Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет инфокоммуникационных технологий

ОТЧЕТ

по лабораторной работе №2. «Двоичные деревья поиска» по дисциплине «Алгоритмы и структуры данных»

Выполнил студент 1 курса, группы К3140

Байков Иван

Преподаватель: Харьковская Татьяна Александровна

05 Июня, 2022 года, г. Санкт-Петербург

Задание 1

Описание

В этой задаче вы реализуете три основных способа обхода двоичного дерева «в глубину»: центрированный (in-order), прямой (pre-order) и обратный (post-order).

Исходный код

```
class BinaryTreeNode:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None


class BinaryTree:
    def __init__(self, root_data):
        self.root = BinaryTreeNode(root_data)

    def __str__(self):
        preorder_recursive = str(self.preorder_dfs_recursive(self.root))
        inorder_recursive = str(self.inorder_dfs_recursive(self.root))
        postorder_recursive = str(self.postorder_dfs_recursive(self.root))

        return f"""
            \nPre-Order\n{preorder_recursive}
            \nIn-Order\n{inorder_recursive}
            \nPost-Order\n{postorder_recursive}
        """

    def preorder_dfs_recursive(self, start_node, result=[]):
        if start_node is not None:
            result.append(start_node.data)
            self.preorder_dfs_recursive(start_node.left, result)
            self.preorder_dfs_recursive(start_node.right, result)
        return result

    def inorder_dfs_recursive(self, start_node, result=[]):
        if start_node is not None:
```

```
    def postorder_dfs_recursive(self, start_node, result=[]):
        if start_node is not None:
            self.postorder_dfs_recursive(start_node.left, result)
            self.postorder_dfs_recursive(start_node.right, result)
            result.append(start_node.data)
        return result


#          2
#        /   \
#       1     3

binary_tree = BinaryTree(2)

binary_tree.root.left = BinaryTreeNode(1)
binary_tree.root.right = BinaryTreeNode(3)

print(binary_tree)
```

Решение

Мы обходим дерево по определенным порядкам, стартовой точкой (entry point) всегда является корень, но он не обязан быть первым в списке обхода. Порядок обходов на рисунке:

Тесты

```
Pre-Order
[2, 1, 3]

In-Order
[1, 2, 3]

Post-Order
[1, 3, 2]
```

Взял простое дерево [2,1,3], чтобы легче показать на примере.

Вывод

Было полезно попрактиковаться в реализации алгоритмов обхода, чтобы лучше понимать такую полезную структуру данных и как с ней взаимодействовать.

Задание 6-7

Описание

В данной задаче необходимо проверить, правильно ли реализована структура BST. В 7 задаче усложнение – деревья могут содержать равные ключи.

Исходный код

```python
from typing import List
import sys


class TreeNode():
    root: int
    children: List[int]

    def __init__(self, root, left, right):
        self.root = root
        self.children = [left, right]


tree = []
_input = """3
2 1 2
1 -1 -1
3 -1 -1"""

_input = _input.split("\n")
n = int(_input[0])


def is_valid(tree, node, min_root, max_root):
    if node == -1:
        return True
    if tree[node].root <= min_root or tree[node].root >= max_root:
        return False

    result_left = is_valid(tree, tree[node].children[0], min_root, tree[node].root)
    result_right = is_valid(tree, tree[node].children[1], tree[node].root, max_root)
```

```
    result_left = is_valid(tree, tree[node].children[0], min_root, tree[node].root)
    result_right = is_valid(tree, tree[node].children[1], tree[node].root, max_root)

    return result_left and result_right


for line in _input[1:]:
    root, left, right = list(map(int, line.split(" ")))
    tree.append(TreeNode(root, left, right))

if n == 0:
    print("CORRECT")
else:
    if is_valid(tree, 0, -sys.maxsize - 1, sys.maxsize + 1):
        print("CORRECT")
    else:
        print("INCORRECT")
```

Решение

При выполнении In-order DFS каждый текущий узел во время обхода можно добавлять в массив, тогда в конце обхода можно наглядно убедиться, является ли данное дерево двоичным деревом поиском. Суть в том, что при таком обходе, каждый следующий элемент массива будет больше или равен(7) предыдущему, поэтому простым циклом можно это проверить.

Тесты

```
tree = []
_input = """3
2 1 2
1 -1 -1
3 -1 -1"""
```

```
CORRECT
```

```
tree = []
_input = """3
2 1 2
1 -1 -1
2 -1 -1"""
```

```
INCORRECT
```

Были проверены два теста из условия, один на CORRECT, другой на INCORRECT.

Вывод

Не самая сложная задача, если решать через рекурсивный подход. Помогает детальнее изучить BST и узнать некоторые свойства. Асимптотика алгоритма – O(n), где n – количество узлов.

Задание 11-15

Описание

Исходный код

Решение

Тесты

Вывод