Візуалізація даних

**Лабораторна робота №4**

# Візуалізація бінарних дерев

Бінарне дерево являє собою структуру, в якій кожен вузол (або вершина) має не більше двох вузлів-нащадків і в точності одного з батьків. Самий верхній вузол дерева є єдиним вузлом без батьків; він називається кореневим вузлом. Бінарне дерево з *N* вузлами має не менше [log2N + 1] рівнів (при максимально щільній упаковці вузлів). Якщо рівні дерева занумерувати, вважаючи що корінь лежить на рівні 1, то на рівні з номером *К* лежить 2*К*-1 вузол. У повного бінарного дерева з *j* рівнями (пронумеровані від 1 до j) всі листи лежать на рівні з номером *j*, і у кожного вузла на рівнях з першого по *j* – 1 в точності два безпосередніх нащадка. У повному бінарному дереві з *j* рівнями 2*j* - 1 вузол.

Для побудови бінарного дерева засобами python можно використовувати код, наведений у лістингу 1.

Лістинг 1 – Реалізація бінарного дерева на 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):

# search for the node to be deleted in tree

if node is None:

node = self.search(key) # return the node to be deleted

# root has no parent node

if self.root.key == node.key: # if it is root

parent\_node = self.root

else:

parent\_node = node.parent

'''case 1: The node has no chidren'''

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

'''case 2: The node has children'''

''' if it has a single left node'''

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 it has a single right node'''

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 it has two children'''

'''find the node with the minimum value from the right subtree.

copy its value to thhe node which needs to be removed.

right subtree now has a duplicate and so remove it.'''

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

'''find mimimum value from the right subtree'''

'''case when there is only a root node'''

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)

# Варіанти завдань

Створити бінарне дерево згідно варіанту та виконати його візуалізацію алгоритмом Д. Кнута.

**Варіант 1**

Дійсні числа в діапазоні [10, 50];

**Варіант 2**

Цілі числа в діапазоні [-50, 50];

**Варіант 3**

Дійсні числа в діапазоні [100, 200];

**Варіант 4**

Цілі числа в діапазоні [-500, 0];

**Варіант 5**

Дійсні числа в діапазоні [75, 300];

**Варіант 6**

Цілі числа в діапазоні [-250, 300];

**Варіант 7**

Дійсні числа в діапазоні [-1, 1];

**Варіант 8**

Цілі числа в діапазоні [-1000, 0];

**Варіант 9**

Дійсні числа в діапазоні [0, 2];