МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ОБРАЗОВАНИЯ
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»

Институт № 3 «Системы управления, информатика и электроэнергетика»

Кафедра 304 «Вычислительные машины, системы и сети»

Лабороторная работа №4

по дисциплине «Алгоритмы и обработка данных»

на тему «Алгоритмы сортировки данных»

Выполнили

студенты группы М3О-225БВ-24

Егоров А.В
Федоров А.И.

Принял

Москва
2025

Задача

Лабораторная работа «Бинарные деревья поиска»

Задание. Этап 1.

1. Случайным образом сгенерировать массив размерностью 20-25 элементов, повторные значения не допустимы.
2. Реализовать функции вставки, поиска, удаления узла (три случая), вывода дерева на экран, нахождения высоты дерева и количества узлов.
   Две функции обхода дерева: для получения отсортированной последовательности ключей, для удаления дерева.
3. Реализовать дополнительно функцию в соответствии с вариантом из таблицы: *D* – диапазон изменения значений ключей.

№	*D*	Функция
1	[100; 200]	Подсчет суммы длин путей от корня до каждого из узлов, содержащих четные числа
2	[0; 100]	Определить число узлов в левом и правом поддеревьях
3	[-50; 50]	Подсчет суммы четных отрицательных чисел в узлах, поддеревья которых содержат не более 4 узлов
4	[0; 100]	Определить сумму четных чисел в листьях
5	[1;190]	Определить два минимальных элемента, два максимальных элемента.
6	[-100; 100]	Определить, каких чисел больше - положительных или отрицательных.
7	[-50; 50]	Определить сумму элементов, кратных 5 отдельно в левом и правом поддеревьях.
8	[1; 190]	Подсчет количества нечетных чисел в узлах, имеющих ровно два поддерева

Задание. Этап 2.

Сформировать случайную последовательность из 300 ключей (тип int, повтор значений запрещен). Отсортировать по возрастанию и записать результат в другой массив чисел.

Для случайной последовательности ключей:

1. Используя функции вставки ключей и функции подсчета высоты дерева построить два бинарных дерева: «обычное» и рандомизированное.
2. Построить зависимости высоты каждого дерева от количества ключей. Диапазон количества ключей от 25 до 300 (шаг изменения количества 25). Все результаты оформить в виде таблицы.

Для отсортированной последовательности ключей:

повторить все пункты задания как для случайной последовательности.

Отчет по лабораторной работе должен содержать:

- задание;
- программы;
- результаты работы;
- выводы.

Схемы алгоритмов делать не нужно!

Код

```
#include <iostream>
#include <iomanip>  // Для таблицы (setw)
#include <cstdlib>  // rand, srand
#include <ctime>    // time
using namespace std;
// ============================================================
// СТРУКТУРЫ ДАННЫХ
// ============================================================
// Узел обычного Бинарного Дерева Поиска (BST)
struct Node {
    int key;
    Node* left;
    Node* right;
    Node(int k) : key(k), left(NULL), right(NULL) {}
};
// Узел Рандомизированного дерева (Treap)
struct TreapNode {
    int key;
    int priority;
    TreapNode* left;
    TreapNode* right;
    TreapNode(int k) : key(k), left(NULL), right(NULL) {
        priority = rand();
    }
};

// ============================================================
// ВСПОМОГАТЕЛЬНЫЕ ИНСТРУМЕНТЫ (Без STL)
// ============================================================
// Проверка на уникальность
bool isUnique(int* arr, int n, int val) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == val) return false;
    }
    return true;
}
// Сортировка пузырьком
void bubbleSort(int* arr, int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
```

```
                arr[j + 1] = temp;
            }
        }
    }
}

// Генерация уникального массива
void generateUniqueArray(int* arr, int n, int minVal, int maxVal) {
    int count = 0;
    while (count < n) {
        int val = minVal + rand() % (maxVal - minVal + 1);
        if (isUnique(arr, count, val)) {
            arr[count] = val;
            count++;
        }
    }
}

// ============================================================
// ОСНОВНЫЕ ФУНКЦИИ BST
// ============================================================

// Вставка
void insert(Node*& root, int key) {
    if (root == NULL) {
        root = new Node(key);
        return;
    }
    if (key < root->key)
        insert(root->left, key);
    else if (key > root->key)
        insert(root->right, key);
}
// Поиск
Node* search(Node* root, int key) {
    if (root == NULL || root->key == key) return root;
    if (key < root->key) return search(root->left, key);
    return search(root->right, key);
}
// Поиск минимума
Node* findMin(Node* root) {
    while (root->left != NULL) root = root->left;
    return root;
}
// Удаление узла
```

```
void deleteNode(Node*& root, int key) {
    if (root == NULL) return;
    if (key < root->key) {
        deleteNode(root->left, key);
    } else if (key > root->key) {
        deleteNode(root->right, key);
    } else {
        if (root->left == NULL) {
            Node* temp = root->right;
            delete root;
            root = temp;
        } else if (root->right == NULL) {
            Node* temp = root->left;
            delete root;
            root = temp;
        } else {
            Node* temp = findMin(root->right);
            root->key = temp->key;
            deleteNode(root->right, temp->key);
        }
    }
}
// Высота
int getHeight(Node* root) {
    if (root == NULL) return 0;
    int hL = getHeight(root->left);
    int hR = getHeight(root->right);
    return 1 + (hL > hR ? hL : hR);
}
// Количество узлов
int getCount(Node* root) {
    if (root == NULL) return 0;
    return 1 + getCount(root->left) + getCount(root->right);
}
// --- НОВЫЙ ВЫВОД (ВЕРТИКАЛЬНЫЙ) ---
// Корень сверху, дети снизу с отступами
void printTree(Node* root, int level = 0, const char* type = "ROOT") {
    if (root == NULL) return;
    // Делаем отступ
    for (int i = 0; i < level; i++) cout << "    ";
    // Печатаем узел
    if (level == 0)
        cout << "[" << type << "] " << root->key << endl;
    else
        cout << "|__" << type << ": " << root->key << endl;
```

```
    // Рекурсивно выводим детей (сначала левого, потом правого)
    printTree(root->left, level + 1, "L");
    printTree(root->right, level + 1, "R");
}
// Сортированный вывод
void printSorted(Node* root) {
    if (root == NULL) return;
    printSorted(root->left);
    cout << root->key << " ";
    printSorted(root->right);
}
// Очистка памяти
void deleteTree(Node*& root) {
    if (root == NULL) return;
    deleteTree(root->left);
    deleteTree(root->right);
    delete root;
    root = NULL;
}
// ВАРИАНТ 4: Сумма четных в листьях
void sumEvenLeaves(Node* root, int& sum) {
    if (root == NULL) return;
    if (root->left == NULL && root->right == NULL) {
        if (root->key % 2 == 0) sum += root->key;
    }
    sumEvenLeaves(root->left, sum);
    sumEvenLeaves(root->right, sum);
}
// ============================================================
// ФУНКЦИИ TREAP
// ============================================================

void split(TreapNode* t, int key, TreapNode*& l, TreapNode*& r) {
    if (!t) l = r = NULL;
    else if (key < t->key) split(t->left, key, l, t->left), r = t;
    else split(t->right, key, t->right, r), l = t;
}
void merge(TreapNode*& t, TreapNode* l, TreapNode* r) {
    if (!l || !r) t = l ? l : r;
    else if (l->priority > r->priority) merge(l->right, l->right, r), t = l;
    else merge(r->left, l, r->left), t = r;
}
void insertTreap(TreapNode*& t, TreapNode* it) {
    if (!t) t = it;
```

```
    else if (it->priority > t->priority) split(t, it->key, it->left, it->right), t
= it;
    else insertTreap(it->key < t->key ? t->left : t->right, it);
}
int getTreapHeight(TreapNode* t) {
    if (!t) return 0;
    int hL = getTreapHeight(t->left);
    int hR = getTreapHeight(t->right);
    return 1 + (hL > hR ? hL : hR);
}
void deleteTreap(TreapNode*& t) {
    if (!t) return;
    deleteTreap(t->left);
    deleteTreap(t->right);
    delete t;
    t = NULL;
}
// ============================================================
// MAIN
// ============================================================
int main() {
    setlocale(LC_ALL, "Russian");
    srand(time(0));
    // --- ЭТАП 1 ---
    cout << "=== ЭТАП 1: Проверка BST ===" << endl;
    int n1 = 20;
    int* arr1 = new int[n1];
    generateUniqueArray(arr1, n1, 0, 100);
    Node* root = NULL;
    cout << "Ключи: ";
    for (int i = 0; i < n1; i++) {
        cout << arr1[i] << " ";
        insert(root, arr1[i]);
    }
    cout << endl << endl;
    cout << "Структура дерева:" << endl;
    printTree(root);
    cout << "--------------------------------" << endl;

    cout << "Высота: " << getHeight(root) << endl;
    cout << "Узлов: " << getCount(root) << endl;
    cout << "Сортировка: ";
    printSorted(root);
    cout << endl;
    int sum = 0;
```

```
sumEvenLeaves(root, sum);
cout << "[Вар. 4] Сумма четных листьев: " << sum << endl;
int keyDel = arr1[0];
cout << "\nУдаляем узел " << keyDel << "..." << endl;
deleteNode(root, keyDel);
cout << "Структура после удаления:" << endl;
printTree(root);
deleteTree(root);
delete[] arr1;
cout << endl;
// --- ЭТАП 2 ---
cout << "=== ЭТАП 2: Сравнение высот ===" << endl;
cout << left << setw(8) << "N"
     << " | " << setw(15) << "Rand BST"
     << " | " << setw(15) << "Rand Treap"
     << " | " << setw(15) << "Sort BST"
     << " | " << setw(15) << "Sort Treap" << endl;
cout << string(80, '-') << endl;

for (int n = 25; n <= 300; n += 25) {
    int* randKeys = new int[n];
    int* sortedKeys = new int[n];
    generateUniqueArray(randKeys, n, 0, 10000);
    for(int i=0; i<n; i++) sortedKeys[i] = randKeys[i];
    bubbleSort(sortedKeys, n);
    Node* bstRand = NULL;
    TreapNode* treapRand = NULL;
    for (int i = 0; i < n; i++) {
        insert(bstRand, randKeys[i]);
        insertTreap(treapRand, new TreapNode(randKeys[i]));
    }
    int h_bst_r = getHeight(bstRand);
    int h_treap_r = getTreapHeight(treapRand);
    deleteTree(bstRand);
    deleteTreap(treapRand);
    Node* bstSort = NULL;
    TreapNode* treapSort = NULL;
    for (int i = 0; i < n; i++) {
        insert(bstSort, sortedKeys[i]);
        insertTreap(treapSort, new TreapNode(sortedKeys[i]));
    }
    int h_bst_s = getHeight(bstSort);
    int h_treap_s = getTreapHeight(treapSort);
    deleteTree(bstSort);
    deleteTreap(treapSort);
```

```
        cout << left << setw(8) << n
             << " | " << setw(15) << h_bst_r
             << " | " << setw(15) << h_treap_r
             << " | " << setw(15) << h_bst_s
             << " | " << setw(15) << h_treap_s << endl;
        delete[] randKeys;
        delete[] sortedKeys;
    }
    return 0;
}
```

Тесты программы

```
=== ЭТАП 1: Проверка BST ===
Ключи: 48 18 1 54 8 7 27 91 79 81 85 80 55 15 61 20 30 23 16 90

Структура дерева:
[ROOT] 48
    |__L: 18
        |__L: 1
            |__R: 8
                |__L: 7
                |__R: 15
                    |__R: 16
        |__R: 27
            |__L: 20
                |__R: 23
            |__R: 30
    |__R: 54
        |__R: 91
            |__L: 79
                |__L: 55
                    |__R: 61
                |__R: 81
                    |__L: 80
                    |__R: 85
                        |__R: 90
--------------------------------
Высота: 7
Узлов: 20
Сортировка: 1 7 8 15 16 18 20 23 27 30 48 54 55 61 79 80 81 85 90 91
[Вар. 4] Сумма четных листьев: 216
```

```
Удаляем узел 48...
Структура после удаления:
[ROOT] 54
    |__L: 18
        |__L: 1
            |__R: 8
                |__L: 7
                |__R: 15
                    |__R: 16
        |__R: 27
            |__L: 20
                |__R: 23
            |__R: 30
    |__R: 91
        |__L: 79
            |__L: 55
                |__R: 61
            |__R: 81
                |__L: 80
                |__R: 85
                    |__R: 90

=== ЭТАП 2: Сравнение высот ===
N        | Rand BST        | Rand Treap      | Sort BST        | Sort Treap
-------------------------------------------------------------------------------
25       | 8               | 6               | 25              | 7
50       | 12              | 12              | 50              | 11
75       | 11              | 12              | 75              | 14
100      | 12              | 13              | 100             | 14
125      | 14              | 14              | 125             | 13
150      | 16              | 14              | 150             | 14
175      | 16              | 15              | 175             | 14
200      | 16              | 17              | 200             | 16
225      | 15              | 17              | 225             | 16
250      | 21              | 19              | 250             | 15
275      | 18              | 19              | 275             | 16
300      | 17              | 18              | 300             | 17
```

Вывод

В ходе лабораторной работы были изучены структура и принципы функционирования бинарных деревьев поиска (BST). На первом этапе были успешно реализованы базовые алгоритмы управления деревом, включая вставку, поиск, три случая удаления узлов и функции обхода. Сравнительный анализ во второй части работы показал, что высота обычного дерева поиска сильно зависит от порядка входных данных: при использовании отсортированных последовательностей дерево вырождается в список. В то же время рандомизированное дерево поиска продемонстрировало стабильную логарифмическую зависимость высоты от количества узлов независимо от структуры входного массива. Таким образом, на практике было подтверждено преимущество рандомизированных структур для обеспечения эффективности операций в худших сценариях.