Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки

Кафедра ІСТ

Звіт

з лабораторної роботи № 6 з дисципліни

«Теорія алгоритмів»

“Бінарні дерева пошуку”

Виконали ІА-34 Потейчук Софія Андріївна, Тунік Олександр Ігорович, Швець Роман Вадимович

Перевірив Степанов Андрій Сергійович

Київ 2024

Завдання

У даній роботі необхідно виконати два завдання.

Завдання 1. Перетворити вхідне бінарне дерево у бінарне дерево пошуку

На вхід подається деяке бінарне дерево, із фіксованою структурою (тобто зв’язками між вузлами, їх батьком та нащадками). Необхідно переписати значення вузлів дерева таким чином, щоби:

a) їх нові значення брались тільки з того набору, який присутній у вхідному дереві;

b) зберігалась внутрішня структура дерева (зв’язки між вузол-батько та вузол-нащадки).

Наприклад, нехай задане наступне дерево:

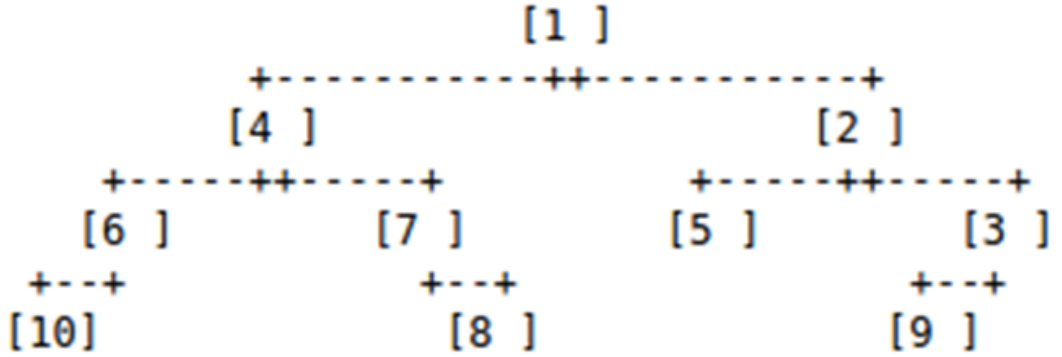

Рис. 1.

В ньому присутні 10 вузлів із значеннями від 1 до 10. Необхідно переписати значення вузлів так, щоби структура дерева зберігалась, але воно стало бінарним деревом пошуку:

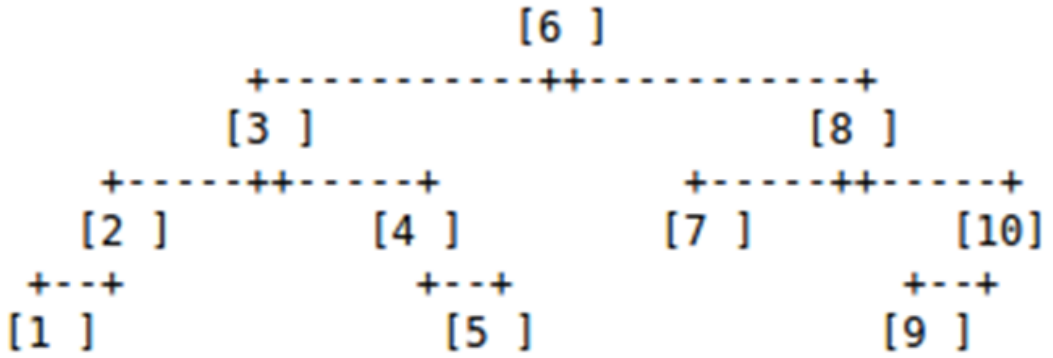

Рис. 2.

Як бачимо, для цього дерева (рис. 2) виконується умова бінарних дерев пошуку.

Маємо ще один приклад вхідного дерева:

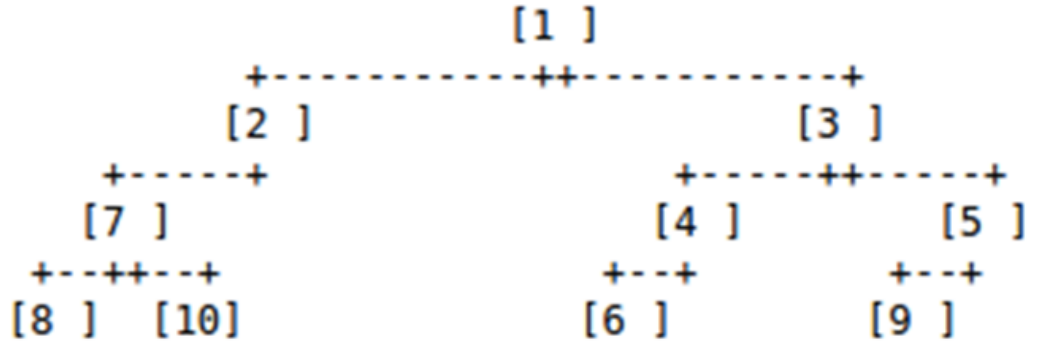

Рис. 3.

та відповідне йому бінарне дерево пошуку:

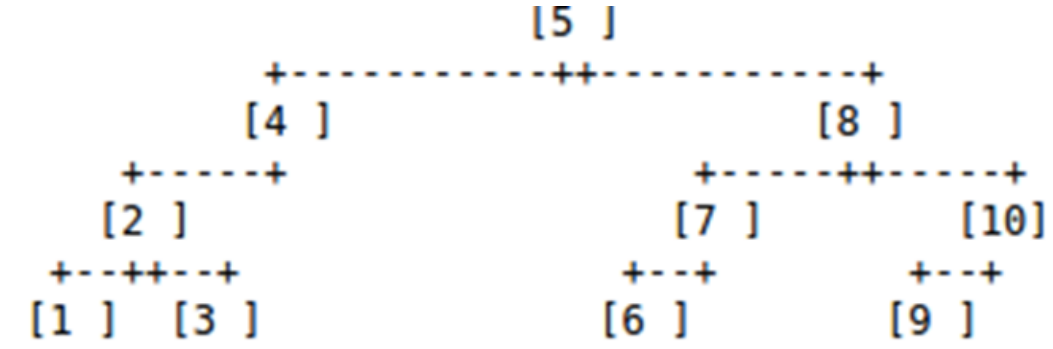

Рис. 4.

Розв’язати цю задачу можна за допомогою наступного алгоритму:

1. Обійти задане дерево у внутрішньому порядку (in-order) та зберігати всі значення в масиві.

2. Відсортувати масив у зростаючому порядку.

3. Знову обійти дерево у внутрішньому порядку та послідовно вписати значення відсортованого масиву у вузли дерева за порядком обходу.

Завдання 2. Пошук сум послідовних вузлів в дереві

Після того, як вхідне дерево перетворене на бінарне дерево пошуку, необхідно розв’язати наступну задачу. Додатково задається деяке число S. В отриманому бінарному дереві пошуку необхідно знайти всі такі монотонні шляхи (які не обов'язково йдуть від кореня, але всі прямують згори вниз), що сума значень вузлів, які належать знайденим шляхам, дорівнює числу S.

Наприклад, для першого прикладу бінарного дерева (Рис. 2) і для S = 9 маємо три монотонні шляхи в дереві, сума вузлів яких утворює 9: 6+3, 4+5 та окремо вузол 9.

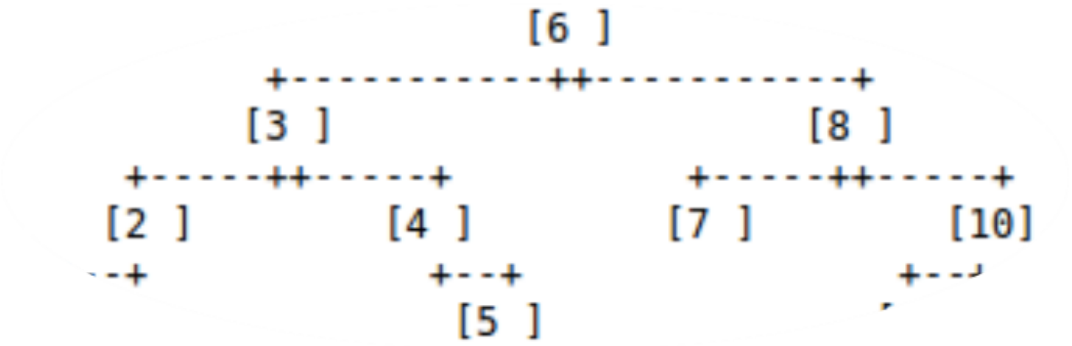

Рис 5.

Для дерева на рис. 4 та S = 9 маємо також три монотонних шляхи: 4+2+3, 5+4 та окремо вузол 9.

Формат вхідних/вихідних даних

Розроблена програма повинна зчитувати вхідні дані з файлу заданого формату та записувати дані у файл заданого формату. У вхідному файлі зберігається задане бінарне дерево

Вхідний файл представляє собою текстовий файл, в якому в один рядок записані всі вузли дерева, якщо обходити його у прямому порядку (pre-order tree walk). У цей обхід також додаються нульові значення для того, щоб позначити відсутність тих або інших листків дерева.

Наприклад, дерево на рис. 1 має наступний вхідний файл:

```
1 4 6 10 0 0 0 7 0 8 0 0 2 5 0 0 3 9 0 0 0
```

Перший елемент — завжди корінь. Наступний елемент — його лівий нащадок 4. Потім йде рекурсивний запис елементів вже для лівого нащадку кореня (вузол 4) — його лівий нащадок 6 і т.д. Вузол 10 є листком, тож щоб позначити це у вхідному файлі зазначається, що обидва його потенційних нащадки відсутні — після 10 йде два нулі (0 0). Далі у файлі йде ще один 0, який вказує на відсутність правого нащадка для вузла 6.

У наведеному вище дереві вказані нульові елементи ({0}), які позначають відсутність нащадку.

Ім’я вхідного файлу та число суми S передається в якості аргументів командного рядку.

Вихідний файл є текстовим. Кожен його рядок містить одну монотонну послідовність вузлів.

Числа в рядку записані через пробіл. Так для наведеного вище дерева (рис. 1) та S = 9, як вже зазначалось, маємо три послідовності: 6+3, 4+5 та окремо вузол 9. Тож вихідний файл буде мати вигляд:

```
4 5

9

6 3
```

Приклад вихідного файлу для дерева з рис. 1.

Псевдокод

Завдання 1:

```
construct_tree_from_preorder(preorder)
    if preorder is empty
        return None
    val = preorder.pop(0)
    if val == 0
        return None
    node = TreeNode(val)
    node.left = construct_tree_from_preorder(preorder)
    node.right = construct_tree_from_preorder(preorder)
    return node

inorder_traversal(root)
    if root is None
        return []
    return inorder_traversal(root.left) + [root.val] +
inorder_traversal(root.right)
```

```
rebuild_binary_search_tree(root, values)
    if root is None
        return
    val_iter = iter(values)
    rebuild_binary_search_tree_helper(root, val_iter)

rebuild_binary_search_tree_helper(node, val_iter)
    if node is None
        return
    rebuild_binary_search_tree_helper(node.left, val_iter)
    node.val = next(val_iter)
    rebuild_binary_search_tree_helper(node.right, val_iter)

read_preorder_from_file(filename)
    open filename for reading as f
        preorder = list(map(int, f.read().split()))
    return preorder

transformToBinarySearchTree(input_file)
    preorder = read_preorder_from_file(input_file)
    root = construct_tree_from_preorder(preorder)
    inorder_values = sorted(set(val for val in
inorder_traversal(root) if val != 0))
    rebuild_binary_search_tree(root, inorder_values)
    return root
```

Завдання 2:

```
find_paths_with_sum(root, target_sum, current_path, result)
    if root is None
        return
    current_path.append(root.val)
    if sum(current_path) == target_sum
        result.append(list(current_path))
```

```
    find_paths_with_sum(root.left, target_sum, current_path,
result)
    find_paths_with_sum(root.right, target_sum, current_path,
result)
    current_path.pop()

find_all_paths_from_node(node, target_sum, result)
    if node is None
        return
    find_paths_with_sum(node, target_sum, [], result)
    find_all_paths_from_node(node.left, target_sum, result)
    find_all_paths_from_node(node.right, target_sum, result)

find_all_paths_with_sum(root, target_sum)
    if root is None
        return []
    result = []
    find_all_paths_from_node(root, target_sum, result)
    print(result)
    return result

write_tree_and_paths_to_file(root, paths, filename)
    open filename for writing as f
        write_tree_to_file(root, f)
        f.write("\nPaths with sum:\n")
        for path in paths
            f.write(' '.join(map(str, path)) + '\n')

write_tree_to_file(root, filey)
    if root is None
        filey.write("0 ")
        return
    filey.write(str(root.val) + " ")
    write_tree_to_file(root.left, filey)
```

```
    write_tree_to_file(root.right, filey)

cycle(input_file, output_file, target_sum)
    root = transformToBinarySearchTree(input_file)
    monotonic_paths = find_all_paths_with_sum(root,
target_sum)
    write_tree_and_paths_to_file(root, monotonic_paths,
output_file)
```

Код

Завдання 1:

```python
class TreeNode:
    def __init__(self, val):
        self.val = val
        self.left = None
        self.right = None


def construct_tree_from_preorder(preorder):
    if not preorder:
        return None

    val = preorder.pop(0)
    if val == 0:
        return None

    node = TreeNode(val)
    node.left = construct_tree_from_preorder(preorder)
    node.right = construct_tree_from_preorder(preorder)

    return node


def inorder_traversal(root):
    if root is None:
        return []
    return inorder_traversal(root.left) + [root.val] + inorder_traversal(root.right)


def rebuild_binary_search_tree(root, values):
    if root is None:
        return

    # Використовуємо ітератор для значень, щоб не обрізати список values
    val_iter = iter(values)

    # Рекурсивно перебудовуємо дерево пошуку, вставляючи значення зі списку values
    rebuild_binary_search_tree_helper(root, val_iter)


def rebuild_binary_search_tree_helper(node, val_iter):
    if node is None:
        return

    # Перебудова лівого піддерева
    rebuild_binary_search_tree_helper(node.left, val_iter)

    # Оновлення значення поточного вузла
    node.val = next(val_iter)
```

```
    node.val = next(val_iter)

    # Перебудова правого піддерева
    rebuild_binary_search_tree_helper(node.right, val_iter)


def read_preorder_from_file(filename):
    with open(filename, 'r') as f:
        preorder = list(map(int, f.read().split()))
    return preorder


# Основна частина програми

def transformToBinarySearchTree(input_file: str):
    # Зчитуємо preorder з файлу
    preorder = read_preorder_from_file(input_file)

    # Побудова бінарного дерева з preorder
    root = construct_tree_from_preorder(preorder)

    # Побудова inorder з урахуванням нулів
    inorder_values = sorted(
        set(val for val in inorder_traversal(root) if val != 0))

    # Перебудова дерева пошуку
    rebuild_binary_search_tree(root, inorder_values)

    return root
```

Завдання 2:

```python
import sys
import os
from task1 import transformToBinarySearchTree


def find_paths_with_sum(root, target_sum, current_path, result):
    if root is None:
        return

    # Додаємо поточний вузол до поточного шляху
    current_path.append(root.val)

    # Якщо значення поточного шляху дорівнює цільовій сумі, додаємо його до результату
    if sum(current_path) == target_sum:
        result.append(list(current_path))

    # Рекурсивно шукаємо вліво та вправо
    find_paths_with_sum(root.left, target_sum, current_path, result)
    find_paths_with_sum(root.right, target_sum, current_path, result)

    # Після проходження вузла видаляємо його з поточного шляху
    current_path.pop()


def find_all_paths_from_node(node, target_sum, result):
    if node is None:
        return

    # Запускаємо пошук шляхів з поточного вузла
    find_paths_with_sum(node, target_sum, [], result)

    # Рекурсивно запускаємо пошук для лівого та правого піддерева
    find_all_paths_from_node(node.left, target_sum, result)
    find_all_paths_from_node(node.right, target_sum, result)


def find_all_paths_with_sum(root, target_sum):
    if root is None:
        return []

    result = []
    find_all_paths_from_node(root, target_sum, result)
    print(result)
    return result


def write_tree_and_paths_to_file(root, paths, filename):
    with open(filename, 'w') as f:
```

```
        # Запис бінарного дерева пошуку
        write_tree_to_file(root, f)

        # Роздільник між бінарним деревом та шляхами
        f.write("\nPaths with sum:\n")

        # Запис шляхів у файл
        for path in paths:
            f.write(' '.join(map(str, path)) + '\n')


def write_tree_to_file(root, filey):
    if root is None:
        filey.write("0 ")
        return
    filey.write(str(root.val) + " ")
    write_tree_to_file(root.left, filey)
    write_tree_to_file(root.right, filey)


# Основна частина програми
directory = "D:\\КПІ\\ТА. Лабораторні\\ТА.Lab6"
inputDir = "\\examples"
outputDir = "\\results"


def cycle(input_file: str, output_file: str, target_sum: int):
    root = transformToBinarySearchTree(input_file)

    # Знаходимо всі монотонні шляхи з сумою target_sum
    monotonic_paths = find_all_paths_with_sum(root, target_sum)

    # Запис бінарного дерева та шляхів у файл
    write_tree_and_paths_to_file(root, monotonic_paths, output_file)


if __name__ == "__main__":
    cycle(directory+inputDir+"\\input_1000c.txt",
          directory+outputDir+"\\output_1000c.txt", 513)
else:
    # Основна частина програми
    if len(sys.argv) != 4:
        print("Потрібно ввести ім'я вхідного файлу, ім'я вихідного файлу та число S як аргументи командного рядка.")
        sys.exit(1)
```

```
    input_file = sys.argv[1]
    output_file = sys.argv[2]
    target_sum = int(sys.argv[3])

    # Перевірка наявності вхідного файлу
    if not os.path.exists(input_file):
        print("Вхідний файл не знайдено.")
        sys.exit(1)

    cycle(input_file, output_file, target_sum)
```

Результати роботи програми

results > output_10a.txt

```
1   6 3 2 1 0 0 0 4 0 5 0 0 8 7 0 0 10 9 0 0 0
2   Paths with sum:
3   6 3
4   4 5
5   9
```

results > output_10b.txt

```
5 2 1 0 0 3 0 4 0 0 8 7 6 0 0 0 10 9 0 0 0
Paths with sum:
5 2
3 4
7
```

results > output_100b.txt

```
59 29 14 5 2 1 0 0 3 0 4 0 0 9 8 7 6 0 0 0 0 12 11 10 0 0 0 13 0 0 21 15 0 17 16 0 0 1
Paths with sum:
29 14 5 9 8 7 6
24 26 28
78
```

results > output_1000c.txt

```
493 253 130 69 33 16 12 6 3 2 1 0 0 0 5 4 0 0 0 8 7 0 0 11 9 0 10 0 0 0 14 1
Paths with sum:
253 130 69 33 16 12
130 69 99 112 103
69 99 81 91 87 86
91 87 86 82 84 83
258 255
257 256
513
```

Висновки

В ході виконання лабораторної роботи ми розглянули задачу визначення послідовності медіан для заданого вхідного масиву. Під час додавання нового елементу до масиву розглядалась відповідність інваріанту, що забезпечує рівність кількостей елементів в обох пірамідах. При порушенні цього інваріанту виконувалися операції для його відновлення.

Ми визначили, що використання двох пірамід для зберігання меншої та більшої половин масиву дозволяє досягти ефективності у визначенні медіани. Вставка нового елементу та відновлення інваріанту відбуваються за час O(log(i)), що робить алгоритм практично застосовним для великих масивів.

Оскільки операції з пірамідами виконуються за логарифмічний час, загальний час визначення медіани для всього масиву складає O(n log n). Це

показник набагато кращий, ніж O(n^2), який можна було б отримати при використанні простих методів сортування.

Загалом, розглянутий алгоритм є ефективним та простим у реалізації розв'язком задачі визначення медіани для послідовно надходячих елементів масиву. Його можна успішно використовувати для різних практичних задач, де потрібно визначити медіану масиву елементів зі складністю O(log(i)) на кожній ітерації.