

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»

Алгоритмы и обработка данных

Отчет по лабораторной работе №2

по теме: «Бинарные деревья поиска»

Выполнили студенты группы М3О-211Б-21

Мороз А.И.

Гордеева М.Д.

Проверила:

доцент каф. 304, к.т.н. Дмитриева Е.А.

Москва 2023 г.

Содержание

[Задание 3](#_Toc153111569)

[Код программы 4](#_Toc153111570)

[Результаты работы программы 11](#_Toc153111571)

[Вывод 14](#_Toc153111572)

# **Задание**

1. Реализовать функции вставки, поиска, удаления узла, обхода дерева, вывода дерева на экран, нахождения высоты дерева и количества узлов.
2. Реализовать дополнительно функцию в соответствии с вариантом: *T* – тип ключей, *D* – диапазон изменения значений ключей.

Вариант задания

|  |  |  |  |
| --- | --- | --- | --- |
| № | *T* | *D* | Функция |
| 6 | **char** | [а..я, А..Я] | Подсчет количества гласных на четных уровнях |

# **Код программы**

#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

using namespace std;

struct Node {

char key; // ключ узла

Node\* left; // указатель на левого потомка

Node\* right; // указатель на правого потомка

Node(char value) : key(value), left(nullptr), right(nullptr) {} // инициализация полей узла

};

// функция для вставки узла в бинарное дерево поиска

void Insert(Node\*& root, char key)

{

if (root == nullptr)

{

// если дерево пустое, создаем новый узел с ключом "key"

root = new Node(key);

return;

}

if (key < root->key) // если ключ меньше ключа текущего узла, рекурсивно идем влево

Insert(root->left, key);

else if (key > root->key) // если ключ больше ключа текущего узла, рекурсивно идем вправо

Insert(root->right, key);

}

// функция для поиска узла в бинарном дереве поиска

int Search(Node\* root, char key)

{

// если дерево пустое

if (root == nullptr)

return -1;

// ключ текущего узла равен искомому ключу

if (root->key == key)

return root->key;

// если ключ меньше ключа текущего узла, рекурсивно идем влево

if (key < root->key)

return Search(root->left, key);

// если ключ больше ключа текущего узла, рекурсивно идем вправо

return Search(root->right, key);

}

// функция для вывода дерева на экран

void PrintTree(Node\* root, int l = 0)

{

if (root)

{

PrintTree(root->right, l + 1); // рекурсивный вызов функции для правого поддерева

// вывод значения

for (int i = 0; i < l; i++) cout << " ";

cout << root->key << endl;

PrintTree(root->left, l + 1); // рекурсивный вызов функции для левого поддерева

}

}

// функция для нахождения высоты дерева

int TreeHeight(Node\* root)

{

if (!root)

return 0;

else

{

// рекурсивный вызов функции для правого поддерева

int left\_height = TreeHeight(root->left);

// рекурсивный вызов функции для левого поддерева

int right\_height = TreeHeight(root->right);

// макс. из высот левого и правого поддеревьев (+ 1 корень)

return 1 + max(left\_height, right\_height);

}

}

// функция для подсчета количества узлов в дереве

int CountNodes(Node\* root)

{

if (root == nullptr)

return 0;

// подсчет кол-ва узлов в левом и правом поддеревьях (+ 1 корень)

return 1 + CountNodes(root->left) + CountNodes(root->right);

}

void Delete(Node\*& root, char key)

{

Node\* parent = nullptr; // указатель на родителя

Node\* current = root; // на текущий узел

// поиск удаляемого узла и его родителя

while (current != nullptr && current->key != key)

{

parent = current; // сохранение текущего узла как потенциального родителя

if (key < current->key)

current = current->left; // переход влево, если искомый ключ меньше текущего

else

current = current->right; // переход вправо, если искомый ключ больше текущего

}

// удаление несуществующего элемента

if (current == nullptr)

{

cout << "Элемента с значением '" << key << "' не существует\n\n" << endl;

return;

}

// условия для разных случаев удаления узла

if (current->left == nullptr && current->right == nullptr) {

// удаление листового узла

if (parent != nullptr)

{

// обновление ссылки родителя на узел

if (parent->left == current)

parent->left = nullptr;

else

parent->right = nullptr;

}

else

root = nullptr; // удаляем корневой узел

delete current; // освобождение памяти

cout << "Узел со значением '" << key << "' являлся листовым узлом. Он был удален\n\n";

cout << "Дерево после удаления узла с ключом " << key << ":" << endl;

PrintTree(root);

cout << endl;

cout << "Высота дерева: " << TreeHeight(root) << "\n";

cout << "Кол-во узлов: " << CountNodes(root) << "\n\n";

}

else if (current->left != nullptr && current->right == nullptr)

{

// удаление узла с одним левым потомком

if (parent != nullptr)

{

// обновление ссылки родителя на потомка

if (parent->left == current)

parent->left = current->left;

else

parent->right = current->left;

}

else

root = current->left;

delete current; // освобождение памяти

cout << "Узел со значением '" << key << "' являлся узлом с левым потомком. Он был удален\n\n";

cout << "Дерево после удаления узла с ключом " << key << ":" << endl;

PrintTree(root);

cout << endl;

cout << "Высота дерева: " << TreeHeight(root) << "\n";

cout << "Кол-во узлов: " << CountNodes(root) << "\n\n";

}

else if (current->left == nullptr && current->right != nullptr)

{

// удаление узла с одним правым потомком

if (parent != nullptr)

{

// обновление ссылки родителя на потомка

if (parent->left == current)

parent->left = current->right;

else

parent->right = current->right;

}

else

root = current->right;

delete current; // освобождение памяти

cout << "Узел со значением '" << key << "' являлся узлом с правым потомком. Он был удален\n\n";

cout << "Дерево после удаления узла с ключом " << key << ":" << endl;

PrintTree(root);

cout << endl;

cout << "Высота дерева: " << TreeHeight(root) << "\n";

cout << "Кол-во узлов: " << CountNodes(root) << "\n\n";

}

else if (current->left != nullptr && current->right != nullptr)

{

// удаление узла с двумя потомками

Node\* Priem = current->left;

Node\* ParentPriem = nullptr;

// находим максимальный элемент в левом поддереве (предшественника)

while (Priem->right != nullptr) {

ParentPriem = Priem;

Priem = Priem->right;

}

// обновление связи для удаления текущего узла

if (ParentPriem != nullptr) {

ParentPriem->right = Priem->left;

Priem->left = current->left;

}

Priem->right = current->right;

if (parent != nullptr)

{

// обновление ссылки родителя на предшественника

if (parent->left == current)

parent->left = Priem;

else

parent->right = Priem;

}

else

root = Priem;

delete current; // освобождение памяти

cout << "Узел со значением '" << key << "' являлся узлом с двумя потомками. Он был удален\n\n";

cout << "Дерево после удаления узла с ключом " << key << ":" << endl;

PrintTree(root);

cout << endl;

cout << "Высота дерева: " << TreeHeight(root) << "\n";

cout << "Кол-во узлов: " << CountNodes(root) << "\n\n";

}

}

// функция для прямого обхода дерева

void Preoder(Node\* root)

{

if (root != nullptr)

{

cout << root->key << " "; // посещение корня

Preoder(root->left); // рекурсивный вызов для левого поддерева

Preoder(root->right); // рекурсивный вызов для правого поддерева

}

}

// функция для обратного обхода дерева

void Postoder(Node\* root)

{

if (root != nullptr)

{

Postoder(root->left); // рекурсивный вызов для левого поддерева

Postoder(root->right); // рекурсивный вызов для правого поддерева

cout << root->key << " "; // посещение корня

}

}

// функция для симметричного обхода дерева

void Inoder(Node\* root)

{

if (root != nullptr)

{

Inoder(root->left); // рекурсивный вызов для левого поддерева

cout << root->key << " "; // посещение корня

Inoder(root->right); // рекурсивный вызов для правого поддерева

}

}

// функция для подсчета кол-ва гласных букв на четных уровнях

void Glasn(Node\* root, int l, int& Count, char\* Glasnie)

{

if (root == nullptr) // проверка дерева на пустоту

return;

if (l % 2 == 0) // проверка уровня узла на четность

{

for (int i = 0; i < 12; i++) // проверка элемента на гласные

{

if ((root->key) == Glasnie[i]) // если элемент - гласный

{

Count++; // увеличиваем счетчик гласных на четных уровнях

cout << root->key << " \t" << l << endl; // вывод буквы и её уровня

break; // прекращаем проверку на гласные

}

}

}

Glasn(root->left, l + 1, Count, Glasnie); // рекурсивно проверяем левые поддеревья

Glasn(root->right, l + 1, Count, Glasnie); // рекурсивно проверяем правые поддеревья

}

int main()

{

setlocale(LC\_ALL, "Rus");

system("color F0");

Node\* root = nullptr; // инициализация корня дерева

cout << "\*\*\*\*\*\*\*\*Первоначальная информация\*\*\*\*\*\*\*\*\n";

srand(static\_cast<unsigned>(time(nullptr)));

char allLetters[] = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";

vector<char> availableLetters(allLetters, allLetters + sizeof(allLetters) - 1);

char P[16];

cout << "Узлы дерева: ";

for (int i = 0; i < 16; i++)

{

int randomIndex = rand() % availableLetters.size();

P[i] = availableLetters[randomIndex];

cout << P[i] << " ";

// Удаляем выбранную букву из вектора

availableLetters.erase(availableLetters.begin() + randomIndex);

}

cout << endl;

//char P[] = { 'b', 'f', 'd', 'g', 'I', 'D', 'A', 'a' }; // массив элементов дерева

int n = sizeof(P) / sizeof(P[0]); // кол-во элементов в массиве

int Count = 0; // число гласных на четных уровнях дерева

char Glasnie[12] = { 'a', 'e', 'i', 'o', 'u', 'y', 'A', 'E', 'I', 'O', 'U', 'Y' }; // массив гласных букв

// вставка узлов в дерево

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

{

Insert(root, P[i]);

}

cout << "Дерево:\n";

PrintTree(root);

cout << endl;

cout << "Высота дерева: " << TreeHeight(root) << "\n";

cout << "Кол-во узлов: " << CountNodes(root) << "\n";

cout << endl;

cout << "\*\*\*\*\*\*\*\*Подсчета количества гласных букв на четных уровнях\*\*\*\*\*\*\*\*\n";

cout << "Символ\tУровень" << endl;

Glasn(root, 1, Count, Glasnie);

cout << "Число гласных на четных уровнях = " << Count << "\n\n";

cout << "\*\*\*\*\*\*\*\*Поиск узла\*\*\*\*\*\*\*\*\n";

char searchKey;

cout << "Введите узел, который хотите найти: ";

cin >> searchKey;

char result = Search(root, searchKey);

if (result != -1)

cout << "Узел с ключом " << searchKey << " найден в дереве\n\n";

else

cout << "Узел с ключом " << searchKey << " не найден в дереве\n\n";

cout << "\*\*\*\*\*\*\*\*Обход дерева - 3 варианта\*\*\*\*\*\*\*\*\n";

cout << " 1)Прямой обход дерева: ";

Preoder(root);

cout << "\n\n";

cout << " 2)Обратный обход дерева: ";

Postoder(root);

cout << "\n\n";

cout << " 3)Симметричный обход дерева: ";

Inoder(root);

cout << "\n\n";

cout << "\*\*\*\*\*\*\*\*Удаление узла\*\*\*\*\*\*\*\*\n";

char deleteKey1;

cout << "Введите узел, который хотите удалить: ";

cin >> deleteKey1;

cout << endl;

Delete(root, deleteKey1);

return 0;

}

# **Результаты работы программы**

Вывод дерева, нахождение его высоты и кол-ва узлов

При каждом запуске программы дерево строится из рандомной последовательности узлов.

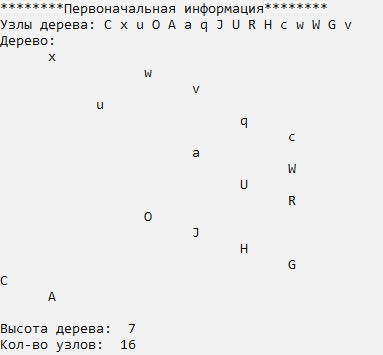


Рис. 1 – Дерево, созданное при первом запуске программы

Подсчет кол-ва гласных букв на четных уровнях

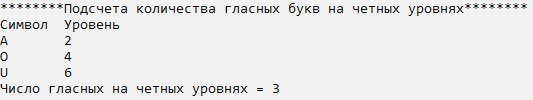


Рис. 2 – Подсчет количества гласных букв на четных уровнях в дереве из Рис. 1

Поиск узла



Рис. 3 – Поиск узла в дереве из Рис. 1

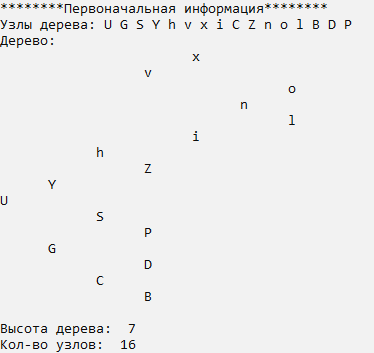


Рис. 4 – Дерево, созданное при втором запуске программы для вывода ошибки о несуществующем узле



Рис. 5 – Поиск несуществующего узла в дереве из Рис. 4

Обход дерева (прямой, обратный, симметричный)

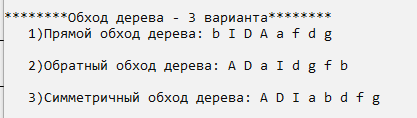
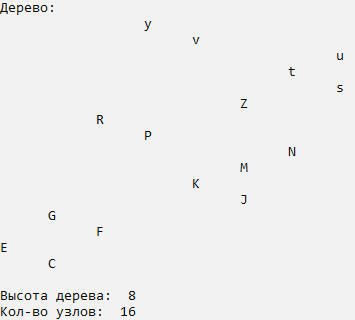
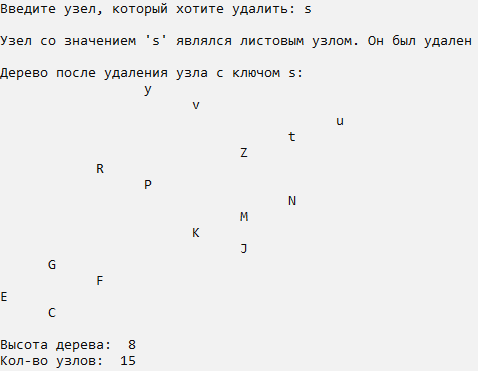


Рис. 6 – 3 варианта обхода дерева из Рис. 1

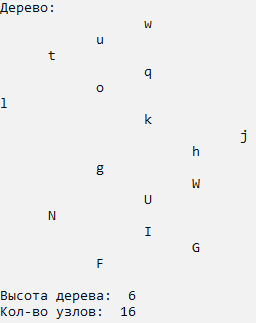
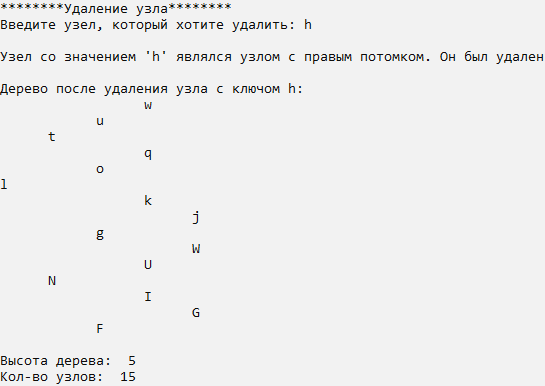
Удаление узла

* Листового (не имеющего потомков)

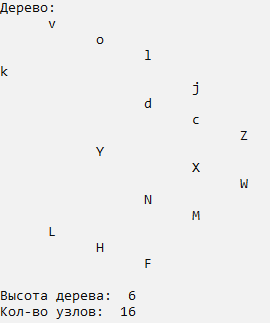
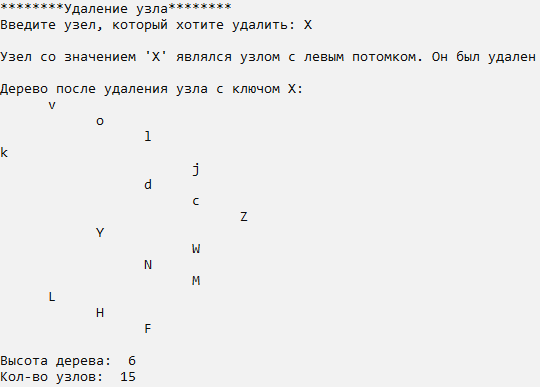
Узел 's' являлся листовым и был удален.

* С правым потомком

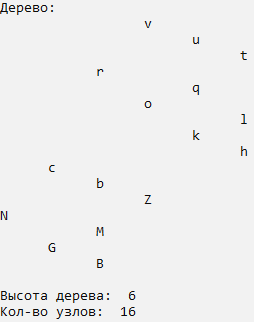
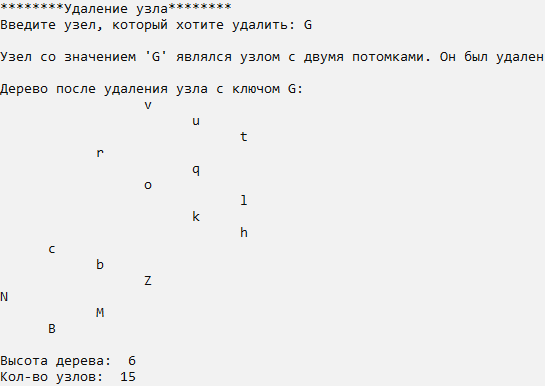
Узел 'h' имел правого потомка – 'j' и был удален. После удаления потомок «j» встал на его место.

* С левым потомком

Узел 'X' имел левого потомка – 'W' и был удален. После удаления потомок 'W' встал на его место.

* С двумя потомками

Узел 'G' имеет двух потомков – 'B' и 'M' и был удален. После удаления потомок 'B' встал на его место.

## **Вывод**

В результате выполнения данной лабораторной работы были успешно реализованы основные функции работы с бинарными деревьями. Функции вставки, поиска и удаления узла позволяют эффективно изменять структуру дерева, а функции обхода и вывода на экран позволяют получить наглядное представление о его содержимом.

Дополнительно была реализована функция поиска гласных букв на чётных уровнях бинарного дерева, а также функции нахождения высоты дерева и количества узлов, что позволяет получить информацию о его структуре и масштабе.

Результаты выполнения программы подтвердили корректность реализации всех функций. Были получены ожидаемые результаты при работе с бинарным деревом, что свидетельствует о правильной работе программы.

Работа завершена так как:

1) Полученные результаты совпадают с ожидаемыми;

2) Набор тестов считается полным.