Бюджетное учреждение высшего образования Ханты-Мансийского автономного округа – Югры «СУРГУТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Политехнический институт

Кафедра автоматики и компьютерных систем

Отчет

по лабораторной работе № 5

по дисциплине «Алгоритмы и структуры данных»

Проверила: старший преподаватель кафедры АиКС,

Карамышев. Э. Р

Выполнил: студент группы 605-31,

Хайитов Ш. Д.

Сургут 2025

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc194868567)

[ОСНОВНАЯ ЧАСТЬ 4](#_Toc194868568)

[ЗАКЛЮЧЕНИЕ 10](#_Toc194868569)

# **ВВЕДЕНИЕ**

**Цель:** изучить базовые алгоритмы работы с деревьями: построение, обход, поиска элемента, удаление элемента, подсчет количества узлов, нахождение высоты дерева, исследовать свойства деревьев, закрепить навыки структурного программирования.

**Задачи:**

1. Разработать функции вставки, поиска и удаления узла, обход дерева, вставка в корень, вывода дерева на экран, нахождения высоты дерева и количества узлов, а также функции вставки, строящую рандомизированное дерево.
2. Построить зависимость высоты дерева от количества ключей.
3. Реализовать подсчет суммы длин путей от корня до каждого из узлов, содержащих четные числа.
4. Составить отчет, в котором привести графики зависимостей.

Вариант данный преподавателем представлен на рисунке 1.

****

Рисунок 1 – Вариант 10

# **ОСНОВНАЯ ЧАСТЬ**

Высота обычного бинарного дерева от количества ключей. Данные измерения представлены в таблице 1.

Таблица 1 – Высота обычного бинарного дерева от количества ключей.

|  |  |  |
| --- | --- | --- |
| Кол-во узлов | Высота дерева | Гласные (внутренние узлы) |
| 10 | 5 | 2 |
| 20 | 8 | 3 |
| 50 | 13 | 4 |
| 100 | 15 | 11 |
| 500 | 25 | 83 |

По этим данным был составлен график, представленный на рисунке 2.

**A screenshot of a graph

AI-generated content may be incorrect.**

Рисунок 2 – Зависимость высоты обычного бинарного дерева, от количества

ключей

Высота рандомизированного дерева от количества ключей. Данные занесены в таблицу 2.

Таблица 2 – Высота рандомизированного дерева от количества ключей.

|  |  |  |
| --- | --- | --- |
| Кол-во | Высота дерева | Гласные (внутренние узлы) |
| 10 | 4 | 3 |
| 20 | 6 | 5 |
| 50 | 7 | 8 |
| 100 | 8 | 12 |
| 500 | 12 | 30 |

По этим данным был составлен график, представленный на рисунке 3.

A graph with blue and orange squares

AI-generated content may be incorrect.

Рисунок 3 – Зависимость высоты рандомизированного дерева от количества ключей.

**Листинг программы:**

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <cctype>

#include <cstdio>

using namespace std**;**

struct Node **{**

char Data**;**

Node**\*** Left**;**

Node**\*** Right**;**

**};**

**typedef** Node**\*** pNode**;**

pNode NewNode**(**char data**)** **{**

pNode node **=** **(**pNode**)**malloc**(sizeof(**Node**));**

node**->**Data **=** data**;**

node**->**Left **=** nullptr**;**

node**->**Right **=** nullptr**;**

**return** node**;**

**}**

void Insert**(**pNode**\*** proot**,** char data**)** **{**

pNode**&** root **=** **\***proot**;**

**if** **(!**root**)** **{**

root **=** NewNode**(**data**);**

**}** **else** **{**

**if** **(**data **<** root**->**Data**)** **{**

Insert**(&**root**->**Left**,** data**);**

**}** **else** **{**

Insert**(&**root**->**Right**,** data**);**

**}**

**}**

**}**

void InsertRoot**(**pNode**\*** proot**,** char data**)** **{**

pNode**&** root **=** **\***proot**;**

**if** **(!**root**)** **{**

root **=** NewNode**(**data**);**

**}** **else** **if** **(**rand**()** **%** 2**)** **{** // Рандомизация

InsertRoot**(&**root**->**Left**,** data**);**

**}** **else** **{**

InsertRoot**(&**root**->**Right**,** data**);**

**}**

**}**

int tree\_height**(**pNode root**)** **{**

**if** **(!**root**)** **return** 0**;**

int left **=** tree\_height**(**root**->**Left**);**

int right **=** tree\_height**(**root**->**Right**);**

**return** 1 **+** **(**left **>** right **?** left **:** right**);**

**}**

void FreeTree**(**pNode root**)** **{**

**if** **(**root**)** **{**

FreeTree**(**root**->**Left**);**

FreeTree**(**root**->**Right**);**

free**(**root**);**

**}**

**}**

char random\_tree\_keygen**()** **{**

**if** **(**rand**()** **%** 2 **==** 0**)** **{**

**return** 'a' **+** rand**()** **%** 26**;**

**}** **else** **{**

**return** 'A' **+** rand**()** **%** 26**;**

**}**

**}**

bool is\_vowel**(**char c**)** **{**

char lower **=** tolower**(**c**);**

**return** lower **==** 'a' **||** lower **==** 'e' **||** lower **==** 'i' **||** lower **==** 'o' **||** lower **==** 'u'**;**

**}**

int count\_internal\_vowels**(**pNode root**)** **{**

**if** **(!**root**)** **return** 0**;**

int count **=** 0**;**

**if** **(**root**->**Left **||** root**->**Right**)** **{**

**if** **(**is\_vowel**(**root**->**Data**))** **{**

count**++;**

**}**

**}**

count **+=** count\_internal\_vowels**(**root**->**Left**);**

count **+=** count\_internal\_vowels**(**root**->**Right**);**

**return** count**;**

**}**

int main**()** **{**

srand**(**time**(NULL));**

FILE**\*** output **=** fopen**(**"height\_vowel\_data.txt"**,** "w"**);**

system**(**"chcp 65001 > nul"**);**

**if** **(!**output**)** **{**

printf**(**"Ошибка создания файла!\n"**);**

**return** 1**;**

**}**

int sizes**[]** **=** **{**10**,** 20**,** 50**,** 100**,** 500**};**

int num\_sizes **=** **sizeof(**sizes**)** **/** **sizeof(**sizes**[**0**]);**

fprintf**(**output**,** "Количество узлов | Высота обычного | Высота рандомизированного | Гласные (обычное дерево)\n"**);**

fprintf**(**output**,** "---------------------------------------------------------------------------------------\n"**);**

**for** **(**int i **=** 0**;** i **<** num\_sizes**;** **++**i**)** **{**

int current\_size **=** sizes**[**i**];**

pNode normal\_tree **=** nullptr**;**

pNode rand\_tree **=** nullptr**;**

**for** **(**int j **=** 0**;** j **<** current\_size**;** j**++)** **{**

char key\_rand **=** random\_tree\_keygen**();**

char key\_norm **=** random\_tree\_keygen**();**

Insert**(&**normal\_tree**,** key\_norm**);**

InsertRoot**(&**rand\_tree**,** key\_rand**);**

**}**

int height\_normal **=** tree\_height**(**normal\_tree**);**

int height\_rand **=** tree\_height**(**rand\_tree**);**

int vowel\_count\_normal **=** count\_internal\_vowels**(**normal\_tree**);**

fprintf**(**output**,** "%15d | %15d | %25d | %25d\n"**,** current\_size**,** height\_normal**,** height\_rand**,** vowel\_count\_normal**);**

FreeTree**(**normal\_tree**);**

FreeTree**(**rand\_tree**);**

**}**

fclose**(**output**);**

printf**(**"Данные сохранены в height\_vowel\_data.txt.\n"**);**

**return** 0**;**

**}**

# **ЗАКЛЮЧЕНИЕ**

На основе результатов программы можно сделать следующие выводы о производительности различных алгоритмов поиска:

1. **Последовательный (линейный) поиск**:
   * В худшем варианте необходимо просмотреть все N элементов массива.
   * Число операций сравнения линейно увеличивается с размером массива (O(n)).
   * В нашем тесте для массива из 100 элементов потребовалось выполнить 100 итераций.
2. **Бинарный поиск**:
   * В среднем количество операций сравнения составило примерно 6.
   * Это значение близко к теоретической оценке log₂(100) ≈ 6.64.
   * Алгоритм показывает логарифмическую сложность (O(log n)).
3. **Поиск с помощью прыжков**:
   * В ходе эксперимента понадобилось примерно 20 итераций.
   * Теоретическая оценка для оптимального шага (√n) показывает, что для 100 элементов потребуется около 10 прыжков (√100 = 10).
   * Разница с полученными данными может быть объяснена особенностями реализации и структурой данных.

**Сравнительный анализ**:

* Бинарный поиск оказался самым эффективным с 6 итерациями.
* Поиск прыжками продемонстрировал среднюю эффективность, требуя 20 итераций.
* Линейный поиск оказался менее эффективным, так как потребовал 100 итераций.