Министерство образования и науки РФ

Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа «Компьютерных технологий и информационных систем»

ОТЧЕТ

по дисциплине ««Теория и технология программирования»

**Лабораторная работа № 7**

**Выполнил:**

Cтудент гр. 5130902/40002 Г.Ю. Рюмин

**Проверил**

Ст. преподаватель А.М. Журавская

Санкт-Петербург

2025 г.

**1. Цель работы**

Цели задания:

* освоить методы поиска;
* спроектировать и использовать созданное дерево поиска.

**2. Задание**

Составить программу для поиска в линейном (одномерном или многомерном) массиве данных, представленном в виде дерева. В работе использовать следующие виды поиска:

* последовательный;
* бинарный.

Создать дерево (деревья (массив указателей на корни деревьев в случае с многомерной задачей)) поиска (в соответствии с индивидуальным заданием) и заполнить дерево данными из линейного массива данных.

Осуществить поиск данных по дереву. Провести вывод на экран (или файл, по выбору) следующих данных:

* исходного массива данных (матрица или текст в зависимости от варианта);
* значений дерева разными способами (прямой, обратный, центральный обход);
* результатов последовательного и бинарного поиска.

В случае невозможности (из условий индивидуального задания) поиска бинарным методом, провести поиск бинарным методом одной буквы (цифры). Придумать тестовые примеры, для которых были бы эффективными каждый из методов.

|  |  |  |
| --- | --- | --- |
| 27. | Произвести поиск элемента (последовательности элементов) в матрице, заданной случайным образом, отсортированной по контуру по часовой стрелке. Начальный элемент будет находиться на позиции (0,0), конечный в центре матрицы. | Бинарное дерево |

## **Код программы**

#include <iostream>

#include <vector>

#include <algorithm>

#include <ctime>

#include <locale>

#include <limits>

using namespace std;

struct TreeNode {

int value;

TreeNode\* left;

TreeNode\* right;

TreeNode(int val) : value(val), left(nullptr), right(nullptr) {}

};

void getValidMatrixSize(int& input, const string& prompt) {

while (true) {

cout << prompt;

cin >> input;

if (cin.fail() || cin.peek() != '\n') {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Ошибка: введите целое число от 1 до 100" << endl;

}

else if (input < 1 || input > 100) {

cout << "Ошибка: число должно быть от 1 до 100" << endl;

}

else {

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

break;

}

}

}

void sortMatrixAscending(vector<vector<int>>& matrix) {

vector<int> allElements;

for (auto& row : matrix)

for (int val : row)

allElements.push\_back(val);

sort(allElements.begin(), allElements.end());

size\_t index = 0;

for (auto& row : matrix)

for (int& val : row)

val = allElements[index++];

}

TreeNode\* insert(TreeNode\* root, int value, vector<int>& path) {

if (!root) {

path.push\_back(value);

return new TreeNode(value);

}

path.push\_back(root->value);

if (value < root->value)

root->left = insert(root->left, value, path);

else

root->right = insert(root->right, value, path);

return root;

}

void preOrder(TreeNode\* root) {

if (!root) return;

cout << root->value << " ";

preOrder(root->left);

preOrder(root->right);

}

void inOrder(TreeNode\* root) {

if (!root) return;

inOrder(root->left);

cout << root->value << " ";

inOrder(root->right);

}

void postOrder(TreeNode\* root) {

if (!root) return;

postOrder(root->left);

postOrder(root->right);

cout << root->value << " ";

}

bool sequentialSearch(TreeNode\* root, int target, vector<int>& path) {

if (!root) return false;

path.push\_back(root->value);

if (root->value == target) return true;

return sequentialSearch(root->left, target, path) ||

sequentialSearch(root->right, target, path);

}

bool binarySearch(TreeNode\* root, int target, vector<int>& path) {

if (!root) return false;

path.push\_back(root->value);

if (root->value == target) return true;

if (target < root->value)

return binarySearch(root->left, target, path);

return binarySearch(root->right, target, path);

}

vector<vector<int>> generateMatrix(int M, int N) {

vector<vector<int>> matrix(M, vector<int>(N));

for (int i = 0; i < M; i++)

for (int j = 0; j < N; j++)

matrix[i][j] = rand() % 90 + 10;

return matrix;

}

void printMatrix(const vector<vector<int>>& matrix) {

for (const auto& row : matrix) {

for (int val : row) cout << val << " ";

cout << endl;

}

}

void deleteTree(TreeNode\* root) {

if (!root) return;

deleteTree(root->left);

deleteTree(root->right);

delete root;

}

int main() {

setlocale(LC\_ALL, "Russian");

srand(time(0));

int M, N;

getValidMatrixSize(M, "Введите количество строк (1-100): ");

getValidMatrixSize(N, "Введите количество столбцов (1-100): ");

vector<vector<int>> matrix = generateMatrix(M, N);

cout << endl << "Исходная матрица:" << endl;

printMatrix(matrix);

sortMatrixAscending(matrix);

cout << endl << "Матрица после сортировки:" << endl;

printMatrix(matrix);

TreeNode\* root = nullptr;

vector<int> insertPath;

for (auto& row : matrix)

for (int val : row)

root = insert(root, val, insertPath);

cout << endl << "Обходы дерева:";

cout << endl << "1. Прямой обход: ";

preOrder(root);

cout << endl << "2. Центральный обход: ";

inOrder(root);

cout << endl << "3. Обратный обход: ";

postOrder(root);

cout << endl;

int target;

while (true) {

cout << endl << "Введите элемент для поиска: ";

cin >> target;

if (cin.fail() || cin.peek() != '\n') {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Ошибка ввода!" << endl;

}

else {

break;

}

}

vector<int> seqPath, binPath;

bool seqFound = sequentialSearch(root, target, seqPath);

bool binFound = binarySearch(root, target, binPath);

cout << endl << "Результаты поиска:";

cout << endl << "Последовательный поиск (" << (seqFound ? "+" : "-") << "): ";

for (size\_t i = 0; i < seqPath.size(); ++i) {

cout << seqPath[i];

if (i != seqPath.size() - 1) cout << " -> ";

}

cout << endl << "Бинарный поиск (" << (binFound ? "+" : "-") << "): ";

for (size\_t i = 0; i < binPath.size(); ++i) {

cout << binPath[i];

if (i != binPath.size() - 1) cout << " -> ";

}

deleteTree(root);

return 0;

}

**Пример работы программы**

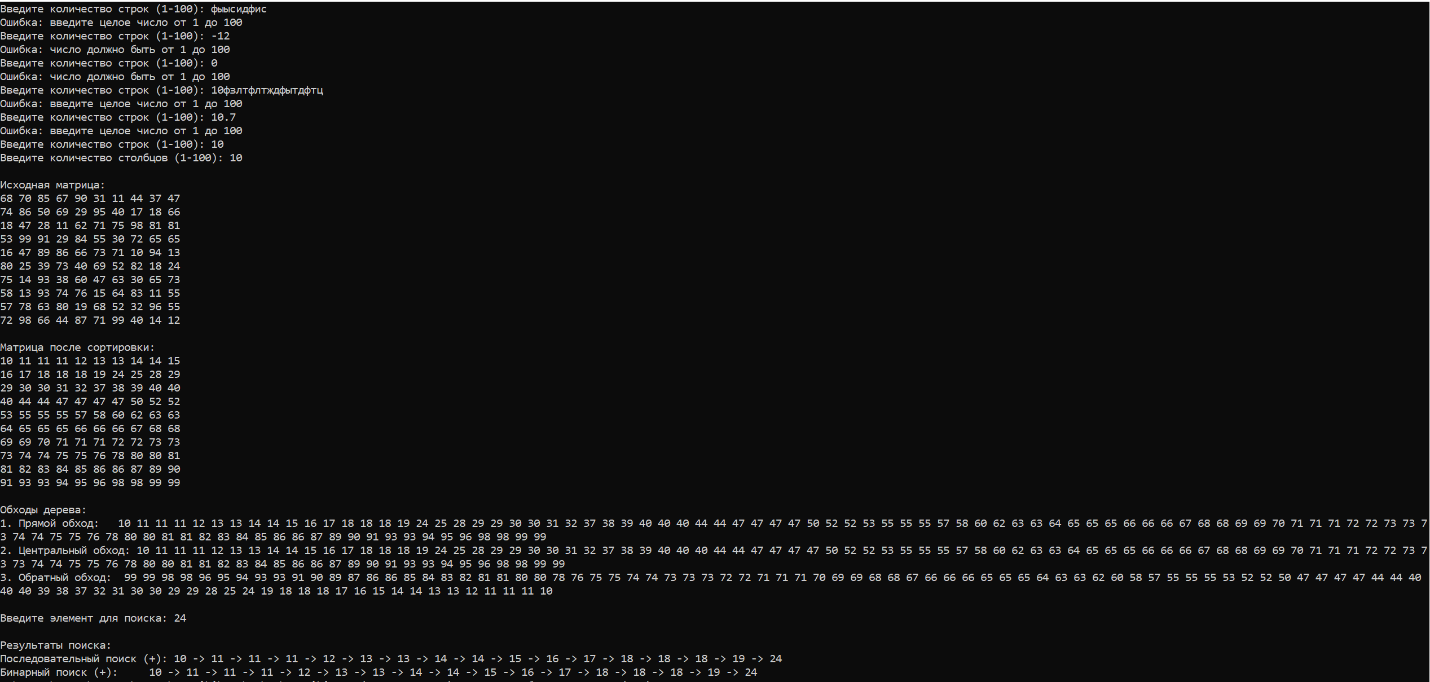


Рисунок 1 – Пример работы программы с некорректным, корректным вводом, выводом обхода деревьев и пути поиска элемента

## **Вывод**

В ходе лабораторной работы были освоены методы поиска данных: последовательный и бинарный. Было создано бинарное дерево поиска, заполненное данными из случайно сгенерированной матрицы, отсортированной по контуру по часовой стрелке. Реализованы прямой, обратный и центральный обходы дерева, а также проведен поиск элемента с использованием последовательного и бинарного методов. Результаты поиска и обходов дерева были выведены на экран. Работа позволила закрепить навыки работы с деревьями поиска, а также сравнить эффективность различных методов поиска в зависимости от структуры данных.