

# **DTS203TC**

# **Design and Analysis of Algorithms**

## **Lecture 8: Data Structures**

Dr. Qi Chen and Pascal Lefevre  
School of AI and Advanced Computing

Pascal.Lefevre@xjtlu.edu.cn

Office hours: D5012

Wednesday/Thursday 14:00-16:00

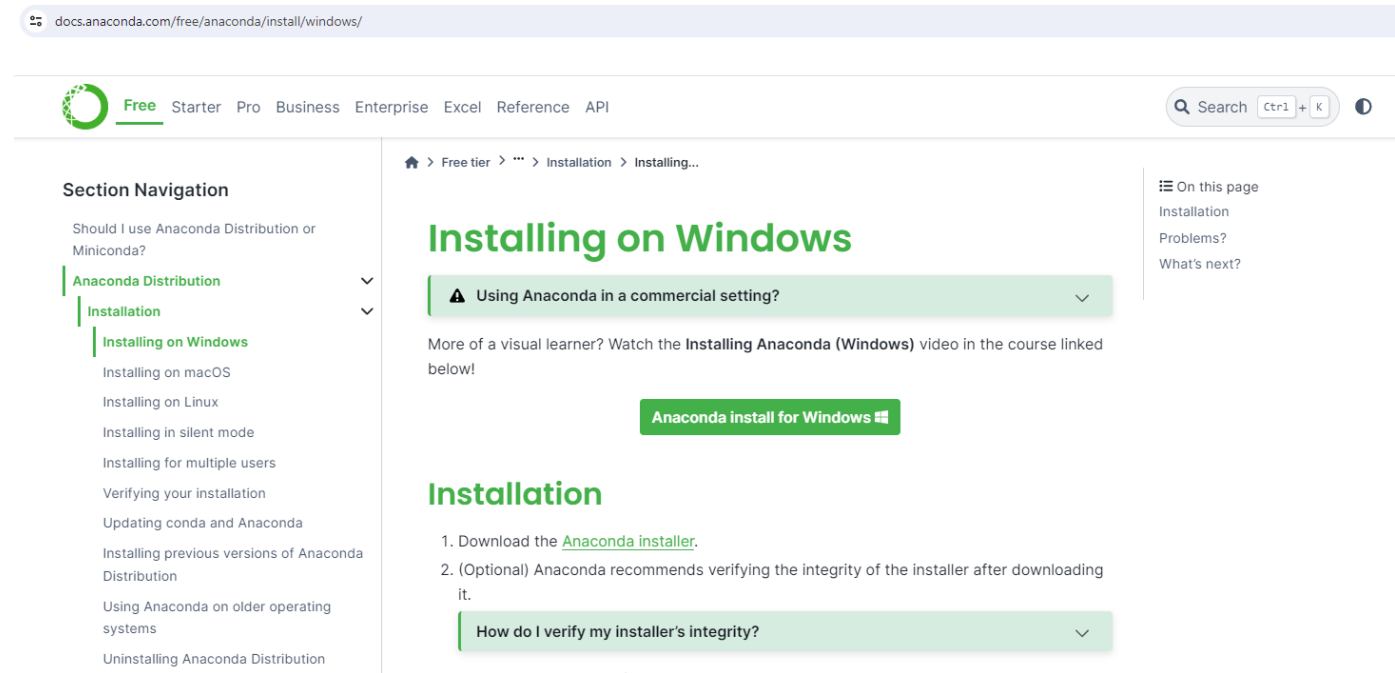
# Learning outcomes

- Installation tips
- Linked List
- Binary Search Tree
- AVL Tree

# Installation

- Install Anaconda:

<https://docs.anaconda.com/free/anaconda/install/windows/>



# Installation

- Install Jupyter notebook: <https://jupyter.org/install>

## Jupyter Notebook

Install the classic Jupyter Notebook with:

```
pip install notebook
```

To run the notebook:

```
jupyter notebook
```

# Installation

```
Anaconda Prompt - jupyter notebook
01-Mar-24 17:36 <DIR> Videos
0 File(s) 0 bytes
17 Dir(s) 793,941,274,624 bytes free

(base) C:\Users\Pascal.Lefevre>jupyter notebook
[I 12:37:01.635 NotebookApp] Writing notebook server cookie secret to C:\Users\Pascal.Lefevre\AppData\Roaming\jupyter\runtime\notebook_cookie_secret

Jupyter Notebook

Read the migration plan to Notebook 7 to learn about the new features and the actions to take if you are using extensions.
https://jupyter-notebook.readthedocs.io/en/latest/migrate_to_notebook7.html

Please note that updating to Notebook 7 might break some of your extensions.

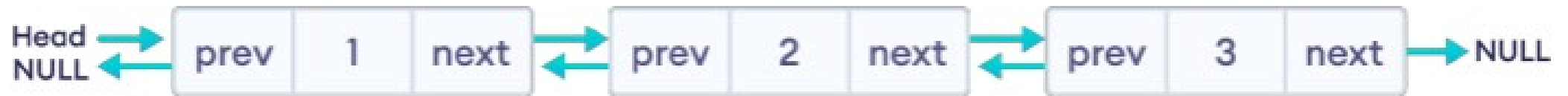
[W 12:37:02.916 NotebookApp] Loading JupyterLab as a classic notebook (v6) extension.
[C 12:37:02.916 NotebookApp] You must use Jupyter Server v1 to load JupyterLab as notebook extension. You have v2.5.0 installed.
You can fix this by executing:
    pip install -U "jupyter-server<2.0.0"
[I 12:37:04.838 NotebookApp] Serving notebooks from local directory: C:\Users\Pascal.Lefevre
[I 12:37:04.838 NotebookApp] Jupyter Notebook 6.5.4 is running at:
[I 12:37:04.838 NotebookApp] http://localhost:8888/?token=89014f9af5f6401a3b3062eb6222de4a9598047dec5ee322
[I 12:37:04.854 NotebookApp] or http://127.0.0.1:8888/?token=89014f9af5f6401a3b3062eb6222de4a9598047dec5ee322
[I 12:37:04.854 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 12:37:04.885 NotebookApp]

To access the notebook, open this file in a browser:
    file:///C:/Users/Pascal.Lefevre/AppData/Roaming/jupyter/runtime/nbserver-9156-open.html
Or copy and paste one of these URLs:
    http://localhost:8888/?token=89014f9af5f6401a3b3062eb6222de4a9598047dec5ee322
    or http://127.0.0.1:8888/?token=89014f9af5f6401a3b3062eb6222de4a9598047dec5ee322
This version of python seems to be incorrectly compiled
(internal generated filenames are not absolute).
This may make the debugger miss breakpoints.
Related bug: http://bugs.python.org/issue1666807
[I 12:46:33.190 NotebookApp] 302 GET / (127.0.0.1) 1.010000ms
[I 12:46:33.192 NotebookApp] 302 GET /tree? (127.0.0.1) 0.000000ms
[I 12:46:47.331 NotebookApp] 302 GET /?token=89014f9af5f6401a3b3062eb6222de4a9598047dec5ee322 (127.0.0.1) 2.550000ms
```

# Linked List

# Definition – Linked List

- Linear data structure + series of nodes
- Each node has a value
- Traversal: using pointers
- Types: singly, doubly, circular



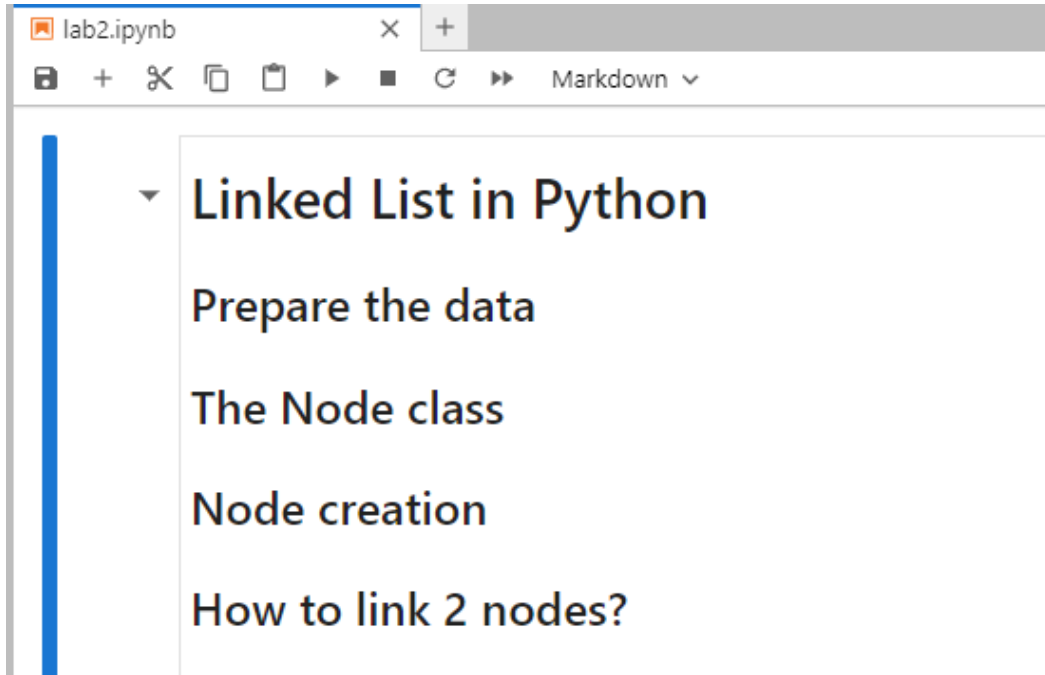


# Operations – Linked List

- Traversal
- Search
- Insertion, deletion



# Implementation – Linked List



## Create a Linked List

Singly

Doubly

## Manipulating Linked Lists

Traversal

Insertion

Deletion

See notebook lab2.ipynb

# Notebook – Linked List

- See: Lecture 8 - Data Structures - Notebook.ipynb

# Notebook – Linked List

## Linked List in Python

### The Node class

```
In [1]: class Node:
        def __init__(self, data):
            self.data = data
            self.prev = None
            self.next = None

        def __str__(self):
            return f'Node({self.data})'
```

### Node creation

```
In [2]: # a variable node contains the value 10
node = Node(10)
```

```
In [3]: # display value inside the node
print(node.value) # gives an error, why?
```

```
-----
AttributeError                                Traceback (most recent call last)
Cell In[3], line 2
      1 # display value inside the node
----> 2 print(node.value)

AttributeError: 'Node' object has no attribute 'value'
```

```
In [ ]: # display the pointers
print(node.prev, node.next)
# the node is not connected to others nodes
```

# Notebook – Linked List

## How to link 2 nodes?

```
In [ ]: # declare two nodes
n1 = Node(10)
n2 = Node(20)

n1.next = n2
```

```
In [ ]: print(n1.next)
```

```
In [ ]: # Link 2 nodes with 2 pointers
# how?

n2.prev = n1

prnt(n2.prev)
```

## Create a Linked List

```
In [ ]: # our Data
number_list = [10, 20, 4, 2]

# define the nodes
node_list = [Node(10), Node(20), Node(4), Node(2)]

# define using the List called number_list
node_list = [Node(data) for data in number_list]
```

## Singly

```
In [ ]: node_list[0].next = node_list[1]
node_list[1].next = node_list[2]
node_list[2].next = node_list[3]

# we start with the head
head = node_list[0]

print(head)
print(head.next)
print(head.next.next)
print(head.next.next.next)
print(head.next.next.next.next)
```

# Notebook – Linked List

## Doubly

```
In [ ]: # use the Node.prev method to link the nodes in the reverse order

tail = node_list[3]
tail.prev = node_list[2]
node_list[2].prev = node_list[1]
# and so on
```

## Manipulating Linked Lists

### Traversal

```
In [ ]: # traversal of a linked list using a loop
head = node_list[0]

while head is not None:
    print(head)
    head = head.next
```

```
In [ ]: # how about the search?

value = 4

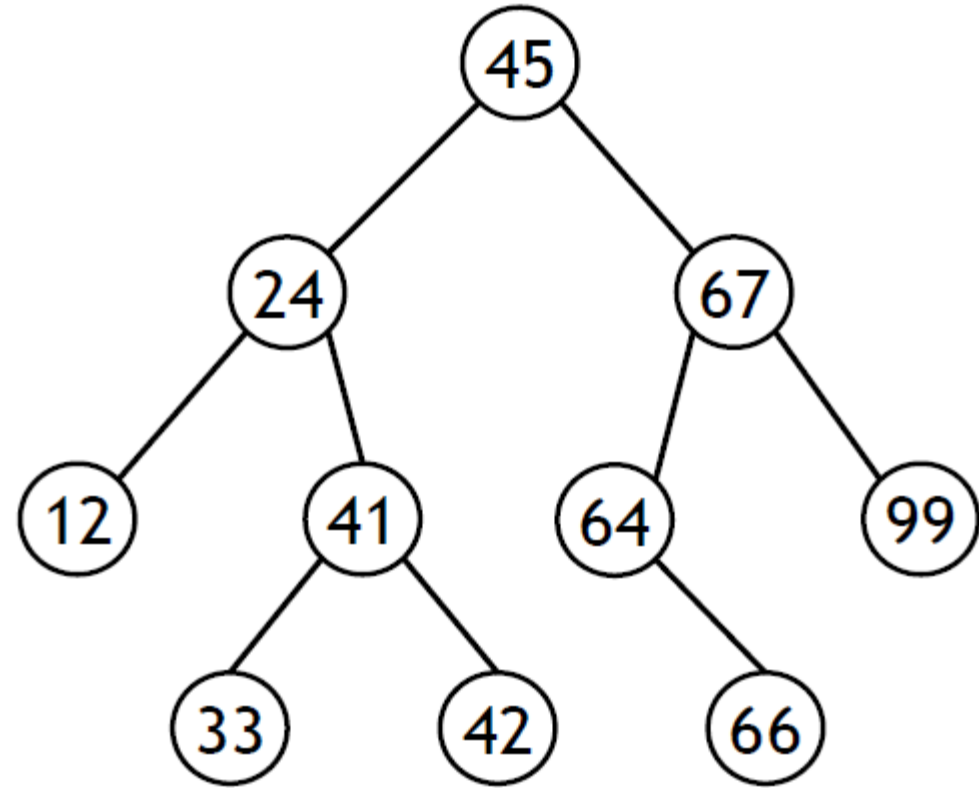
head = node_list[0]

while head is not None:
    print(head)
    # search here
    # if value .....
    # ....
    head = head.next
```

# Binary Search Tree

# Definition – Binary Search Tree

- Binary tree
- Each node has a key
- Two children at most
  - Left child
  - Right child

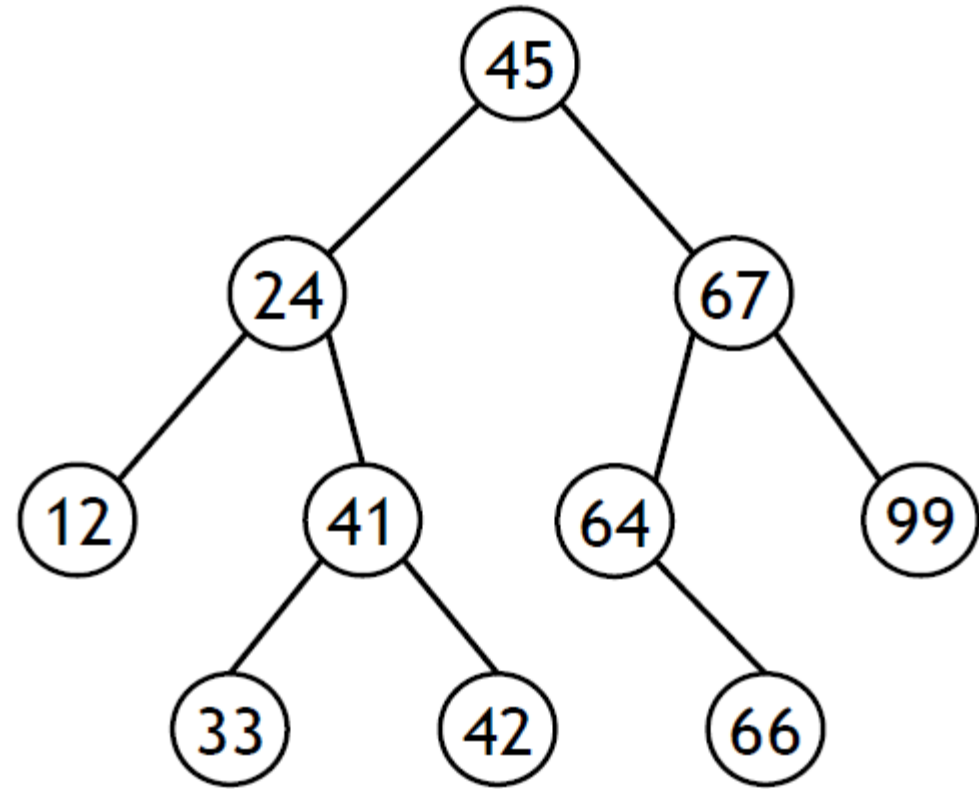


Node.left.key <= node.key <= node.right.key



# Operations – Binary Search Tree

- Traversal
- Search
- Insertion, deletion
- Find the Minimum and Maximum



# Notebook – Binary Search Tree

- See: Lecture 8 - Data Structures - Notebook.ipynb

# Notebook – Binary Search Tree

## The Node class for Linked List

```
In [ ]: class NodeLL:
        def __init__(self, data):
            self.data = data
            self.prev = None
            self.next = None

        def __str__(self):
            return f'NodeLL({self.data})'
```

## The Node class for BST

```
In [13]: class Node:
        def __init__(self, key):
            self.left = None
            self.right = None
            self.key = key

        def __str__(self):
            return f'Node({self.key})'
```

## Data - Create a BST

```
In [14]: # create 3 nodes and display them
a, b, c = 2, 3, 1

n1 = Node(a)
n2 = Node(b)
n3 = Node(c)

print(n1)
print(n2)
print(n3)
```

# Notebook – Binary Search Tree

## Link the nodes to get a BST

```
In [15]: # we start with a, then b and finally c
root = n1
root.left = n3
root.right = n2

# print the nodes from the root
print(root)
print(root.left, root.right)

Node(2)
Node(1) Node(3)
```

## Insert a key

```
In [16]: # Let's add the key 0
# first, create a node

new_node = Node(0)

# then add it to the BST, but how?
root.left.left = new_node

# because 0 < 1, so we insert at the left of Node(1)
```

```
In [17]: # Let's print the new_node
print(root.left.left)

Node(0)
```

```
In [18]: # Let's add another node, say a key 10
another_node = Node(10)

root.right.right = another_node
print(root.right.right)
```

# Notebook – Binary Search Tree

## Searching inside a BST

```
In [7]: # Search(x,k)
#       if x == NULL or k == x.key
#         return x
#       if k < x.key
#         return Search(x.left,k)
#       else
#         return Search(x.right,k)
```

```
In [19]: # x is the root of BST
# k is the key vaue to search

def search(x, k):
    if x == None or k == x.key:
        return x
    if k < x.key:
        return search(x.left, k)
    else:
        return search(x.right, k)
```

```
In [26]: print(root)

# can we find element -1?
element_found = search(root, -1)

print(element_found) # return None, so we cannot find -1

Node(2)
None
```

```
In [ ]: # Can we find 10?
element_found = search(root, 10)

print(element_found) # return 10, so yes we found the element
```

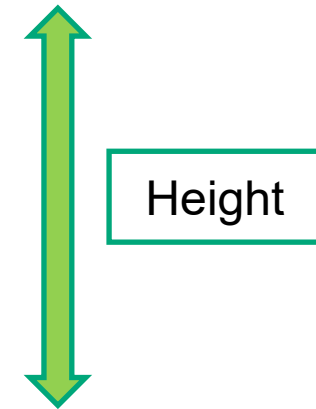
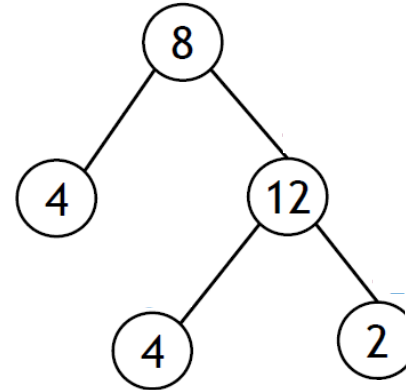
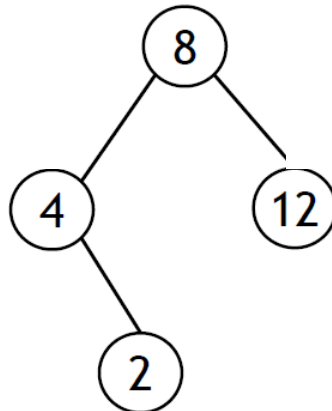
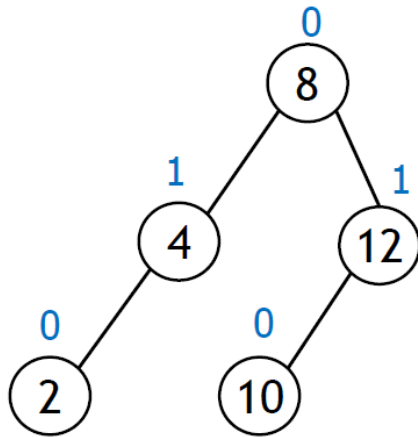
# AVL Tree

# Definition – AVL Tree

- BST
- Height-balanced

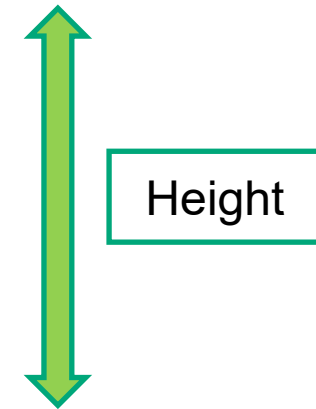
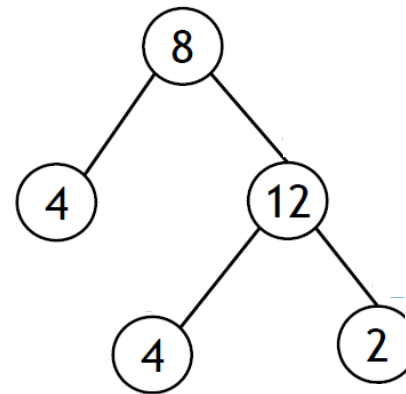
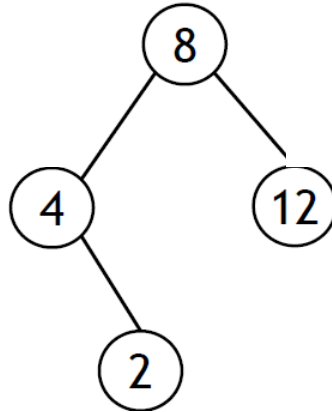
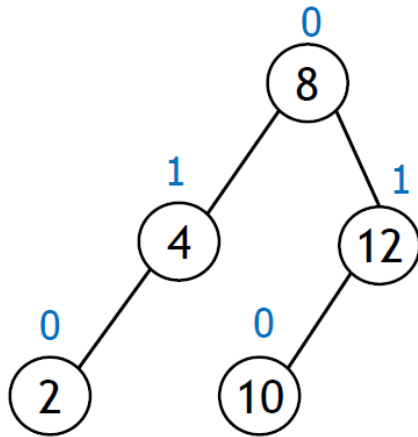
Balance factor of a node =  $\{-1, 0, 1\}$

Balance factor = height of left subtree - height of right subtree



# Definition – AVL Tree

- Rotation
- Insertion
- Deletion





# Notebook – AVL Tree

- See: Lecture 8 - Data Structures - Notebook.ipynb

# Notebook – AVL Tree

## AVL Tree

```
In [28]: class Node:
          def __init__(self, key):
              self.key = key
              self.left = None
              self.right = None
              self.height = 1
```

```
In [29]: n1 = Node(10)
          n2 = Node(20)
          n3 = Node(30)
```

```
In [ ]: root = Node(20)
```

```
In [ ]: # create a BST
```

```
In [ ]: # create an AVL tree
          # it needs to be balanced
          # we need to adjust the height
          # calculate the balance factor for each node
          # example
          # 8
          # 4 12
          # 2 null 10 null
```

# Notebook – AVL Tree

```
In [ ]: # perform a single rotation
        # initially: 10, 20
        # insert 30 in this BST (the tree becomes imbalanced)
        # perform a Left Rotation

        root = Node(30)
        root.left = Node(10)

        # insert 30
        # perform a search for 30, if 30 is not in this AVL tree, we insert at the suitable position in the BST
        # reuse the previous search algorithm for BST

        # perform Left Rotation
        # how to do it manually?
```

```
In [ ]: # perform a Right Rotation
```

```
In [ ]: # perform a Double Rotation: Right-Left Rotation
        # it is 1 rotation in 2 steps
```

Any questions?