


Data Structures and Algorithms – Lab9

Iulia-Cristina Stanica



Exercise 1

- ▶ Calculate the maximumHeight in a tree – the number of edges from the longest path root–leaf.
 - ▶ Suggestion: the maximum height of a tree is the maximum between the heights of its children.
 - ▶ **Obs:** for all exercises, you should firstly draw your tree from the example to check if you get the right result.
- 

Exercise 2

- ▶ Using the function from the previous exercise, write a function which calculates, for the root node, the difference between the height of the left subtree and the height of the right subtree.

Exercise 3

- ▶ Write a function which gets the greatest value of a BST.

Suggestion:

- Traverse the tree from the root to the right until the right pointer is NULL
- The node which has the right node NULL is the maximum
- `BinarySearchTree<T> *p = this;` – reference to the root

What do we do if we want to obtain the minimum value?

Exercise 4

- Write a function that displays the nodes' values greater than k_1 and smaller than k_2 , where k_1 and k_2 are function parameters

RANGEQUERYSIMPLE(T, k_1, k_2):

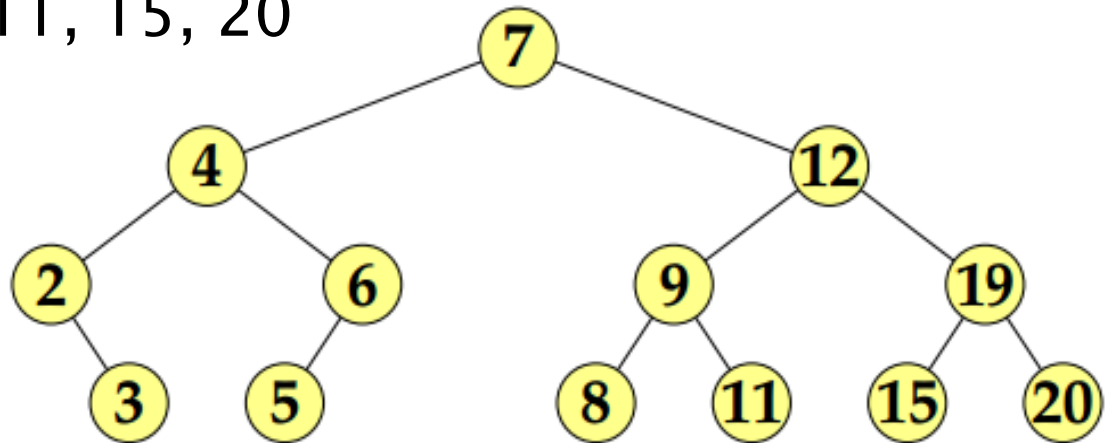
- 1: If T is empty, return
- 2: **if** $\text{key}(T.\text{root}) < k_1$ **then**
- 3: RangeQuerySimple($T.\text{right}, k_1, k_2$)
- 4: **else if** $\text{key}(T.\text{root}) > k_2$ **then**
- 5: RangeQuerySimple($T.\text{left}, k_1, k_2$)
- 6: **else**
- 7: RangeQuerySimple($T.\text{left}, k_1, k_2$)
- 8: report $T.\text{root}$
- 9: RangeQuerySimple($T.\text{right}, k_1, k_2$)

Exercise 5

- ▶ Write a function which displays the values from a certain level of a tree (given as a param).
- ▶ Hint:
 - Display (T,level):
 - if(level==0) then print(T.info)
 - else Display (T.left, level-1); Display (T.right, level-1);

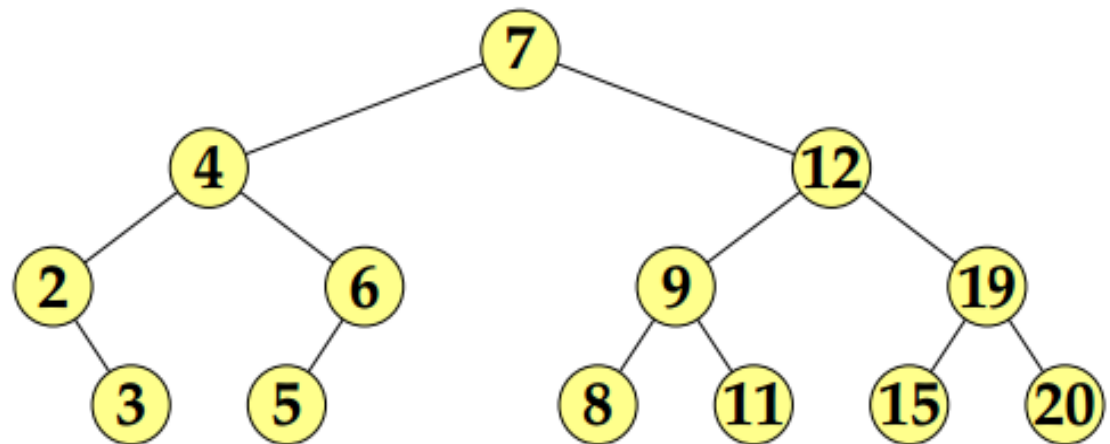
Exercise 6

- ▶ Use the previous functions to implement a new one to display the whole tree by levels.
- ▶ Ex: for this tree we will have:
 - Level 0: 7
 - Level 1: 4, 12
 - Level 2: 2, 6, 9, 19
 - Level 3: 3, 5, 8, 11, 15, 20



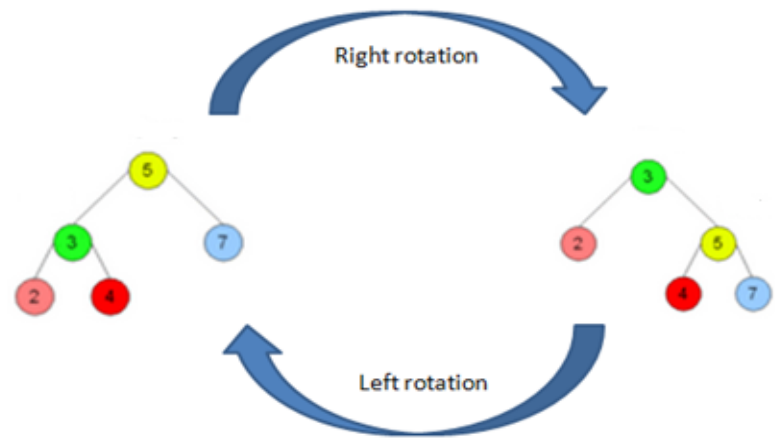
Bonus

- ▶ Given a tree, write a program to find the **lowest common ancestor (LCA)** of two given nodes in the tree.
- ▶ Ex: for this tree we will have:
 - $\text{LCA}(2, 5) = 4$
 - $\text{LCA}(3, 8) = 7$

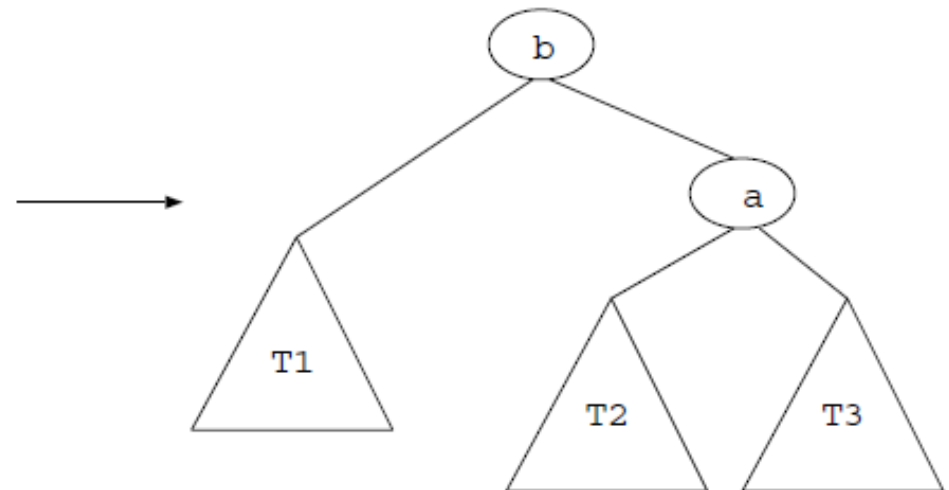
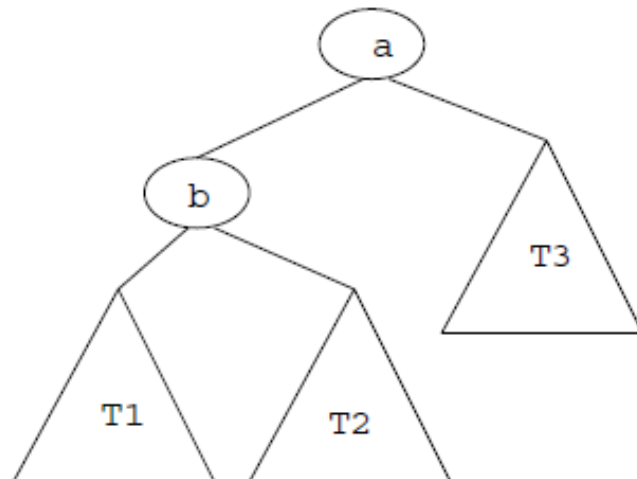
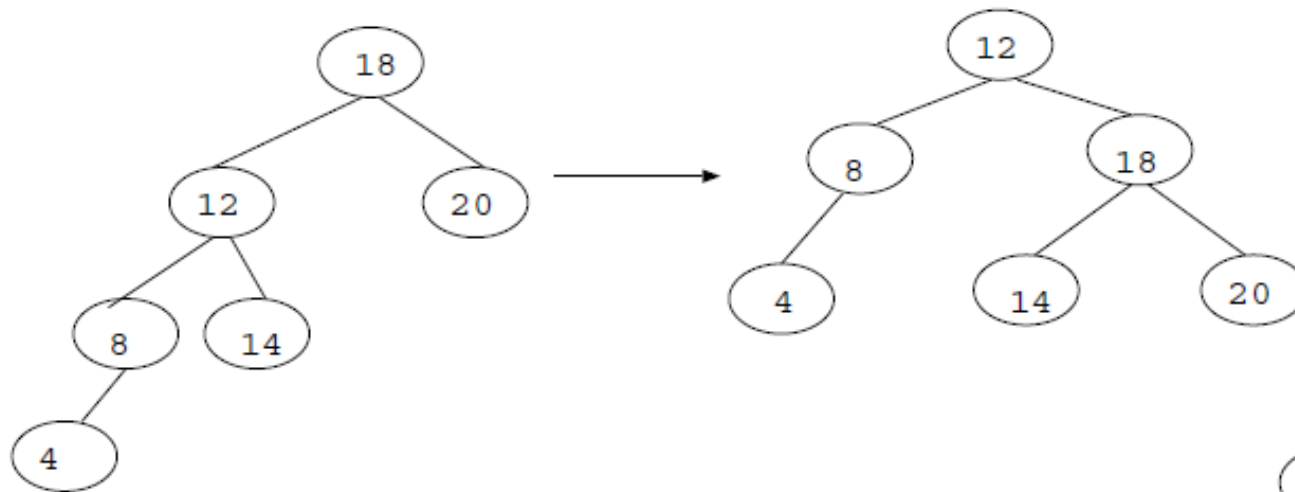


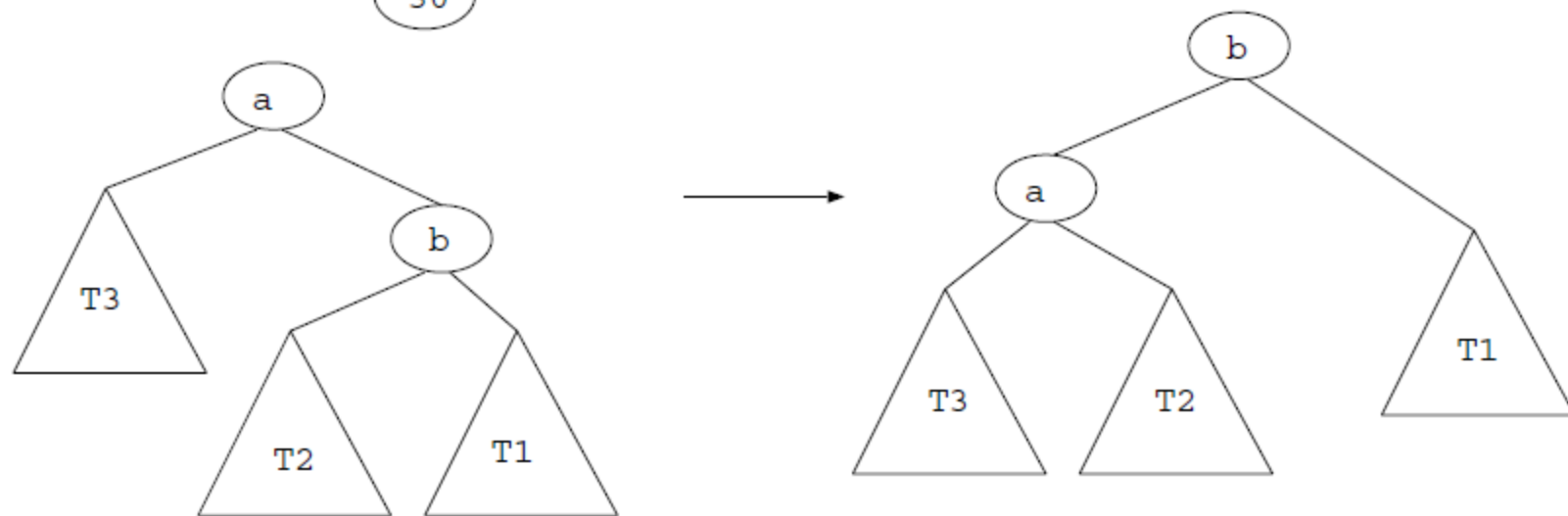
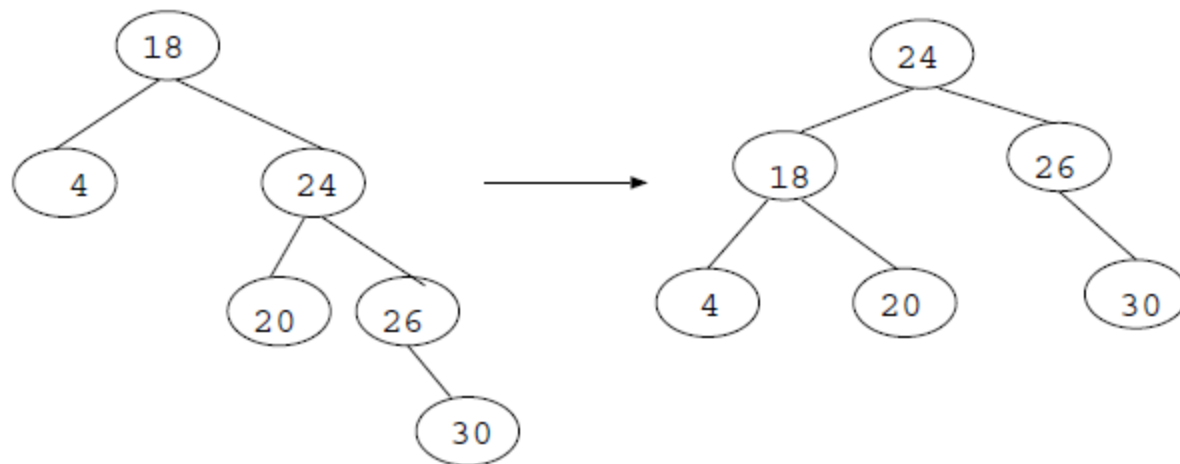
Bonus

- ▶ Implement a rotation function (not an instance method!) that rotates a binary search tree sent as an argument to the function. A rotation of the tree is an operation on a binary search tree that modifies the structure without interfering with the in order traversal of the elements.
- ▶ The rotation takes place as follows:
 - turn the binary search tree to the right, if the height of the right child is smaller than the size of the left child;
 - rotation of the binary search tree to the left, if the size of the left child is smaller than the height of the right child
 - nothing if the tree is well balanced.



Don't forget! We must respect the BST properties => same order of the leaves





Hint – right rotation

- ▶ Algorithm to rotate a node to the right
 - Left child becomes the new root
 - Right child of new root is assigned to left child of old root
 - Previous root becomes the new root's right child

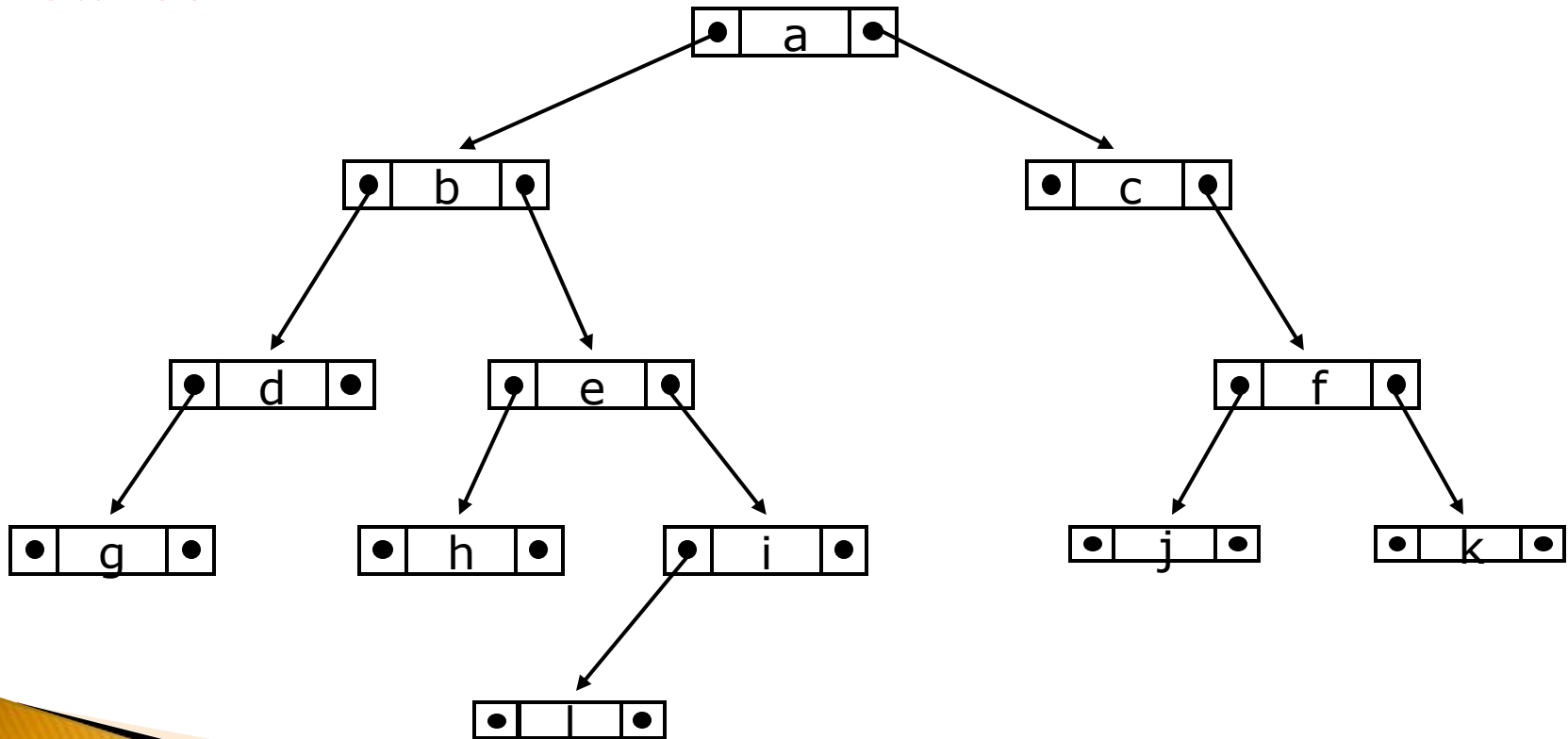
Hint – left rotation

- ▶ Algorithm to rotate a node to the right
 - Right child becomes the new root
 - Left child of new root is assigned to right child of old root
 - Previous root becomes the new root's left child


Theory

Binary trees (reminder)

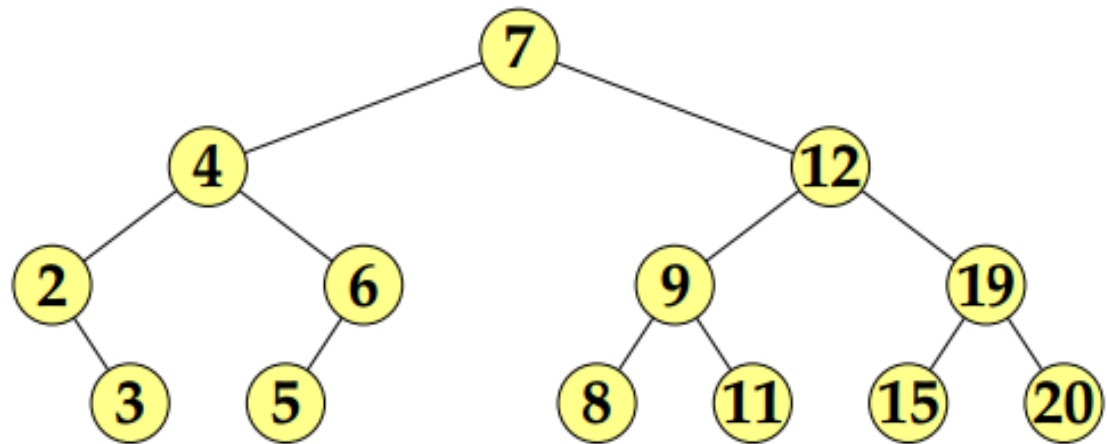
- ▶ If it is not empty, the binary tree has a **root node**.
- ▶ The nodes which have no children are called **leaves**.



Binary search trees

- ▶ Binary trees with additional properties:
 - The elements (infos) saved in the nodes are **comparable**
 - The elements from the **left sub-tree** of the node p are \leq than the element saved in p
 - The elements from the **right sub-tree** of the node p are $>$ than the element saved in p
 - Fast search
 - Complicated deletion (we must maintain the properties of the BST)
- 

Example

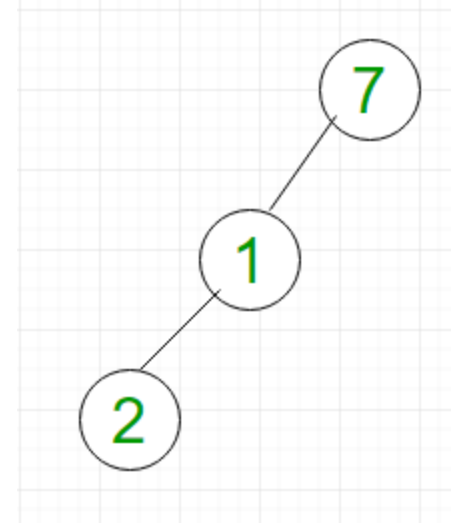


Pre-order (Root Left Right)	7, 4, 2, 3, 6, 5, 12, 9, 8, 11, 19, 15, 20.
In order (Left Root Right)	2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 15, 19, 20.
Post-order (Left Right Root)	3, 2, 5, 6, 4, 8, 11, 9, 15, 20, 19, 12, 7

In order traversal of a BST gives a sorted sequence of values

Complexity

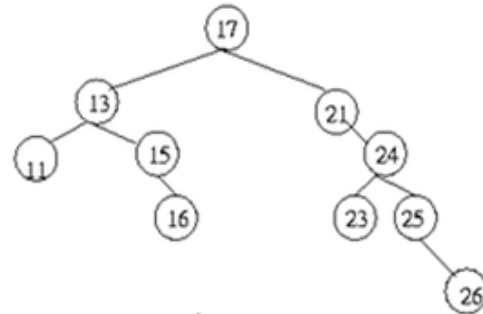
- ▶ **Searching** has worst case complexity of $O(n)$. In general, time complexity is $O(h)$ where h is height of BST.
- ▶ **Insertion** has worst case complexity of $O(n)$. In general, time complexity is $O(h)$.
- ▶ **Deletion** has worst case complexity of $O(n)$. In general, time complexity is $O(h)$.



BST – implementation

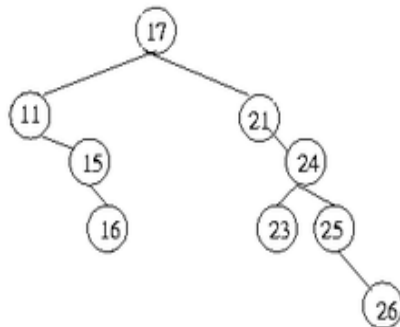
- ▶ Download the example on moodle
- ▶ Let's draw the BST from our example
- ▶ Delete node 'C' and see what happens

Reminder



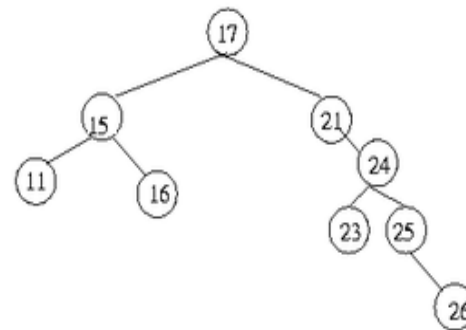
delete 13

/* delete node with both
left and right subtrees */



Method 1.

Find highest valued element
among the descendants of
left child



Method 2

Find lowest valued element
among the descendants of
right child

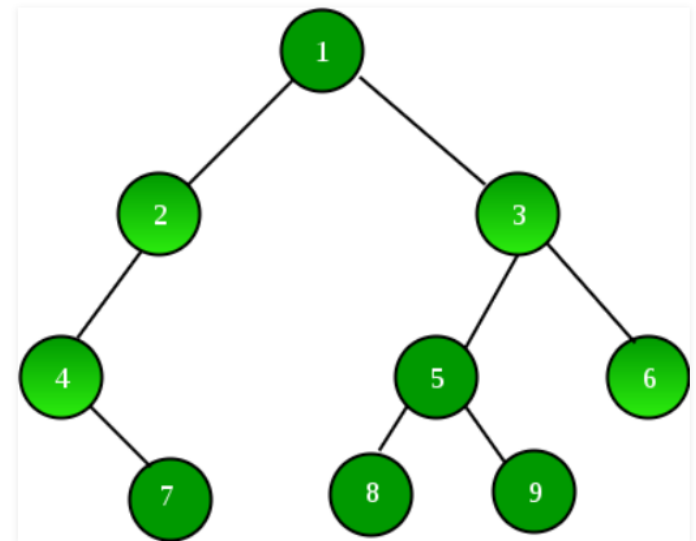
Extra exercise

- Find the mirror of a given node in a BST.

Have a look at:

<https://www.geeksforgeeks.org/find-mirror-given-node-binary-tree/>

Examples:



In above tree-

Node 2 and 3 are mirror nodes

Node 4 and 6 are mirror nodes.