

**“AZƏRBAYCAN HAVA YOLLARI” CJSC NATIONAL AVIATION ACADEMY**

**Individual Work № 7:**

**Topic: Binary trees with examples in Python**

**Subject: Obyektyönümlü proqramlaşdırma**

**Teacher: Mehemmed Shahmaliyev**

**Group: 1459i Student: Hesenova Firuze**

**Date: Signature:H.F**

**Baku 2022**

**Binarytree Module in Python**

A binary tree is a data structure in which every node or vertex has atmost two children. In Python, a binary tree can be represented in different ways with different data structures(dictionary, list) and class representation for a node. However, binarytree library helps to directly implement a binary tree. It also supports heap and binary search tree(BST). This module does not come pre-installed with Python’s standard utility module. To install it type the below command in the terminal.

*pip install binarytree*

Creating Node

The node class represents the structure of a particular node in the binary tree. The attributes of this class are values, left, right.

Syntax: binarytree.Node(value, left=None, right=None)  
Parameters:   
value: Contains the data for a node. This value must be number.   
left: Conatins the details of left node child.   
right: Contains details of the right node child. 

Note: If left or right child node is not an instance of binarytree.Node class then binarytree.exceptions.NodeTypeError is raised and if the node value is not a number then binarytree.exceptions.NodeValueError is raised.  
Example:

from binarytree import Node

root = Node(3)

root.left = Node(6)

root.right = Node(8)

# Getting binary tree

print('Binary tree :', root)

# Getting list of nodes

print('List of nodes :', list(root))

# Getting inorder of nodes

print('Inorder of nodes :', root.inorder)

# Checking tree properties

print('Size of tree :', root.size)

print('Height of tree :', root.height)

# Get all properties at once

print('Properties of tree : \n', root.properties)

Output:

Binary tree :   
3   
/ \   
6 8  
List of nodes : [Node(3), Node(6), Node(8)]  
Inorder of nodes : [Node(6), Node(3), Node(8)]  
Size of tree : 3  
Height of tree : 1  
Properties of tree :   
{‘height’: 1, ‘size’: 3, ‘is\_max\_heap’: False, ‘is\_min\_heap’: True, ‘is\_perfect’: True, ‘is\_strict’: True, ‘is\_complete’: True, ‘leaf\_count’: 2, ‘min\_node\_value’: 3, ‘max\_node\_value’: 8, ‘min\_leaf\_depth’: 1, ‘max\_leaf\_depth’: 1, ‘is\_bst’: False, ‘is\_balanced’: True, ‘is\_symmetric’: False}

Build a binary tree from the List:

Instead of using the Node method repeatedly, we can use build() method to convert a list of values into a binary tree.   
Here, a given list contains the nodes of tree such that the element at index i has its left child at index 2\*i+1, the right child at index 2\*i+2 and parent at (i – 1)//2. The elements at index j for j>len(list)//2 are leaf nodes. None indicates the absence of a node at that index. We can also get the list of nodes back after building a binary tree using values attribute.

Syntax: binarytree.build(values)  
Parameters:   
values: List representation of the binary tree.  
Returns: root of the binary tree. 

Example:

# Creating binary tree

# from given list

from binarytree import build

# List of nodes

nodes =[3, 6, 8, 2, 11, None, 13]

# Building the binary tree

binary\_tree = build(nodes)

print('Binary tree from list :\n',

binary\_tree)

# Getting list of nodes from

# binarytree

print('\nList from binary tree :',

binary\_tree.values)

Output:

Binary tree from list :

\_\_\_3

/ \

6 8

/ \ \

2 11 13

List from binary tree : [3, 6, 8, 2, 11, None, 13]

Build a random binary tree:

tree() generates a random binary tree and returns its root node.

Syntax: binarytree.tree(height=3, is\_perfect=False)  
Parameters:   
height: It is the height of the tree and its value can be between the range 0-9 (inclusive)   
is\_perfect: If set True a perfect binary is created.  
Returns: Root node of the binary tree. 

Example:

from binarytree import tree

# Create a random binary

# tree of any height

root = tree()

print("Binary tree of any height :")

print(root)

# Create a random binary

# tree of given height

root2 = tree(height = 2)

print("Binary tree of given height :")

print(root2)

# Create a random perfect

# binary tree of given height

root3 = tree(height = 2,

is\_perfect = True)

print("Perfect binary tree of given height :")

print(root3)

Output:

Binary tree of any height :

14\_\_\_\_

/ \

2 5\_\_

/ / \

6 1 13

/ / / \

7 9 4 8

Binary tree of given height :

1\_\_

/ \

5 2

/ \

4 3

Perfect binary tree of given height :

\_\_3\_\_

/ \

2 4

/ \ / \

6 0 1 5

Building a BST:

The binary search tree is a special type of tree data structure whose inorder gives a sorted list of nodes or vertices. In Python, we can directly create a BST object using binarytree module. bst() generates a random binary search tree and return its root node.

Syntax: binarytree.bst(height=3, is\_perfect=False)  
Parameters:   
height: It is the height of the tree and its value can be between the range 0-9 (inclusive)   
is\_perfect: If set True a perfect binary is created.  
Returns: Root node of the BST. 

Example:

from binarytree import bst

# Create a random BST

# of any height

root = bst()

print('BST of any height : \n',

root)

# Create a random BST of

# given height

root2 = bst(height = 2)

print('BST of given height : \n',

root2)

# Create a random perfect

# BST of given height

root3 = bst(height = 2,

is\_perfect = True)

print('Perfect BST of given height : \n',

root3)

Output:

BST of any height :

\_\_\_\_9\_\_\_\_\_\_

/ \

\_\_5\_\_ \_\_\_\_12\_\_\_

/ \ / \

2 8 10 \_14

/ \ / \ /

1 4 7 11 13

BST of given height :

5

/ \

4 6

/

3

Perfect BST of given height :

\_\_3\_\_

/ \

1 5

/ \ / \

0 2 4 6

Importing heap:

Heap is a tree data structure that can be of two types –

max heap

min heap 

Using the heap() method of binarytree library, we can generate a random maxheap and return its root node. To generate minheap, we need to set the is\_max attribute as False.

Syntax: binarytree.heap(height=3, is\_max=True, is\_perfect=False)  
Parameters:   
height: It is the height of the tree and its value can be between the range 0-9 (inclusive)   
is\_max: If set True generates a max heap else min heap.   
is\_perfect: If set True a perfect binary is created.  
Returns: Root node of the heap.