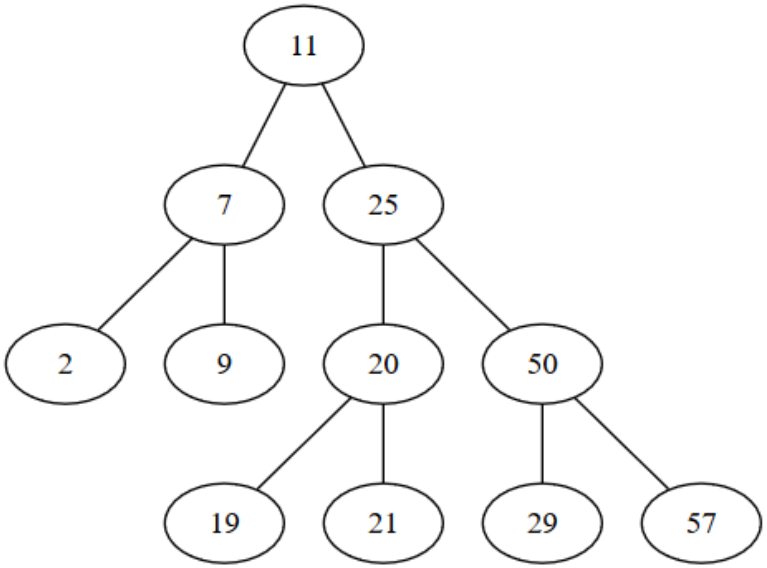
AVL rotations example XXII



- Do we need a rotation?
- If yes, on which node and what type of rotation?

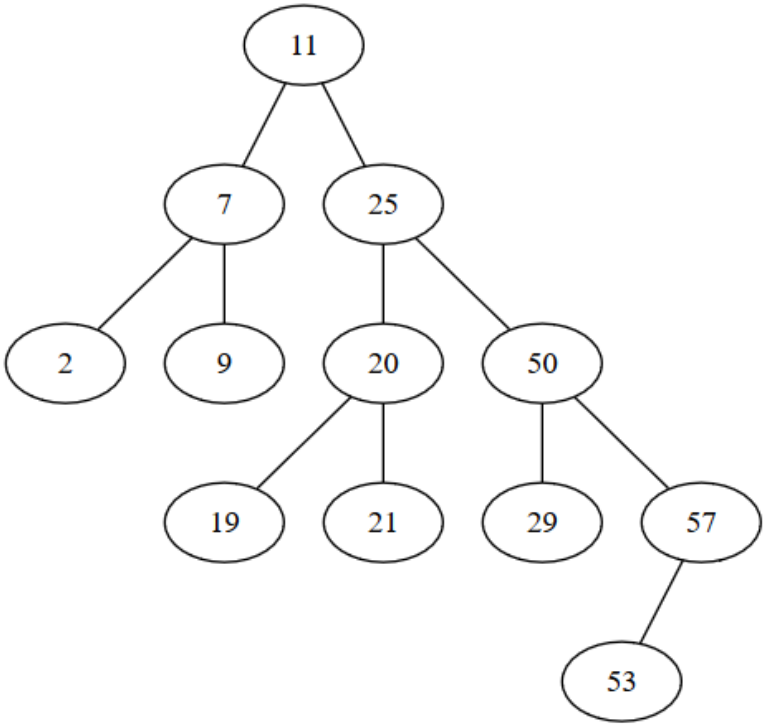
AVL rotations example XXIII

- Yes, we need a single left rotation on node 50
- After the rotation



- Insert 53

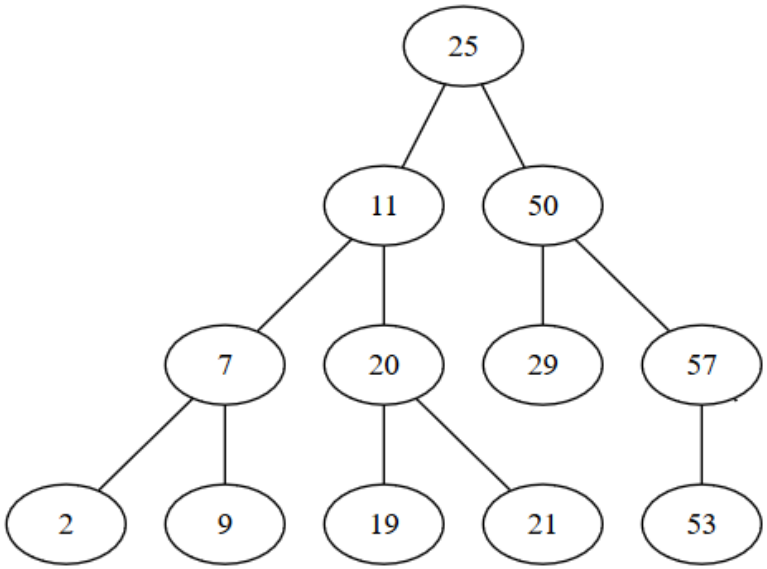
AVL rotations example XXIV



- Do we need a rotation?
- If yes, on which node and what type of rotation?

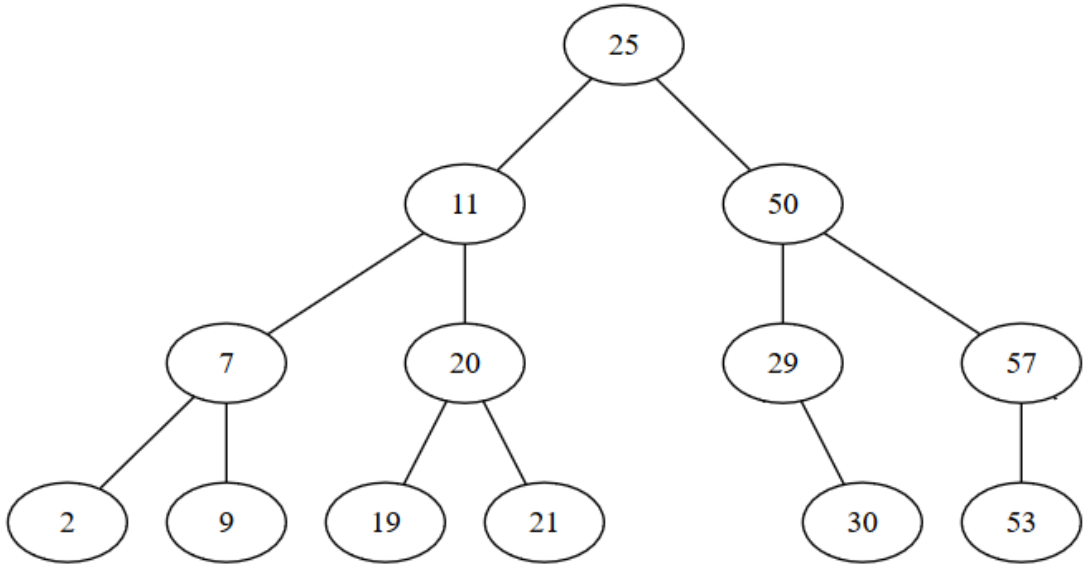
AVL rotations example XXV

- Yes, we need a single left rotation on node 11
- After the rotation



- Insert 30

AVL rotations example XXVI

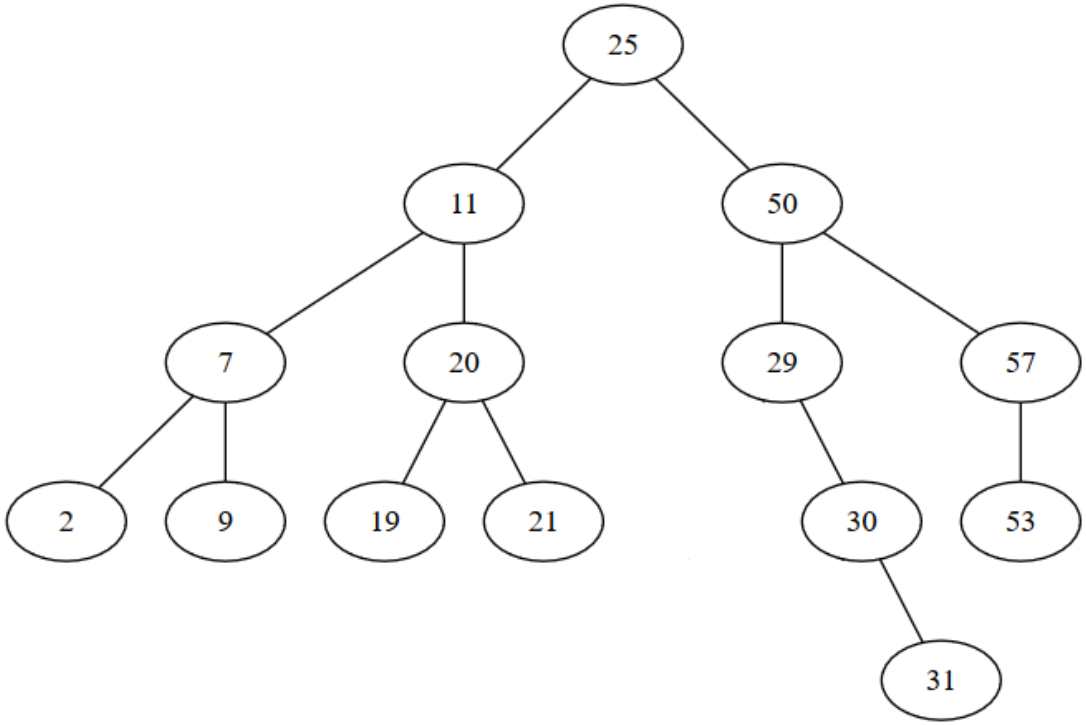


- Do we need a rotation?
- If yes, on which node and what type of rotation?

AVL rotations example XXVII

- No rotation is needed
- Insert 31

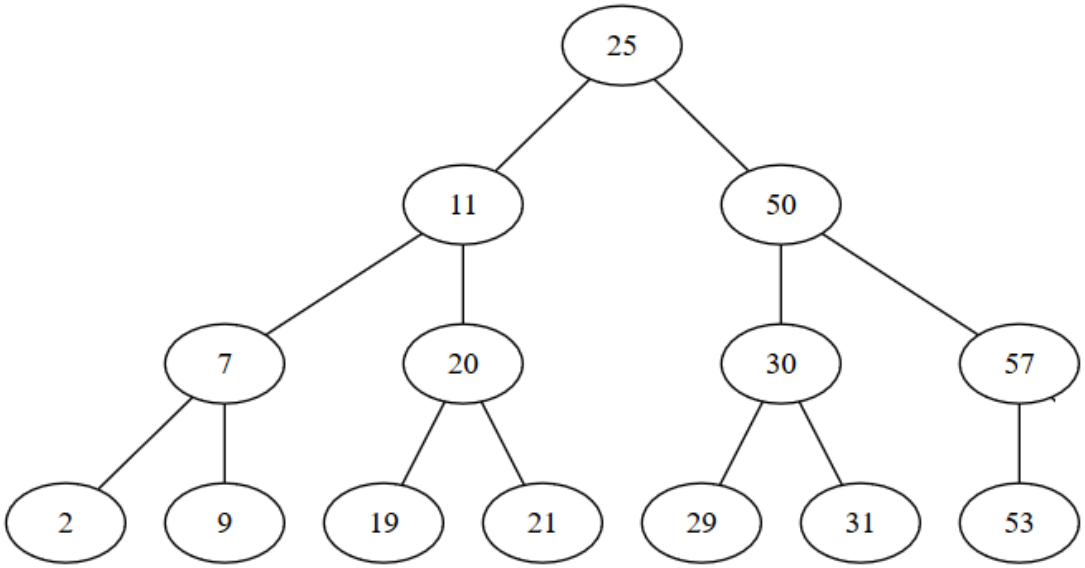
AVL rotations example XXVIII



- Do we need a rotation?
- If yes, on which node and what type of rotation?

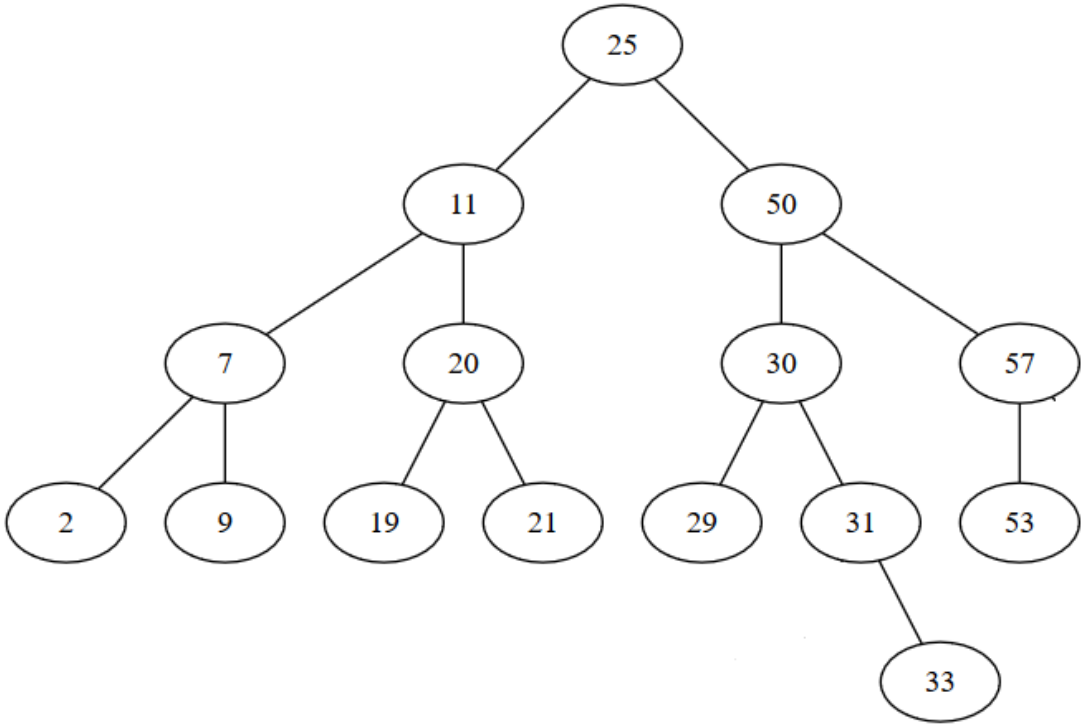
AVL rotations example XXIX

- Yes, we need a single left rotation on node 29
- After the rotation



- Insert 33

AVL rotations example XXX

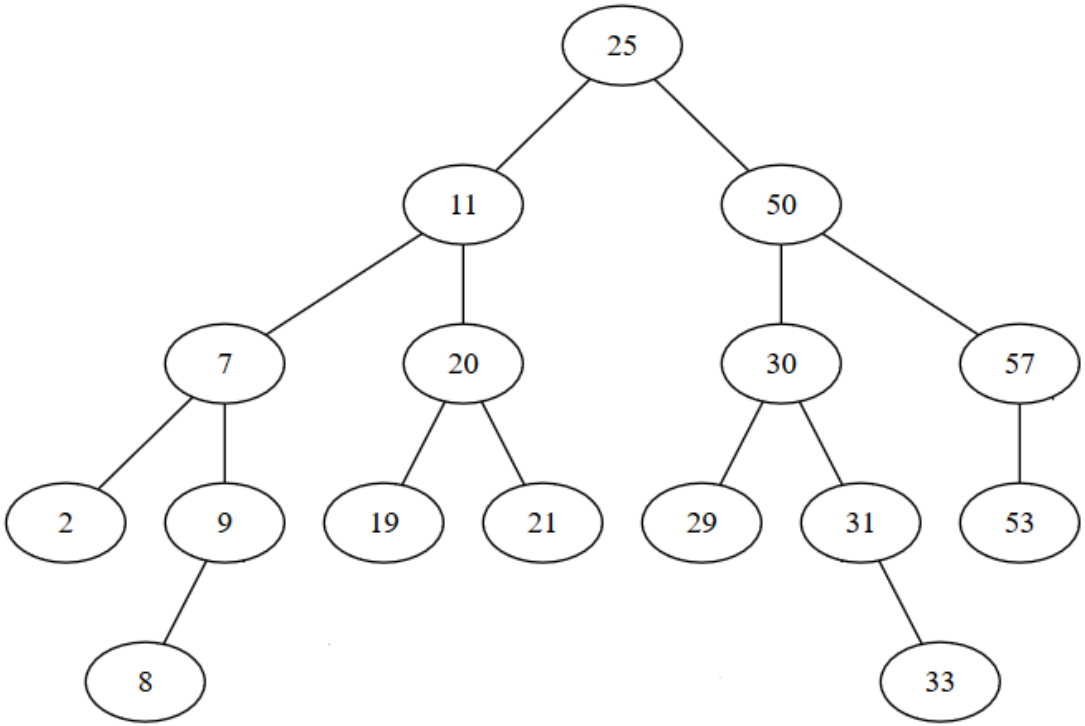


- Do we need a rotation?
- If yes, on which node and what type of rotation?

AVL rotations example XXXI

- No rotation is needed
- Insert 8

AVL rotations example XXXII

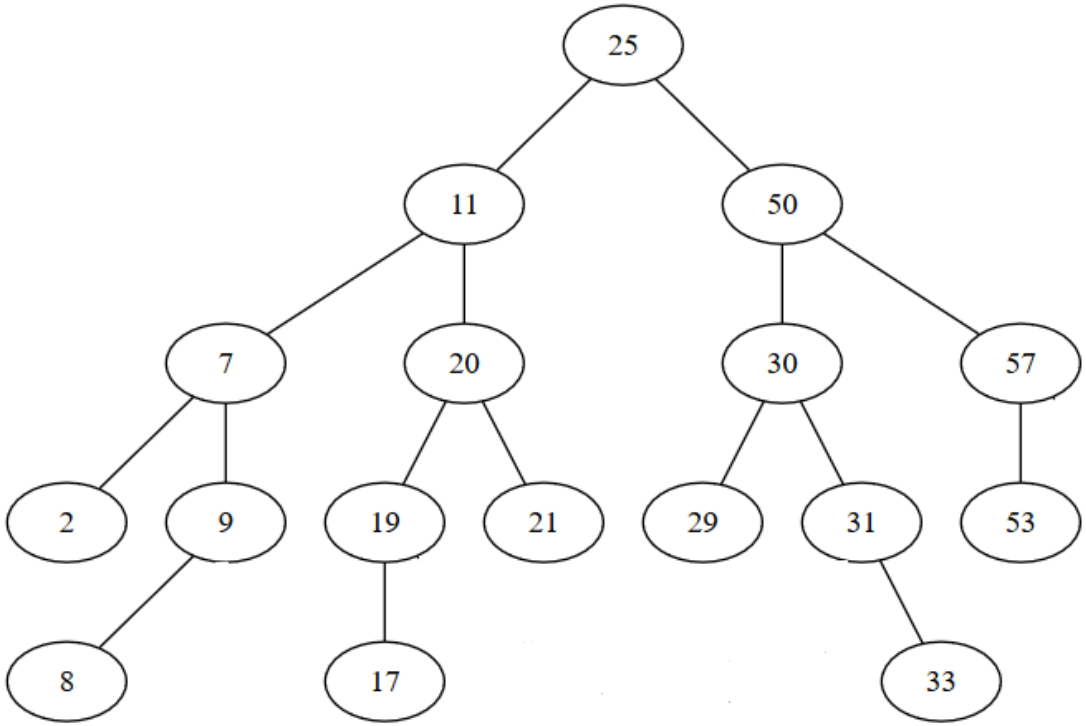


- Do we need a rotation?
- If yes, on which node and what type of rotation?

AVL rotations example XXXIII

- No rotation is needed
- Insert 17

AVL rotations example XXXIV

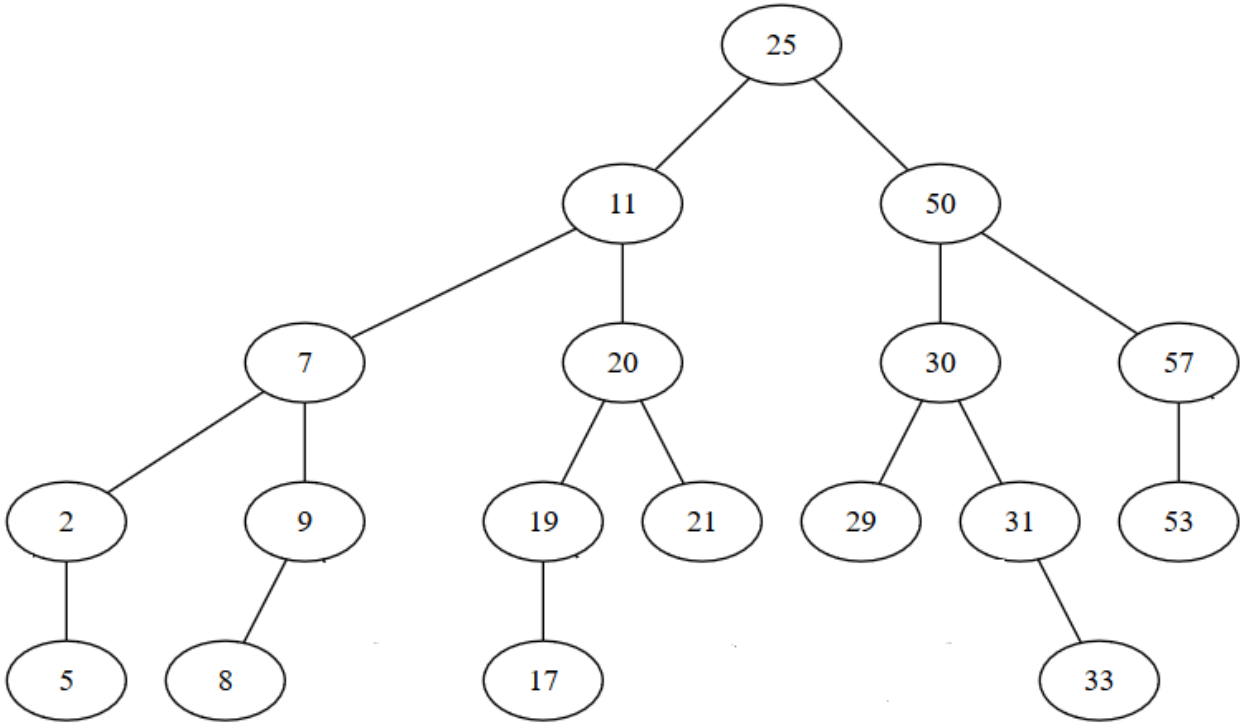


- Do we need a rotation?
- If yes, on which node and what type of rotation?

AVL rotations example XXXV

- No rotation is needed
- Insert 5

AVL rotations example XXXVI

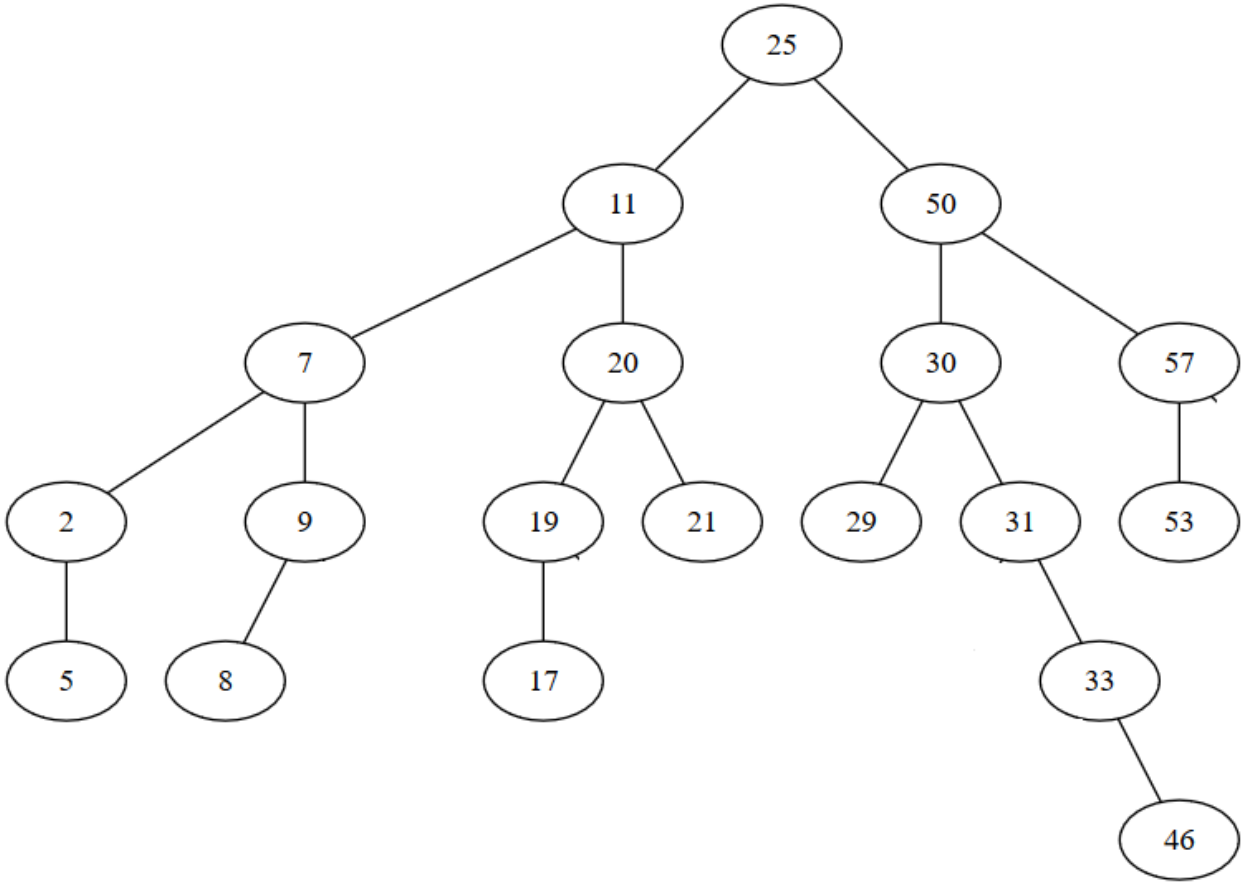


- Do we need a rotation?
- If yes, on which node and what type of rotation?

AVL rotations example XXXVII

- No rotation is needed
- Insert 46

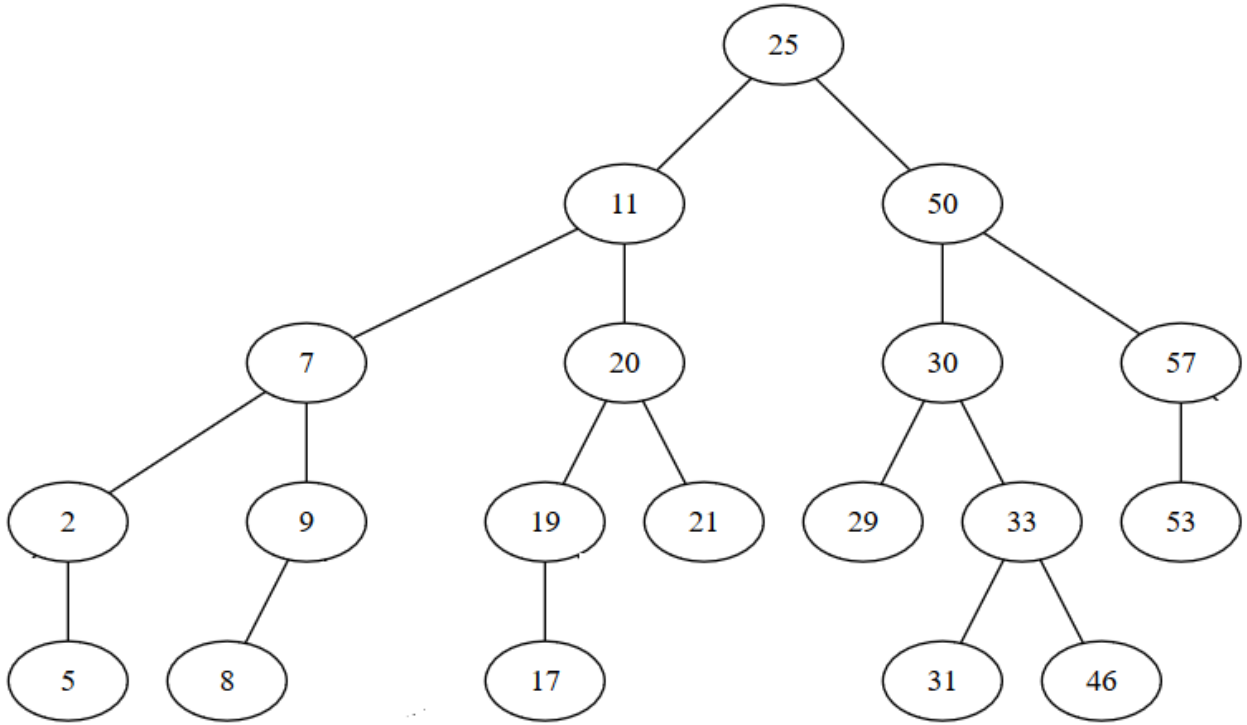
AVL rotations example XXXVIII



- Do we need a rotation?
- If yes, on which node and what type of rotation?

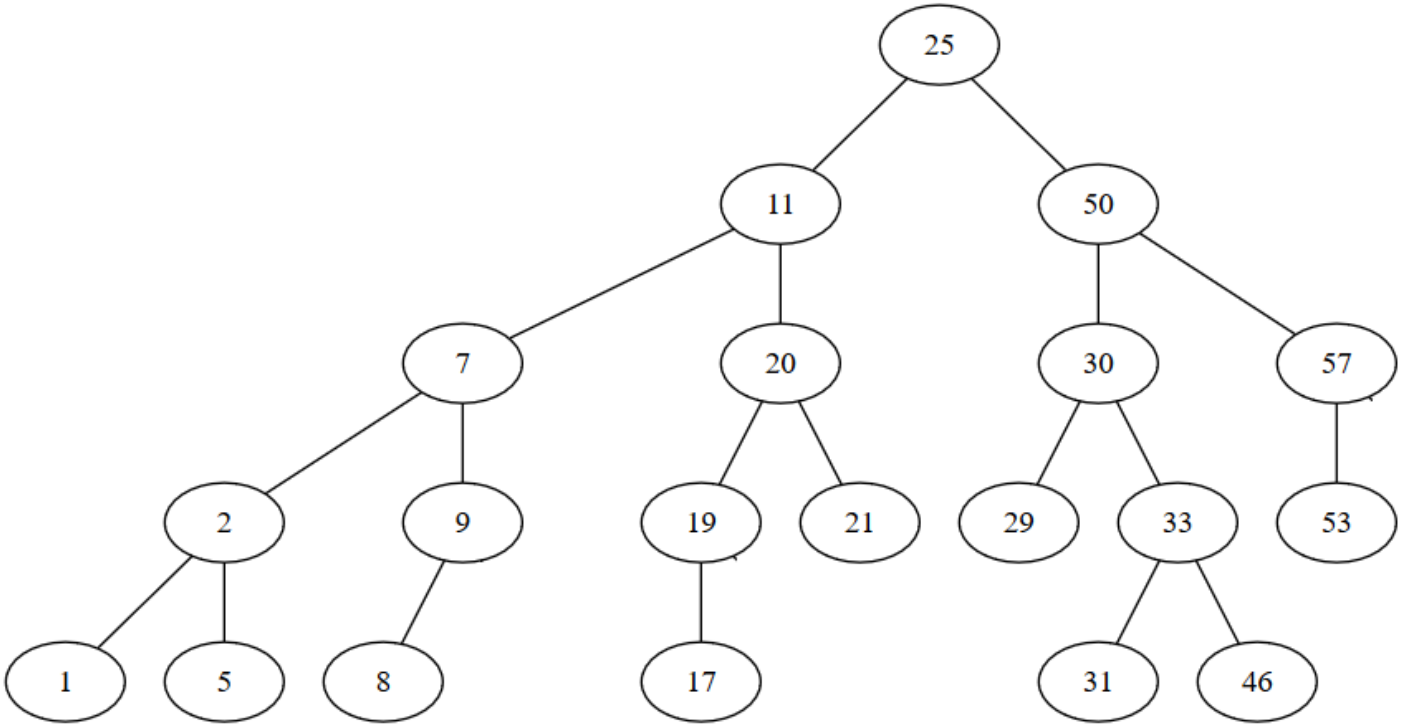
AVL rotations example XXXIX

- Yes, we need a single left rotation on node 31
- After the rotation



- Insert 1

AVL rotations example XL

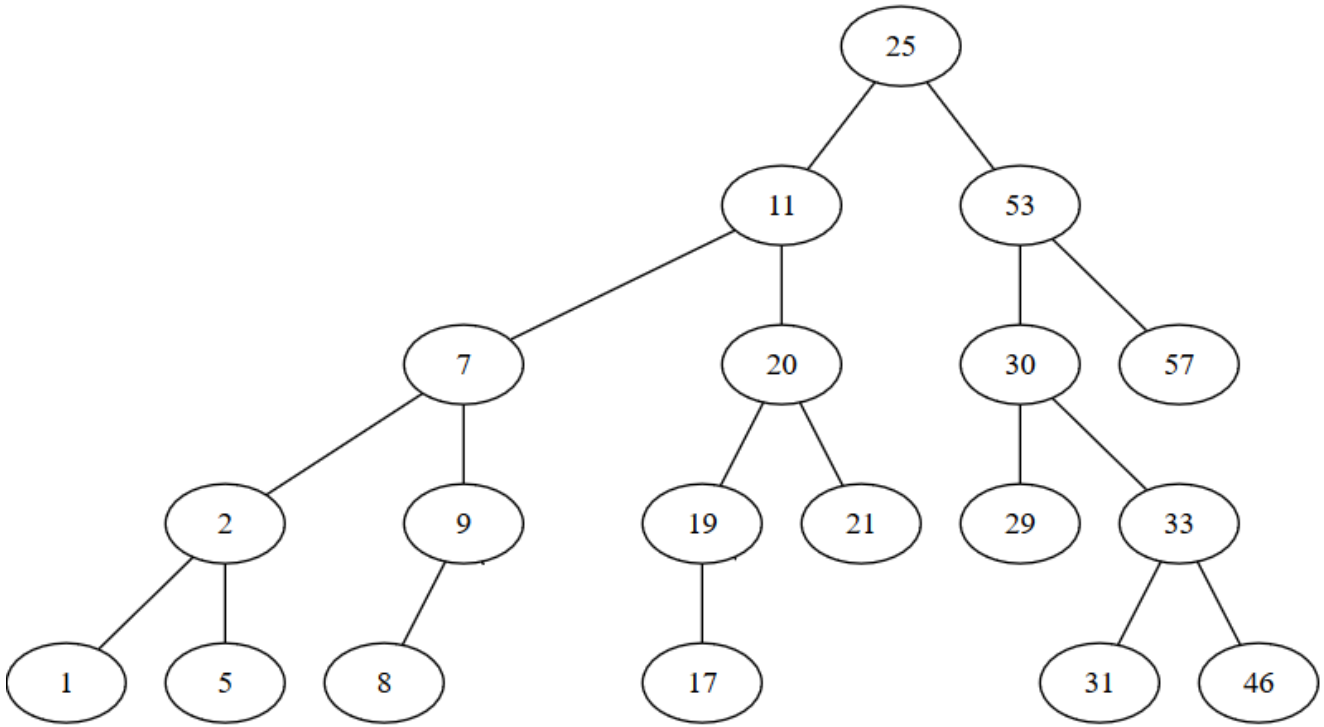


- Do we need a rotation?
- If yes, on which node and what type of rotation?

AVL rotations example XLI

- No rotation is needed
- Remove 50

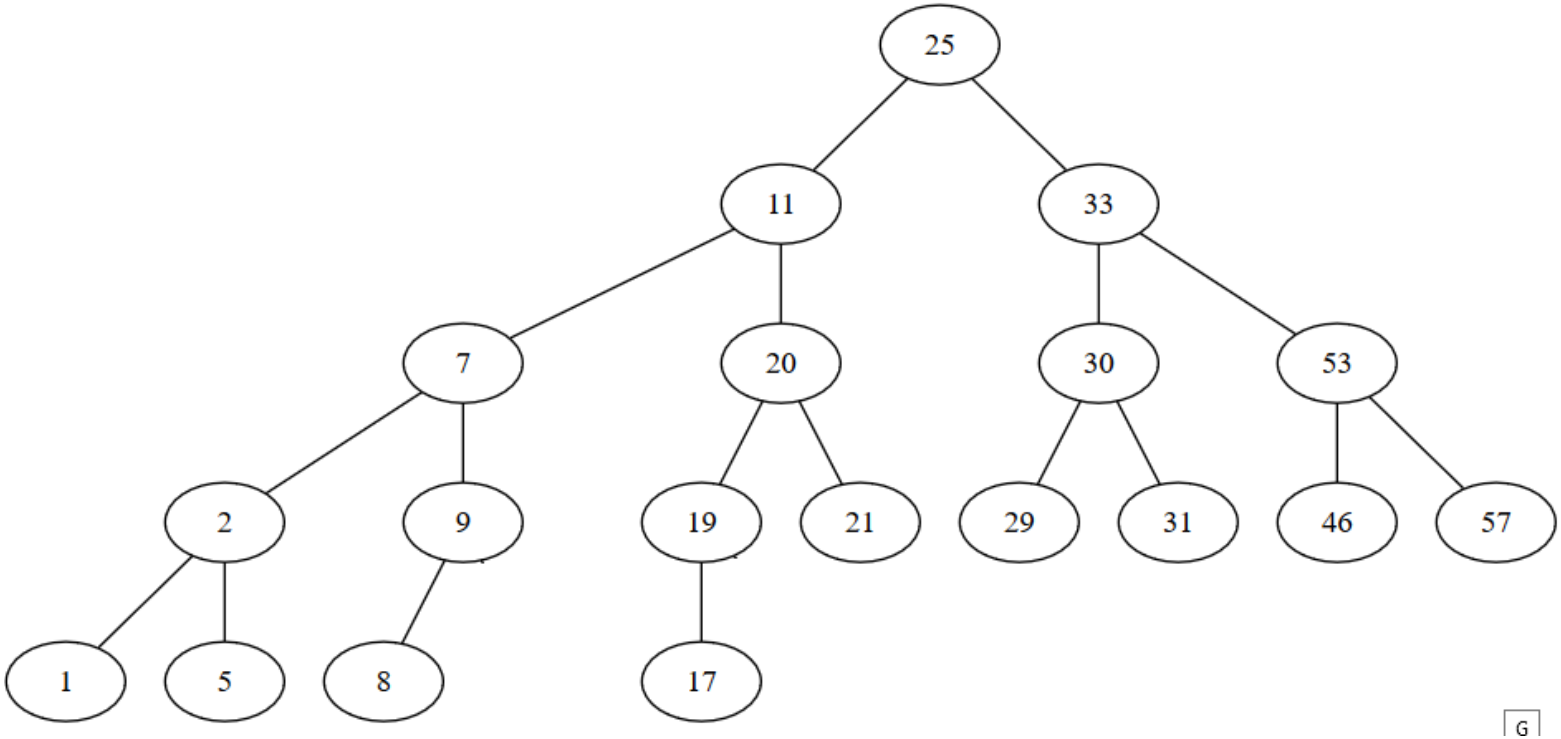
AVL rotations example XLII



- Do we need a rotation?
- If yes, on which node and what type of rotation?

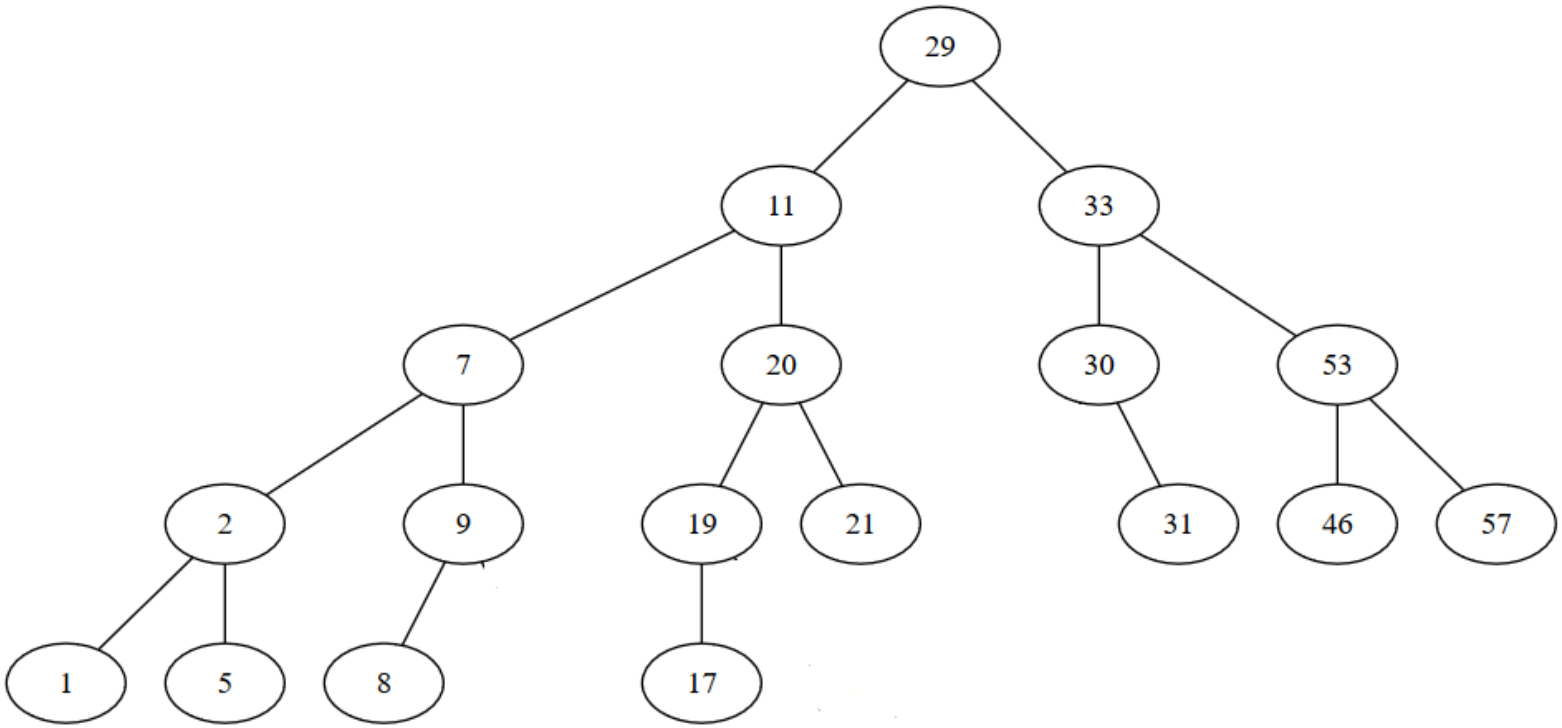
AVL rotations example XLIII

- Yes we need double right rotation on node 53
- After the rotation



- Remove 25

AVL rotations example XLIV

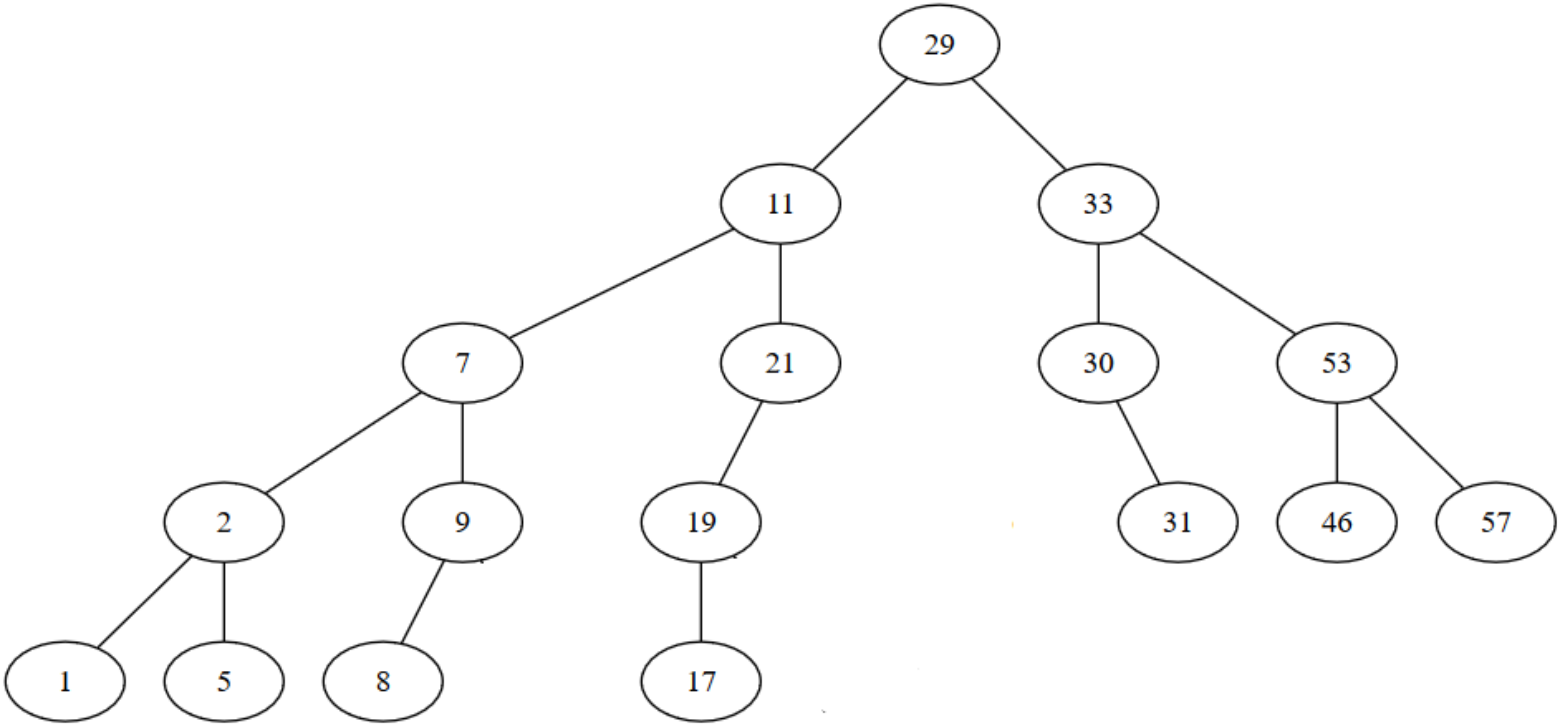


- Do we need a rotation?
- If yes, on which node and what type of rotation?

AVL rotations example XLV

- No rotation is needed
- Remove 20

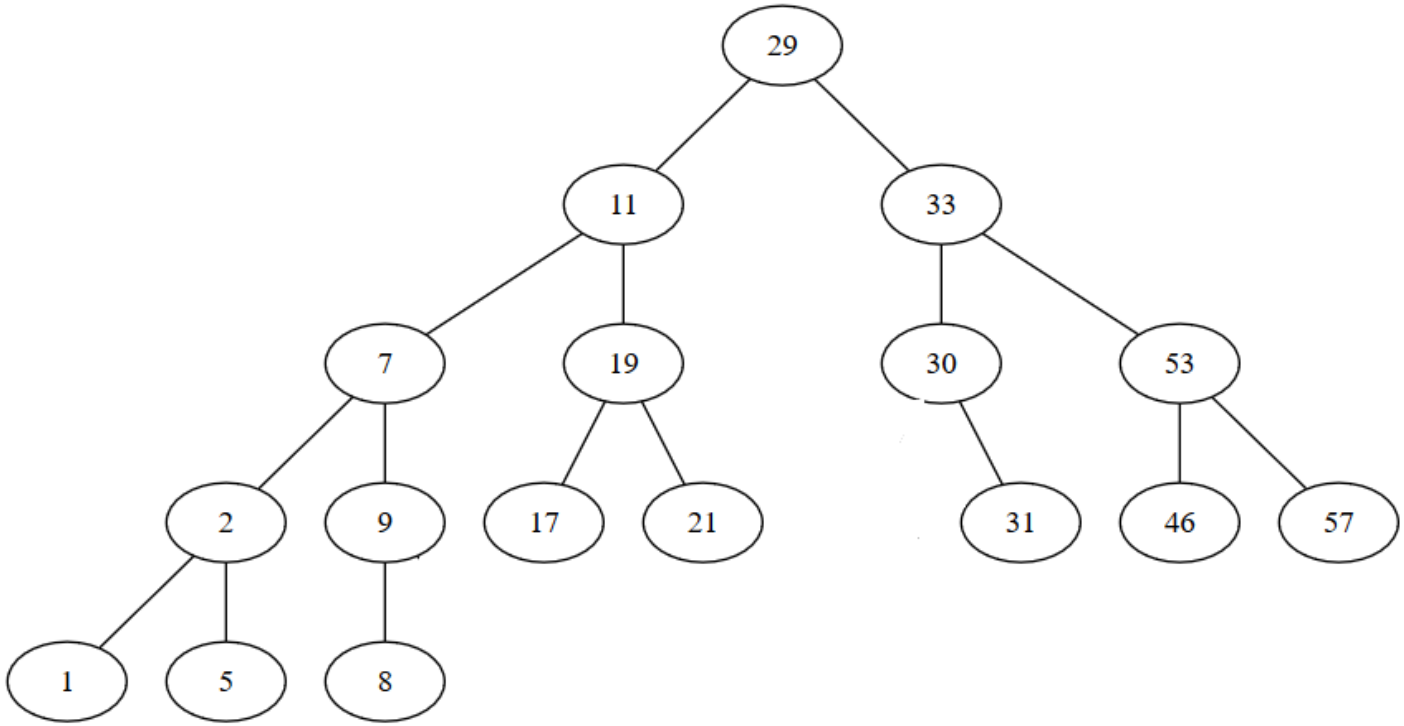
AVL rotations example XLVI



- Do we need a rotation?
- If yes, on which node and what type of rotation?

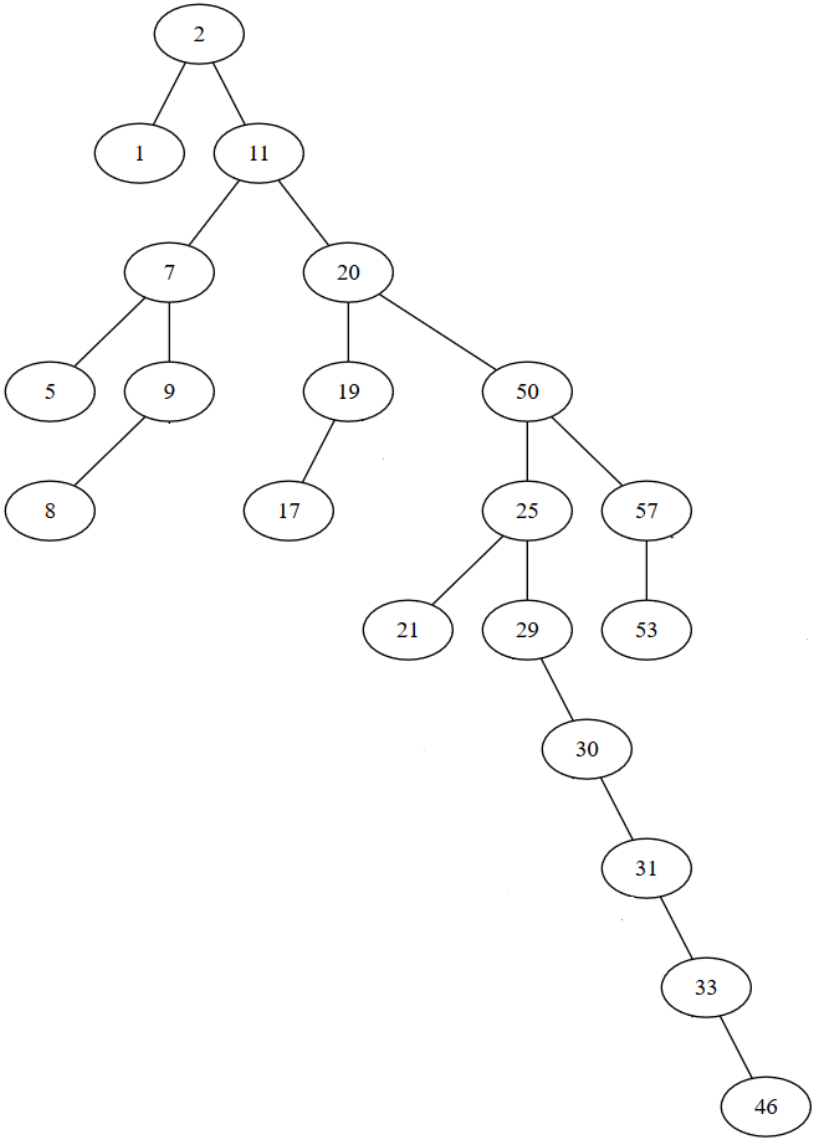
AVL rotations example XLVII

- Yes, we need a single right rotation on node 21
- After the rotation



Comparison to BST

- If, instead of using an AVL tree, we used a binary search tree, after the insertions the tree would have been:


2
1
11
7
20
5
9
19
50
8
17
25
57
21
29
53
30
31
33
46

AVL Trees - representation

- What structures do we need for an AVL Tree?

AVL Trees - representation

- What structures do we need for an AVL Tree?

AVLNode:
info: TComp *//information from the node*
left: ↑ AVLNode *//address of left child*
right: ↑ AVLNode *//address of right child*
h: Integer *//height of the node*

AVLTree:
root: ↑ AVLNode *//root of the tree*

AVL Tree - implementation

- We will implement the *insert* operation for the AVL Tree.
- We need to implement some operations to make the implementation of *insert* simpler:
  - A subalgorithm that (re)computes the height of a node
  - A subalgorithm that computes the balance factor of a node
  - Four subalgorithms for the four rotation types (we will implement only one)
- And we will assume that we have a function, *createNode* that creates and returns a node containing a given information (left and right are NIL, height is 0).

AVL Tree - height of a node

subalgorithm recomputeHeight(node) **is:**
//pre: node is an ↑ AVLNode. All descendants of node have their height (h) set
//to the correct value
//post: if node ≠ NIL, h of node is set

AVL Tree - height of a node

```
subalgorithm recomputeHeight(node) is:
//pre: node is an ↑ AVLNode. All descendants of node have their height (h) set
//to the correct value
//post: if node ≠ NIL, h of node is set
    if node ≠ NIL then
        if [node].left = NIL and [node].right = NIL then
            [node].h ← 0
        else if [node].left = NIL then
            [node].h ← [[node].right].h + 1
        else if [node].right = NIL then
            [node].h ← [[node].left].h + 1
        else
            [node].h ← max ([[node].left].h, [[node].right].h) + 1
        end-if
    end-if
end-subalgorithm
```

- Complexity: $\Theta(1)$

AVL Tree - balance factor of a node

function balanceFactor(node) **is:**
//pre: node is an ↑ AVLNode. All descendants of node have their height (h) set
//to the correct value
//post: returns the balance factor of the node

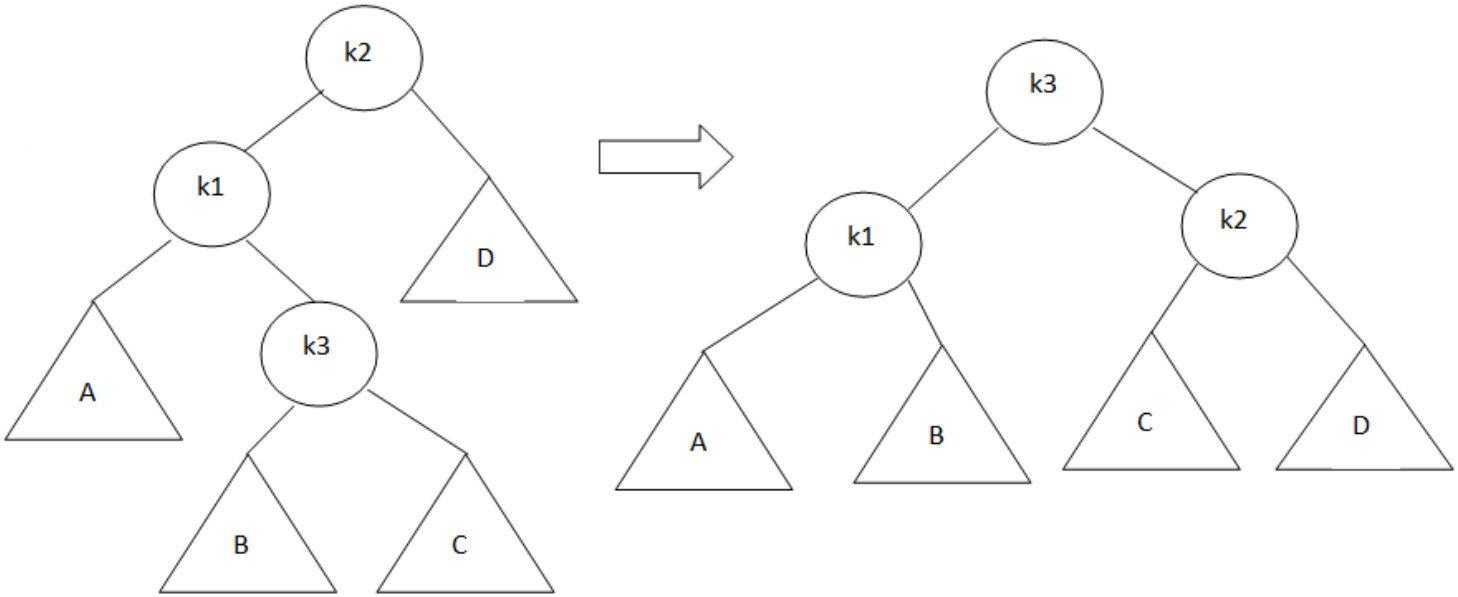
AVL Tree - balance factor of a node

```
function balanceFactor(node) is:
//pre: node is an ↑ AVLNode. All descendants of node have their height (h) set
//to the correct value
//post: returns the balance factor of the node
    if [node].left = NIL and [node].right = NIL then
        balanceFactor ← 0
    else if [node].left = NIL then
        balanceFactor ← -1 - [[node].right].h //height of empty tree is -1
    else if [node].right = NIL then
        balanceFactor ← [[node].left].h + 1
    else
        balanceFactor ← [[node].left].h - [[node].right].h
    end-if
end-subalgorithm
```

- Complexity: $\Theta(1)$

AVL Tree - rotations

- Out of the four rotations, we will only implement one, double right rotation (DRR).
- The other three rotations can be implemented similarly (RLR, SRR, SLR).



AVL Tree - DRR

function DRR(node) **is:** *//pre: node is an ↑ AVLNode on which we perform the double right rotation*
//post: DRR returns the new root after the rotation
    k2 ← node
    k1 ← [node].left
    k3 ← [k1].right
    k3left ← [k3].left
    k3right ← [k3].right

AVL Tree - DRR

function DRR(node) **is:** *//pre: node is an ↑ AVLNode on which we perform the double right rotation*
//post: DRR returns the new root after the rotation
    k2 ← node
    k1 ← [node].left
    k3 ← [k1].right
    k3left ← [k3].left
    k3right ← [k3].right
    //reset the links
    newRoot ← k3
    [newRoot].left ← k1
    [newRoot].right ← k2
    [k1].right ← k3left
    [k2].left ← k3right
//continued on the next slide

AVL Tree - DRR

//recompute the heights of the modified nodes
recomputeHeight(k1)
recomputeHeight(k2)
recomputeHeight(newRoot)
DRR $\leftarrow$ newRoot
end-function

- Complexity: $\Theta(1)$

AVL Tree - insert

function insertRec(node, elem) **is**
//pre: node is a ↑ AVLNode, elem is the value we insert in the (sub)tree that
//has node as root
//post: insertRec returns the new root of the (sub)tree after the insertion
  if node = NIL **then**
    insertRec ← createNode(elem)
  else ifelem ≤ [node].info **then**
     [node].left ← insertRec([node].left, elem)
  else
    [node].right ← insertRec([node].right, elem)
  end-if
//continued on the next slide...

AVL Tree - insert

recomputeHeight(node)
balance $\leftarrow$ getBalanceFactor(node)
if balance = -2 **then**

AVL Tree - insert

recomputeHeight(node)
balance ← getBalanceFactor(node)
if balance = -2 **then**
//right subtree has larger height, we will need a rotation to the LEFT
    rightBalance ← getBalanceFactor([node].right)
    if rightBalance < 0 **then**

AVL Tree - insert

```
recomputeHeight(node)
balance ← getBalanceFactor(node)
if balance = -2 then
//right subtree has larger height, we will need a rotation to the LEFT
    rightBalance ← getBalanceFactor([node].right)
    if rightBalance < 0 then
    //the right subtree of the right subtree has larger height, SRL
        node ← SRL(node)
    else
        node ← DRL(node)
    end-if
//continued on the next slide...
```

AVL Tree - insert

else if balance $= 2$ **then**
//left subtree has larger height, we will need a RIGHT rotation
leftBalance $\leftarrow$ getBalanceFactor([node].left)
if leftBalance > 0 **then**

AVL Tree - insert

```
    else if balance = 2 then
    //left subtree has larger height, we will need a RIGHT rotation
        leftBalance ← getBalanceFactor([node].left)
        if leftBalance > 0 then
        //the left subtree of the left subtree has larger height, SRR
            node ← SRR(node)
        else
            node ← DRR(node)
        end-if
    end-if
    insertRec ← node
end-function
```

AVL Tree - insert

- Complexity of the *insertRec* algorithm: $O(log_2 n)$
- Since *insertRec* receives as parameter a pointer to a node, we need a wrapper function to do the first call on the root

subalgorithm insert(tree, elem) **is**
//pre: tree is an AVL Tree, elem is the element to be inserted
//post: elem was inserted to tree
  tree.root $\leftarrow$ insertRec(tree.root, elem)
end-subalgorithm

- remove subalgorithm can be implemented similarly (start from the remove from BST and add the rotation part).

Project presentation

- Project presentations will be held on 12, 13 and 14 of June.
- Project presentation schedule is available online: http://www.cs.ubbcluj.ro/ marianzsu/DSA/Projects/Schedule.pdf - presentation hours for some groups might change (changed hours will be posted the latest tomorrow evening).
- Every student has to come to the presentation with his/her own group.
- Do not forget to bring the documentation on paper.
- Be prepared to make modifications to your project (small ones). Failure to perform the modifications will result in a failing grade for the project.

Project presentation

- If you fail your project, the will have to redo it for the retake session.
- **No matter what your grade for the project is, you can participate in the written exam in the regular session - if you have the required number of seminar attendances**.

Written exam

Group	Primary date	Secondary date
911	19.06	24.06
912	23.06	22.06
913	23.06	22.06
914	19.06	24.06
915	24.06	19.06
916	24.06	19.06
917	22.06	15.06

- We do not know the rooms and the starting hours yet, but they will be available at the faculty's webpage.

Written exam

- Every student has to participate in the exam on the primary date.
- In the secondary date you can only participate if you have a good reason for asking this, and if you announce me at least 24 hours in advance and have my OK.
- Exam will take 2.5 - 3 hours - results will be given as soon as we can.
- You will need a grade of at least 5 for the written exam to be able to pass this course.

Written exam I

- Subjects for the written exam will be from everything we have covered this semester.
- You will have different problems:
  - Problems where we need short answers (maybe a drawing) - ex. insert something into a given BST tree, remove a node from a binary heap, merge two binomial heaps
  - Implementation problems (only for those data structures for which we have discussed implementation):
    - Given a container pick the most suitable representation so that operation X will have a complexity of Y/minimum complexity and implement the operation
    - Implement a given operation for a given container on a given representation

Written exam II

- Probably - "Pick the right answer from a, b, c, d and explain" - type problems
- Every exam subject will contain a
  - a "think about it" problem
  - a problem dealing with (probably binary) trees
- Anything else

- We want to see how you can put in practice what you have learned, not that you memorized the slides.

Written exam III

- If you have questions, if you think you found mistakes on the slides, you can contact me by mail.
- If you would like to meet to discuss questions, unclear issues, contact me by mail to set up a meeting.