Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа 11

По дисциплине «Основы алгоритмизации и программирования»

На тему «Бинарные деревья»

 Выполнила:

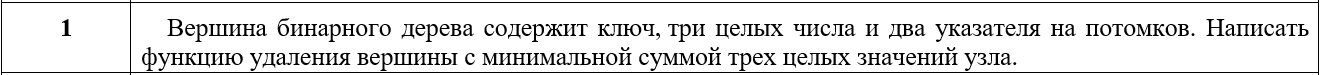
Студентка1 курса 6 группы

Альшевская Алина Михайловна

Преподаватель: асс. Андронова М.В.

2023, Минск

Вариант 1



#include <iostream>

// Определение структуры узла дерева

struct Node {

int key; // Ключ (значение), по которому происходит сортировка

int value1; // Значение 1

int value2; // Значение 2

int value3; // Значение 3

Node\* left; // Указатель на левого потомка

Node\* right; // Указатель на правого потомка

};

// Создание нового узла с заданными значениями

Node\* createNode(int key, int val1, int val2, int val3) {

Node\* newNode = new Node();

newNode->key = key;

newNode->value1 = val1;

newNode->value2 = val2;

newNode->value3 = val3;

newNode->left = newNode->right = nullptr;

return newNode;

}

// Вставка узла в дерево

Node\* insert(Node\* root, int key, int val1, int val2, int val3) {

if (root == nullptr) {

return createNode(key, val1, val2, val3);

}

if (key < root->key) {

root->left = insert(root->left, key, val1, val2, val3);

}

else if (key > root->key) {

root->right = insert(root->right, key, val1, val2, val3);

}

return root;

}

// Поиск узла с минимальным ключом

Node\* findMinNode(Node\* root) {

Node\* current = root;

while (current && current->left != nullptr) {

current = current->left;

}

return current;

}

// Удаление узла с заданным ключом и обновление минимальной суммы значений

Node\* deleteNode(Node\* root, int keyVal, int& minSum) {

if (root == nullptr) {

return nullptr;

}

if (keyVal < root->key) {

root->left = deleteNode(root->left, keyVal, minSum);

}

else if (keyVal > root->key) {

root->right = deleteNode(root->right, keyVal, minSum);

}

else {

int currentSum = root->value1 + root->value2 + root->value3;

if (currentSum < minSum) {

minSum = currentSum;

}

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

delete root;

return nullptr;

}

else if (root->left == nullptr) {

Node\* temp = root->right;

delete root;

return temp;

}

else if (root->right == nullptr) {

Node\* temp = root->left;

delete root;

return temp;

}

Node\* temp = findMinNode(root->right);

root->key = temp->key;

root->value1 = temp->value1;

root->value2 = temp->value2;

root->value3 = temp->value3;

root->right = deleteNode(root->right, temp->key, minSum);

}

return root;

}

// Обход дерева в порядке возрастания ключей (in-order traversal)

void inorderTraversal(Node\* root) {

if (root == nullptr) {

return;

}

inorderTraversal(root->left);

std::cout << "Key: " << root->key << " Values: " << root->value1 << " " << root->value2 << " " << root->value3 << std::endl;

inorderTraversal(root->right);

}

int main() {

Node\* root = nullptr;

root = insert(root, 8, 2, 3, 4);

root = insert(root, 3, 5, 6, 7);

root = insert(root, 10, 1, 2, 3);

root = insert(root, 1, 4, 5, 6);

root = insert(root, 6, 3, 2, 1);

root = insert(root, 14, 8, 9, 10);

std::cout << "Before deletion:" << std::endl;

inorderTraversal(root);

int minSum = INT\_MAX;

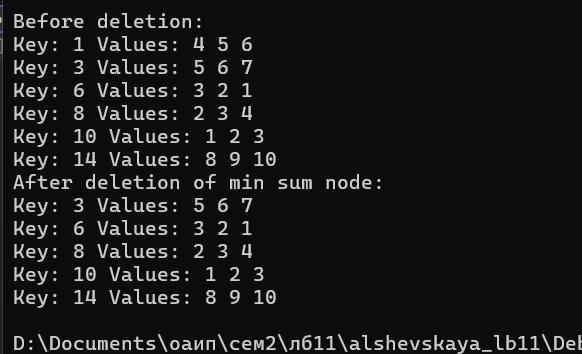
root = deleteNode(root, findMinNode(root)->key, minSum);

std::cout << "After deletion of min sum node:" << std::endl;

inorderTraversal(root);

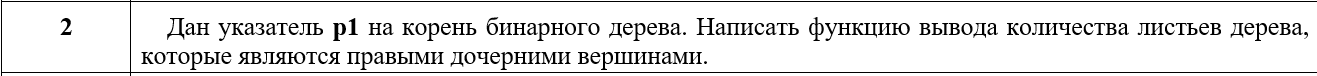
return 0;

}



Дополнительные задания

Вариант 2



#include <iostream>

// Определение структуры узла дерева

struct Node {

int data; // Значение узла

Node\* left; // Указатель на левого потомка

Node\* right; // Указатель на правого потомка

Node(int val) : data(val), left(nullptr), right(nullptr) {}

};

// Функция для подсчета количества правых листьев в дереве

int countRightLeaves(Node\* root) {

if (root == nullptr) {

return 0;

}

// Если узел имеет правого потомка и этот потомок - лист, увеличиваем счетчик

if (root->right != nullptr && root->right->left == nullptr && root->right->right == nullptr) {

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

}

else {

return countRightLeaves(root->left) + countRightLeaves(root->right);

}

}

int main() {

setlocale(LC\_ALL, "Russian");

// Пример создания бинарного дерева

Node\* p1 = new Node(1);

p1->left = new Node(2);

p1->right = new Node(3);

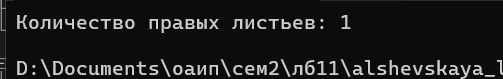
p1->left->left = new Node(4);

p1->right->right = new Node(5);

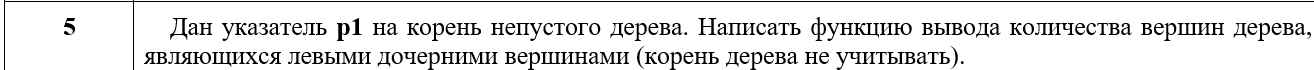
std::cout << "Количество правых листьев: " << countRightLeaves(p1) << std::endl;

return 0;

}



Вариант 5



#include <iostream>

// Определение структуры узла дерева

struct Node {

int data; // Значение узла

Node\* left; // Указатель на левого потомка

Node\* right; // Указатель на правого потомка

Node(int data) : data(data), left(nullptr), right(nullptr) {}

};

// Функция для подсчета количества левых дочерних узлов в дереве

int countLeftChildNodes(Node\* root) {

if (root == nullptr || (root->left == nullptr && root->right == nullptr)) {

return 0;

}

int count = 0;

if (root->left != nullptr) {

count++;

}

count += countLeftChildNodes(root->left);

count += countLeftChildNodes(root->right);

return count;

}

int main() {

setlocale(LC\_ALL, "Russian");

// Пример создания бинарного дерева

Node\* p1 = new Node(1);

p1->left = new Node(2);

p1->right = new Node(3);

p1->left->left = new Node(4);

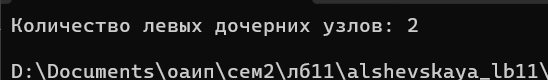
p1->left->right = new Node(5);

int result = countLeftChildNodes(p1);

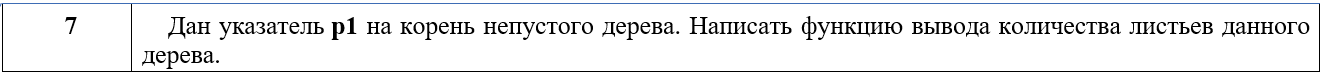
std::cout << "Количество левых дочерних узлов: " << result << std::endl;

return 0;

}



Вариант 7



#include <iostream>

// Определение структуры узла дерева

struct Node {

int data;

Node\* left;

Node\* right;

Node(int data) : data(data), left(nullptr), right(nullptr) {}

};

// Рекурсивная функция для подсчета количества листьев

int countLeaves(Node\* root) {

if (root == nullptr) {

return 0;

}

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

return 1;

}

// Рекурсивно считаем количество листьев в левом и правом поддереве

return countLeaves(root->left) + countLeaves(root->right);

}

int main() {

setlocale(LC\_ALL, "Russian");

// Пример создания бинарного дерева

Node\* root = new Node(1);

root->left = new Node(2);

root->right = new Node(3);

root->left->left = new Node(4);

root->left->right = new Node(5);

// Вызываем функцию для подсчета листьев

int leafCount = countLeaves(root);

std::cout << "Количество листьев в дереве: " << leafCount << std::endl;

return 0;

}

