**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра информационных систем**

**ОТЧЕТ**

**по лабораторной работе №3**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: СБАЛАНСИРОВАННЫЕ ДЕРЕВЬЯ ПОИСКА**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9375 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Гоголев И. Л. |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Пелевин М. С. |

Санкт-Петербург

2020

**Цель:**

Реализовать АВЛ -дерево.

**Задание к лабораторной работе:**

Реализовать АВЛ -дерево, функцию вставки, поиска и удаления.

**Текст программы:**

#include <iostream>

#include <windows.h>

#include <conio.h>

using namespace std;

class AVLNode

{

private:

int Key = NULL;

short int Height;

public:

AVLNode\* left = nullptr;

AVLNode\* right = nullptr;

short int height()

{

return Height;

}

AVLNode(int k)

{

Key = k;

left = NULL;

right = NULL;

Height = 1;

}

short int correctHeight()

{

if ((right == NULL) && (left == NULL))

return Height = 1;

if (right == NULL)

return Height = left->height() + 1;

if (left == NULL)

return Height = right->height() + 1;

if ((left->height()) > (right->height()))

Height = left->height() + 1;

else

Height = right->height() + 1;

return Height;

}

short int balance()

{

if ((right == NULL) && (left == NULL))

return 1;

if (right == NULL)

return -left->height();

if (left == NULL)

return right->height();

return right->height() - left->height();

}

int key()

{

return Key;

}

void initialization(int k)

{

Key = k;

}

};

AVLNode\* rotateR(AVLNode\* r)

{

AVLNode\* l = r->left;

r->left = l->right;

l->right = r;

l->correctHeight();

r->correctHeight();

return l;

}

AVLNode\* rotateL(AVLNode\* l)

{

AVLNode\* r = l->right;

l->right = r->left;

r->left = l;

l->correctHeight();

r->correctHeight();

return r;

}

AVLNode\* toBalance(AVLNode\* l)

{

l->correctHeight();

if (l->balance() >= 2) //r>l

{

if ((l->right)->balance() < 0)//rr<rl

l->right = rotateR(l->right);//правый поворот r

return rotateL(l); //левый поворот

}

if (l->balance() <= -2) //l>r

{

if ((l->left)->balance() > 0) //lr>ll

l->left = rotateL(l->left); //левый поворот l

return rotateR(l); // правый поворот

}

return l;

}

AVLNode\* insert(AVLNode\* l, int k)

{

if (l == nullptr)

return toBalance(new AVLNode(k));

if (k < l->key())

l->left = insert(l->left, k);

if (k > l->key())

l->right = insert(l->right, k);

return toBalance(l);

}

AVLNode\* minNode(AVLNode\* l)

{

if (l->left != nullptr)

return minNode(l->left);

else

return l;

}

AVLNode\* delMin(AVLNode\* l)

{

if (l->left == nullptr)

return l->left;

l->left = delMin(l->left);

return toBalance(l);

}

AVLNode\* delNode(AVLNode\* m, int k)

{

if (m == nullptr) //ищем ключ

return NULL;

if (k < m->key())

m->left = delNode(m->left, k);

else if (k > m->key())

m->right = delNode(m->right, k);

else //нашли

{

AVLNode\* l = m->left;

AVLNode\* r = m->right;

if (r == nullptr) // r - пустое, просто возращаем l

return l;

AVLNode\* min = minNode(r);//если r не пустой - находим минимальный элемент под r.

//к min слева - l, справа - r

min->right = delMin(r);

min->left = l;

return toBalance(min); // min с поддеревьями

}

return toBalance(m);

}

int dec(int x) {

int n = 1;

while ((x /= 10) > 0) n++;

return n;

}

void printTree(AVLNode\* p, int level, int& maxLevel, bool find = false, int findKey = 0)

{

if (p)

{

printTree(p->right, ++level, maxLevel, find, findKey);

level--;

if (maxLevel < level)

maxLevel = level;

for (int i = 0; i <= 8 \* level + 6; i++)

cout << ' ';

if ((p->key() == findKey) && (find == true))

cout << "(" << p->key() << ')';

else

cout << "|" << p->key();

if (p->height() != 1)

{

if ((p->key() == findKey) && (find == true))

{

for (int i = 0; i <= 5 - dec(p->key()); i++)

cout << "-";

cout << '|' << endl;

}

else

{

for (int i = 0; i <= 6 - dec(p->key()); i++)

cout << "-";

cout << '|' << endl;

}

}

else cout << endl;

printTree(p->left, ++level, maxLevel, find, findKey);

}

}

void printTreeString(AVLNode\* p)

{

if (p)

{

printTreeString(p->right);

cout << p->key() << ' ';

printTreeString(p->left);

}

}

bool finder(AVLNode\* p, int key, bool& f)

{

if (p)

{

f = finder(p->right, key, f);

if (p->key() == key)

return f = true;

f = finder(p->left, key, f);

}

return f;

}

int getRandomNumber(int min, int max)

{

static const double fraction = 1.0 / (static\_cast<double>(RAND\_MAX) + 1.0);

return static\_cast<int>(rand() \* fraction \* (max - min + 1) + min);

}

void presentTree()

{

cout << " \_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << " | |" << endl;

cout << " | Avl Tree |" << endl;

cout << " |\_\_\_\_\_\_\_\_\_\_\_\_|" << endl;

cout << endl;

cout << " ArrowUp | Добавить элемент |" << endl;

cout << "ArrowDown | Заполнить случайными элементами |" << endl;

int i;

int firstE;

bool random = false;

i = \_getch();

switch (i)

{

case(72):

{

cout << endl;

cout << "Введите какой элемент хотите добавить: ";

cin >> firstE;

break;

}

case(80):

{

cout << endl;

random = true;

cout << "Количество элементов: ";

cin >> i;

break;

}

case(27):

{

system("cls");

exit(0);

}

default:

system("cls");

return;

}

system("cls");

if (random == true)

{

firstE = getRandomNumber(10, 99);

}

AVLNode t(firstE);

AVLNode\* a1 = &t;

if (random == true)

{

for (int j = 0;j < i - 1;j++)

a1 = insert(a1, getRandomNumber(10, 99));

}

int find = 0;

bool f = false;

for (;1 == 1;)

{

int sizeL = 0;

int maxLevel = 0;

cout << endl;

printTree(a1, 0, maxLevel, f, find);

cout << endl;

printTreeString(a1);

cout << endl;

cout << endl;

cout << " ArrowUp | Добавить элемент" << endl;

cout << "ArrowRight | Удалить элемент" << endl;

cout << "ArrowDown | Найти элемент" << endl;

cout << " Esc | Выйти" << endl;

i = \_getch();

switch (i)

{

case(72):

{

cout << endl << "Какой элемент Вы хотите добавить(НОВЫЙ ЭЛ., ПРЕДЫДУЩИЙ ЭЛ.): ";

cin >> i;

a1 = insert(a1, i);

find = 0;

f = false;

break;

}

case(77):

{

cout << endl << "Какой элемент Вы хотите удалить: ";

cin >> i;

a1 = delNode(a1, i);

find = 0;

f = false;

break;

}

case(80):

{

cout << endl << "Какой элемент Вы хотите найти: ";

cin >> i;

find = i;

f = true;

bool fin = false;

fin = finder(a1, i, fin);

if (fin == true)

cout << "Такой элемент присутствует в дереве" << endl;

else

cout << "Такого элемента неть" << endl;

break;

}

case(27):

{

system("cls");

exit(0);

}

}

}

}

int main()

{

setlocale(LC\_CTYPE, "rus");

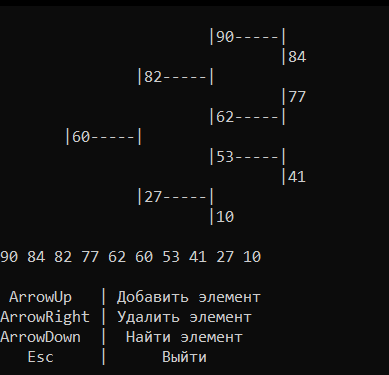
for (;1 == 1;)

presentTree();

return 0;

}

**Пример работы программы:**

****