МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное образовательное учреждение высшего образования

САНКТ – ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ

КАФЕДРА № 42

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| старший преподаватель |  |  |  | С. Ю. Гуков |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №8  Обход двоичного дерева |
|  |
| по курсу: АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ гр. № | 4321 |  |  |  | К.А. Лебедев |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург, 2024

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc178416272)

[2 Задание 4](#_Toc178416273)

[3 Ход разработки 5](#_Toc178416274)

[4 Исходный код программы 6](#_Toc178416275)

[5 Результаты работы программы 8](#_Toc178416276)

[6 Вывод 9](#_Toc178416277)

**1 Цель работы**

Построить in-order, pre-order и post-order обходы данного двоичного дерева.

**2 Задание**

Программа может быть написана на любом языке программирования. Вход. Двоичное дерево. Выход. Все его вершины в трёх разных порядках: in-order, pre-order и post-order. In-order обход соответствует следующей рекурсивной процедуре, получающей на вход корень v текущего поддерева: произвести рекурсивный вызов для v.left, напечатать v.key, произвести рекурсивный вызов для v.right. Pre-order обход: напечатать v.key, произвести рекурсивный вызов для v.left, произвести рекурсивный вызов для v.right. Post-order: произвести рекурсивный вызов для v.left, произвести рекурсивный вызов для v.right, напечатать v.key. Формат входа.

Первая строка содержит число вершин n. Вершины дерева пронумерованы числами от 0 до n−1. Вершина 0 является корнем. Каждая из следующих n строк содержит информацию о вершинах 0, 1, ..., n−1: i-я строка задаёт числа keyi, lefti и righti, где keyi – ключ вершины i, lefti – индекс левого сына вершины i, а righti – индекс правого сына вершины i. Если у вершины i нет одного или обоих сыновей, соответствующее значение равно −1. Формат выхода. Три строки: in-order, pre-order и post-order обходы.

Ограничения. 1 ≤ n ≤ 105 ; 0 ≤ keyi ≤ 109 ; −1 ≤ lefti, righti ≤ n−1. Гарантируется, что вход задаёт корректное двоичное дерево: в частности, если lefti != −1 и righti != −1, то lefti != righti; никакая вершина не является сыном двух вершин; каждая вершина является потомком корня.**3 Ход разработки**

Были реализованы класс задачи и функция для вычисления высоты дерева, написана утилита для прочтения чисел из командной строки на языке TypeScript, интегрирован компилятор в JavaScript, который сразу собирает TypeScript проект без предварительной компиляции.

**4 Исходный код программы**

import TreeNode from "./lib/TreeNode";

import { readlineInterface } from "./utils/readline";

const nodes: TreeNode[] = [];

let n: number;

readlineInterface.on('line', (line: string) => {

if (n === undefined) {

n = parseInt(line); *// Читаем количество узлов*

} else {

const [key, left, right] = line.split(' ').map(Number);

nodes.push(new TreeNode(key, left, right));

if (nodes.length === n) {

*// Когда все узлы прочитаны, выполняем обходы*

const tree = new BinaryTree(nodes);

const inOrderResult: number[] = [];

const preOrderResult: number[] = [];

const postOrderResult: number[] = [];

tree.inOrderTraversal(0, inOrderResult);

tree.preOrderTraversal(0, preOrderResult);

tree.postOrderTraversal(0, postOrderResult);

*// Вывод результатов*

console.log('In-order:', inOrderResult.join(' '));

console.log('Pre-order:', preOrderResult.join(' '));

console.log('Post-order:', postOrderResult.join(' '));

readlineInterface.close(); *// Закрываем интерфейс после завершения*

}

}

});

**class TreeNode {**

**key: number;**

**left: number;**

**right: number;**

**constructor(key: number, left: number, right: number) {**

**this.key = key;**

**this.left = left;**

**this.right = right;**

**}**

**}**

**export default TreeNode;**

**import TreeNode from "./TreeNode";**

**class BinaryTree {**

**nodes: TreeNode[];**

**constructor(nodes: TreeNode[]) {**

**this.nodes = nodes;**

**}**

**inOrderTraversal(nodeIndex: number, result: number[]): void {**

**if (nodeIndex === -1) return;**

**this.inOrderTraversal(this.nodes[nodeIndex].left, result);**

**result.push(this.nodes[nodeIndex].key);**

**this.inOrderTraversal(this.nodes[nodeIndex].right, result);**

**}**

**preOrderTraversal(nodeIndex: number, result: number[]): void {**

**if (nodeIndex === -1) return;**

**result.push(this.nodes[nodeIndex].key);**

**this.preOrderTraversal(this.nodes[nodeIndex].left, result);**

**this.preOrderTraversal(this.nodes[nodeIndex].right, result);**

**}**

**postOrderTraversal(nodeIndex: number, result: number[]): void {**

**if (nodeIndex === -1) return;**

**this.postOrderTraversal(this.nodes[nodeIndex].left, result);**

**this.postOrderTraversal(this.nodes[nodeIndex].right, result);**

**result.push(this.nodes[nodeIndex].key);**

**}**

**}**

**export default BinaryTree;**

**5 Результаты работы программы**

Результат работы программы включает в себя вывод очереди команд

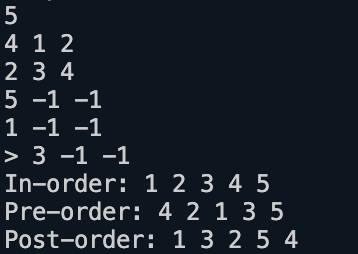


Рисунок 1 – тест программы №1

**6 Вывод**

Алгоритмы и структуры данных играют ключевую роль в эффективном решении задач в программировании. Они позволяют оптимизировать процессы и ускорить выполнение операций при правильном использовании.

Использование рекурсии для вычисления высоты дерева демонстрирует важный аспект работы с деревьями, так как многие задачи, связанные с деревьями, можно решать с помощью рекурсивных подходов.

Основная цель программы — реализовать обход дерева различными способами. Дерево заданно через массив родителей, где каждый элемент массива указывает на родителя соответствующего узла. Деревья — это структуры данных, состоящие из узлов, связанных между собой, и они широко используются в различных алгоритмах, таких как обход, сортировка и хранение иерархических данных.