 Data visualization course

**Laboratory work 4**

# Binary tree visualization

A binary tree is a structure in which each node (or vertex) has at most two descendant nodes and exactly one parent. The topmost node of the tree is the only node with no parents; it is called the root node. A binary tree with N nodes has at least [log2N + 1] levels (with maximally dense packing of nodes). If the levels of the tree are numbered, assuming that the root lies at level 1, then at the level with number *K* lies *2K-1* node. In a complete binary tree with *j* levels (numbered from 1 to *j*), all leaves lie at the level with number *j*, and each node at levels from the first to *j* has exactly two direct descendants. In a complete binary tree with *j* levels, *2j* is 1 node.

You can use the code in Listing 1 to build a binary tree using python.

Listing 1 – Implementation of a binary tree in python

class Node:

    def \_\_init\_\_(self, key):

        self.key = key

        self.left = None

        self.right = None

        self.parent = None

class Tree:

    def \_\_init\_\_(self):

        self.root = None

    def add\_node(self, key, node=None):

        if node is None:

            node = self.root

        if self.root is None:

            self.root = Node(key)

        else:

            if key <= node.key:

                if node.left is None:

                    node.left = Node(key)

                    node.left.parent = node

                    print("left")

                    return

                else:

                    return self.add\_node(key, node=node.left)

            else:

                if node.right is None:

                    node.right = Node(key)

                    node.right.parent = node

                    print("right")

                    return

                else:

                    return self.add\_node(key, node=node.right)

    def search(self, key, node=None):

        if node is None:

            node = self.root

        if self.root.key == key:

            print("key is at the root")

            return self.root

        else:

            if node.key == key:

                print("key exists")

                return node

            elif key < node.key and node.left is not None:

                print("left")

                return self.search(key, node=node.left)

            elif key > node.key and node.right is not None:

                print("right")

                return self.search(key, node=node.right)

            else:

                print("key does not exist")

                return None

    def delete\_node(self, key, node=None):

        if node is None:

            node = self.search(key)

        if self.root.key == node.key:

            parent\_node = self.root

        else:

            parent\_node = node.parent

        if node.left is None and node.right is None:

            if key <= parent\_node.key:

                parent\_node.left = None

            else:

                parent\_node.right = None

            return

        if node.left is not None and node.right is None:

            if node.left.key < parent\_node.key:

                parent\_node.left = node.left

            else:

                parent\_node.right = node.left

            return

        if node.right is not None and node.left is None:

            if node.key <= parent\_node.key:

                parent\_node.left = node.right

            else:

                parent\_node.right = node.right

            return

        if node.left is not None and node.right is not None:

            min\_value = self.find\_minimum(node)

            node.key = min\_value.key

            min\_value.parent.left = None

            return

    def find\_minimum(self, node=None):

        if node is None:

            node = self.root

        if node.right is not None:

            node = node.right

        else:

            return node

        if node.left is not None:

            return self.find\_minimum(node=node.left)

        else:

            return node

    def tree\_data(self, node=None):

        if node is None:

            node = self.root

        stack = []

        while stack or node:

            if node is not None:

                stack.append(node)

                node = node.left

            else:

                node = stack.pop()

                yield node.key

                node = node.right

t = Tree()

t.add\_node(10)

t.add\_node(13)

t.add\_node(14)

t.add\_node(8)

t.add\_node(9)

t.add\_node(7)

t.add\_node(11)

# Task

Create a binary tree according to the student number and perform its visualization with D. Knuth's algorithm.

1. Real numbers in the range [10, 50];
2. Integers in the range [-50, 50];
3. Real numbers in the range [100, 200];
4. Integers in the range [-500, 0];
5. Real numbers in the range [75, 300];
6. Integers in the range [-250, 300];
7. Real numbers in the range [-1, 1];
8. Integers in the range [-1000, 0];
9. Real numbers in the range [0, 2];