БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационных технологий и управления

Кафедра вычислительных методов и программирования

ОТЧЕТ

по лабораторной работе

**«Нелинейные списки»**

|  |  |  |
| --- | --- | --- |
| Выполнила |  | Семенова В. С. |
| Проверила |  | Шатилова О. О. |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Минск 2023

**Цель**: изучить алгоритмы обработки данных с использованием

нелинейных структур в виде дерева.

**Общая постановка задачи**

Разработать проект для работы с деревом поиска, содержащий следующие обработчики, которые должны:

– ввести информацию в массив. Каждый элемент массива должен содержать строку текста и целочисленный ключ (например, ФИО и номер паспорта);

– внести информацию из массива в дерево поиска;

– сбалансировать дерево поиска;

– добавить в дерево поиска новую запись;

– по заданному ключу найти информацию и отобразить ее;

– удалить из дерева поиска информацию с заданным ключом;

– распечатать информацию прямым, обратным обходом и в порядке возрастания ключа;

– решить одну из поставленных задач.

**Вариант 11**

Определить количество записей в дереве, начинающихся с определенной буквы (например, «a»).

**Листинг**

#include <iostream>

#include <windows.h>

#include <string.h>

#include <string>

using namespace std;

struct Info {

int ID; string FIO;

};

struct Tree {

int ID; string FIO;

Tree\* l, \* r;

};

Tree\* List(int , string);

void chet(Tree\* , Info\* , int\* );

void Add\_List(Tree\* , int , string );

Tree\* Del(Tree\*, int);

void View(Tree\*, int);

void Del\_All(Tree\*);

void poisk(Tree\*,int );

void Make\_Blns(Tree\*\*, int, int, Info\*);

void sort(Info\*, int);

void View\_W(Tree\*);

void View\_O(Tree\*);

void View\_P(Tree\*);

Info\* addElemet(Info\*, int\*, string, int);

int TZ(Tree\*, char);

bool proverka(string);

void kol\_vo(Tree\*, int\*);

void kol\_vo(Tree\* r, int\* k) {

if (!r) return;

else {

(\*k)++;

kol\_vo(r->l, k);

kol\_vo(r->r, k);

}

}

Tree\* List(int i, string a)

{

Tree\* t = new Tree;

t->ID = i;

t->FIO = a;

t->l = t->r = NULL;

return t;

}

void chet(Tree\* root , Info\* a , int\* kol)

{

if (!root) return;

else {

a[\*kol].FIO = root->FIO;

a[\*kol].ID = root->ID;

(\*kol)++;

chet(root->r,a,kol);

chet(root->l, a, kol);

}

}

void Add\_List(Tree\* root, int key, string FIO)

{

Tree\* prev = root, \* t;

int find = 1;

t = root;

while (t && find) {

prev = t;

if (key == t->ID) {

find = 0;

cout << " Такой уже есть !" << endl;

}

else

if (key < t->ID) t = t->l;

else

t = t->r;

}

if (find) {

t = List(key,FIO);

if (key < prev->ID)

prev->l = t;

else

prev->r = t;

}

}

Tree\* Del(Tree\* root, int key) {

Tree\* Del, \* Prev\_Del, \* R, \* Prev\_R;

Del = root;

Prev\_Del = NULL;

while (Del != NULL && Del->ID != key) {

Prev\_Del = Del;

if (key < Del->ID) Del = Del->l;

else Del = Del->r;

}

if (Del == NULL) {

cout << "\n Элемент не найден!" << endl;

return root;

}

if (Del->r == NULL) R = Del->l;

else

if (Del->l == NULL) R = Del->r;

else {

Prev\_R = Del;

R = Del->l;

while (R->r != NULL) {

Prev\_R = R;

R = R->r;

}

if (Prev\_R == Del)

R->r = Del->r;

else {

R->r = Del->r;

Prev\_R->r = R->l;

R->l = Prev\_R;

}

}

if (Del == root) root = R;

else

if (Del->ID < Prev\_Del->ID)

Prev\_Del->l= R;

else

Prev\_Del->r = R;

cout << " Удалили " << Del->ID << endl;

delete Del;

return root;

}

void View(Tree\* t, int level) {

if (t) {

View(t->r, level + 1);

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

cout << " ";

cout << t->ID << endl;

View(t->l, level + 1);

}

}

void View\_W(Tree\* t) {

if (t) {

View\_W(t->l);

cout << t->ID << endl;

View\_W(t->r);

}

}

void View\_O(Tree\* t) {

if (t) {

View\_O(t->l);

View\_O(t->r);

cout << t->ID << endl;

}

}

void View\_P(Tree\* t) {

if (t) {

cout << t->ID << endl;

View\_O(t->l);

View\_O(t->r);

}

}

void Del\_All(Tree\* t) {

if (t != NULL) {

Del\_All(t->l);

Del\_All(t->r);

delete t;

}

}

void poisk(Tree\* root, int key) {

Tree\* isk=root;

while (isk != NULL && isk->ID != key) {

if (key < isk->ID) isk = isk->l;

else isk = isk->r;

}

if (isk == NULL) cout << "\nЭлемент не найден!" << endl;

else cout << "ID: " << isk->ID << endl << "ФИО: " << isk->FIO << endl;

}

void Make\_Blns(Tree\*\* p, int n, int k, Info\* a)

{

if (n == k) {

\*p = NULL;

return;

}

else {

int m = (n + k) / 2;

\*p = new Tree;

(\*p)->ID = a[m].ID;

(\*p)->FIO = a[m].FIO;

Make\_Blns(&(\*p)->l, n, m, a);

Make\_Blns(&(\*p)->r, m + 1, k, a);

}

}

void sort(Info a[], int n) {

struct

{

int l;

int r;

} st[20]{};

int i, j, left, right, s = 0,x;

Info t;

st[s].l = 0; st[s].r = n - 1;

while (s != -1)

{

left = st[s].l; right = st[s].r;

s--;

while (left < right)

{

i = left; j = right; x= a[(left + right) / 2].ID;

while (i <= j)

{

while (a[i].ID < x) i++;

while (a[j].ID > x) j--;

if (i <= j) {

t = a[i]; a[i] = a[j]; a[j] = t;

i++; j--;

}

}

if ((j - left) < (right - i))

{

if (i < right) { s++; st[s].l = i; st[s].r = right; }

right = j;

}

else {

if (left < j) { s++; st[s].l = left; st[s].r = j; }

left = i;

}

}

}

}

Info\* addElemet(Info\* m, int \*n, string f,int id)

{

Info\* temp = NULL;

temp = new Info[\*n + 1];

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

temp[i] = m[i];

temp[\*n].FIO = f;

temp[\*n].ID = id;

(\*n)++;

delete[]m;

return temp;

}

int TZ(Tree\* r, char ch) {

if (!r) return 0;

if (r) {

int i = TZ(r->l, ch) + TZ(r->r, ch);

if (r->FIO[0] == ch) return i + 1;

else return i;

}

}

bool proverka(string D) {

int p = 0, u = 0, i = 1;

if (D[0] == '-') {

p++;

}

else {

if (D[0] == '.') {

p++; u++;

}

else {

for (char ii = '0'; ii <= '9'; ii++) {

if (D[0] == ii) {

p++;

}

}

}

}

if (p == 1) {

while (D[i] != '\0') {

if (D[i] == ' ') {

return false;

}

for (char ii = '0'; ii <= '9'; ii++) {

if (D[i] == ii) {

p++;

}

}

if (u == 0) {

if (D[i] == '.') {

p++; u++;

}

}

i++;

}

}

else {

return false;

}

if (p != i) {

return false;

}

if (i + 1 > 1 && D[0] == '0' && D[1] != '.') {

return false;

}

if (i + 1 > 1 && D[0] == '-' && D[1] == '0' && D[2] != '.') {

return false;

}

return true;

}

int main()

{

setlocale(LC\_ALL, "Russian");

SetConsoleCP(1251);

int n, in,i,j=0, key,l=0, kol=0; Info\* a, buf; string fio, prov; Tree\* root = NULL; char k, ch = NULL; bool m;

cout << "Введите кол-во узлов: "; cin >> n;

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

m = TRUE;

cout << " Введите ID : ";

cin >> prov;

if (proverka(prov) == false) { cout << "Ошибка ввода, введите другое значение" << endl; i--; }

else {

in = stod(prov);

cout << " Введите ФИО : ";

cin >> fio;

if (root == NULL)

root = List(in, fio);

else

Add\_List(root, in, fio);

}

}

while (TRUE) {

cout << "\n1) Добавить новую запись;\n2) Найти узел;\n3) Удалить узел;\n4) Вывод информации на экран;\n5) Найти кол-во записей по условию\n6) Сбалансировать дерево;\n0) Выход\n"; cin>>k;

switch (k) {

case'1':

cout << " Введите ID : ";

cin >> prov;

if (proverka(prov) == false) { cout << "Ошибка ввода, введите другое значение" << endl; i--; }

else {

in = stod(prov);

cout << " Введите ФИО : ";

cin >> fio;

if (root == NULL)

root = List(in, fio);

else

Add\_List(root, in, fio);

}

break;

case'2':

cout << "Введите ID: ";

cin >> key;

poisk(root, key);

break;

case'3':

cout << " Введите ID удаляемого объекта : ";

cin >> key;

root = Del(root, key);

break;

case'4':

cout << "1) В виде дерева; 2) Прямым обходом; 3) Обратным обходом; 4) В порядке возрастания ключа;\n "; cin >> k; cout << "----------------------------------" << endl;

switch (k) {

case'1':View(root, 0); break;

case'2':View\_P(root); break;

case'3':View\_O(root); break;

case'4':View\_W(root); break;

default: cout << "Неправильный ввод!\n";

}

break;

case'5': cout << "Введите букву: "; cin >> ch; cout<<TZ(root, ch); break;

case'6': j = 0; kol\_vo(root, &j); a = new Info[j]; kol = 0; chet(root, a, &kol); sort(a, j); Del\_All(root); Make\_Blns(&root, 0, j, a); cout << "Дерево сбалансировано" << endl; break;

case'0': return 0;

default: cout << "Неправильный ввод!\n";

}

}

}