Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа № 11

По дисциплине «Основы алгоритмизации и программирования»

На тему «Бинарные деревья»

Выполнил:

Кулешов Артём

Студент 1 курса 8 группы

Преподаватель: асс. Андронова М.В.

Минск, 2024

3. Разработать программу работы с ***бинарным деревом поиска***, в которую включить основные функции манипуляции данными и функцию в соответствии со своим вариантом из таблицы, представленной ниже.

Вариант №5

Дан указатель **p1** на корень непустого дерева. Написать функцию вывода количества вершин дерева, являющихся левыми дочерними вершинами (корень дерева не учитывать).

Вариант №10

Дан указатель **p1** на корень непустого дерева. Написать функцию вывода вывода суммы значений всех вершин данного дерева.

Вариант №12

Дан указатель **p1** на корень непустого дерева. Написать функцию определения количества узлов с четными ключами.

#include <iostream>

#include <string>

using namespace std;

struct Tree

{

int key;

char text[5];

Tree\* Left, \* Right;

};

Tree\* tree = nullptr;

int counter = 0;

int ch = 0;

int sum = 0;

void insert(int a, string name, Tree\*\* t, int& ch, int& sum);

void print(Tree\* t, int u);

void delAll(Tree\* t);

Tree\* DeleteNum(Tree\* node, int val);

int main()

{

string name;

int choice, num;

Tree\* rc,s[5];

do {

cout << "1. Add elements." << endl;

cout << "2. Output elements." << endl;

cout << "3. Output the number of tree vertices that are left child vertices (option 5)" << endl;

cout << "4. Number of nodes with even keys. (option 12)" << endl;

cout << "5. The sum of the values ​​of all tree vertices. (option 10)" << endl;

cout << "6. Delete certain values." << endl;

cout << "7. Delete all." << endl;

cout << "8. Exit." << endl;

cin >> choice;

switch (choice) {

case 1:

cout << "Number:";

cin >> num;

cout << "Word:";

getline(cin, name);

getline(cin, name);

insert(num, name, &tree, ch, sum);

break;

case 2:

print(tree, 0);

break;

case 3:

cout << "Left: " << counter << endl;

break;

case 4:

cout << "Even keys: " << ch << endl;

break;

case 5:

cout << "Sum of the values: " << sum << endl;

break;

case 6:

cout << "Number:";

cin >> num;

DeleteNum(tree, num);

counter -= 1;

break;

case 7:

delAll(tree);

break;

case 8:

cout << "..." << endl;

exit(0);

break;

default:

cout << "Error." << endl;

break;

}

} while (choice != 8);

return 0;

}

void insert(int a, string name, Tree\*\* t, int& ch, int& sum) //добавление элемента и подсчет четных ключей//

{

if ((\*t) == NULL) //если дерева нет, то создается элемент//

{

(\*t) = new Tree; //создание элемента//

(\*t)->key = a;

(\*t)->Left = (\*t)->Right = NULL;

if (a % 2 == 0) { //проверка на чётность//

ch++;

}

sum += a; //сумма значений всех вершин//

return;

}

if (a > (\*t)->key)

{

insert(a, name, &(\*t)->Right, ch, sum);

}

else {

insert(a, name, &(\*t)->Left, ch, sum);

}

}

void print(Tree\* t, int u) //вывод//

{

if (t == NULL)

{

return;

}

else {

print(t->Left, ++u);

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

cout << "|"; //вывод//

cout << t->key << endl;

u--;

}

print(t->Right, ++u);

}

void delAll(Tree\* t) //удаление//

{

if (t != NULL)

{

delAll(t->Left); //удаление левой//

delAll(t->Right); //правой//

delete t;

}

else {

tree = NULL;

}

}

Tree\* DeleteNum(Tree\* node, int val) { //удаление определенных значений//

if (node == NULL)

{

return node;

}

if (val == node->key) {

Tree\* tmp;

if (node->Right == NULL) //если нет правого поддерева//

{

tmp = node->Left; //перемещаем левое поддерево на место удаляемого узла//

}

else {

Tree\* ptr = node->Right;

if (ptr->Left == NULL) //если правое поддерево не имеет левого потомка//

{

ptr->Left = node->Left; //присваиваем левому потомку правого поддерева левое поддерево удаляемого узла//

tmp = ptr; //перенос указателя на правое поддерево//

}

else {

Tree\* pmin = ptr->Left;

while (pmin->Left != NULL) { //минимальный элемент в правом поддереве//

ptr = pmin;

pmin = ptr->Left;

}

ptr->Left = pmin->Right;

pmin->Left = node->Left;

pmin->Right = node->Right;

tmp = pmin; //перенос указатель на минимальный элемент//

}

}

delete node; //удаляем узел//

return tmp; //возвращаем указатель на новый корень поддерева//

}

else if (val < node->key) //если значение меньше значения текущего узла, рекурсивно ищем в левом поддереве//

{

node->Left = DeleteNum(node->Left, val);

}

else { //если значение больше значения текущего узла, рекурсивно ищем в правом поддереве//

node->Right = DeleteNum(node->Right, val);

}

return node;

}

 

 