ГУО “БГУИР”

Факультет компьютерного проектирования

Кафедра Инженерной психологии и эргономики

Отчёт по

Лабораторной работе №5

СРЕДА ПРОГРАММИРОВАНИЯ VISUAL C++. ПРОГРАММИРОВАНИЕ ЛИНЕЙНЫХ АЛГОРИТМОВ

Подготовил:

Студент гр.110101

Ладутько Я.Д.

Проверила:

Семижон Е.А.

Минск 2022

***Цель работы:*** изучить алгоритмы обработки данных с использованием нелинейных структур в виде дерева.

Вариант 16.



Код:

**///MySpace.h**

#pragma once

#include <iostream>

template <typename T>

void input(T& a)

{

while (true)

{

std::cin >> a;

if (std::cin.fail() || (std::cin.peek() != '\n' && std::cin.peek() != ' ' && std::cin.peek() != '\t'))

{

std::cin.clear();

std::cin.ignore(10000, '\n');

std::cout << "Неверный ввод" << std::endl;

}

else

{

return;

}

}

}

void input(int& x, int a, int b);

void input(bool& b);

**///MySpace.cpp**

#include"MySpace.h"

using namespace std;

void input(int& x, int a, int b)

{

while (true)

{

input(x);

if (x < a || x > b)

{

cout <<

"Неверный ввод\n"

"Введите число на промежутке от " << a << " до " << b << endl;

}

else

return;

}

}

void input(bool& b)

{

char c;

while (true)

{

input(c);

if (c != 'y' && c != 'n')

{

cout <<

"Неверный ввод\n"

"Введите y или n\n";

}

else

{

b = (c == 'y' ? true : false);

return;

}

}

}

**///Tree.h**

#pragma once

#include <string>

struct node

{

public:

int key;

std::string info;

int height;

node\* left;

node\* right;

node(int k, std::string inf) { key = k; left = right = nullptr; height = 1; info = inf; }

};

class Tree

{

public:

node\* root = nullptr;

void addNode(int key, std::string info);

void show();

node\* findByKey(int key);

void printLeftRootRight();

void printRootLeftRight();

void printLeftRightRoot();

void removeByKey(int key);

void clear();

~Tree()

{

clear();

}

};

**///Tree.cpp**

#include "Tree.h"

#include <iomanip>

#include <iostream>

#include <string>

#include <vector>

int height(node\* p)

{

return p ? p->height : 0;

}

int bFactor(node\* p)

{

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

}

void updateHeight(node\* p)

{

int hl = height(p->left);

int hr = height(p->right);

p->height = (hl > hr ? hl : hr) + 1;

}

node\* rotateRight(node\* p)

{

node\* q = p->left;

p->left = q->right;

q->right = p;

updateHeight(p);

updateHeight(q);

return q;

}

node\* rotateLeft(node\* q)

{

node\* p = q->right;

q->right = p->left;

p->left = q;

updateHeight(q);

updateHeight(p);

return p;

}

node\* balance(node\* p)

{

updateHeight(p);

if (bFactor(p) == 2)

{

if (bFactor(p->right) < 0)

p->right = rotateRight(p->right);

return rotateLeft(p);

}

if (bFactor(p) == -2)

{

if (bFactor(p->left) > 0)

p->left = rotateLeft(p->left);

return rotateRight(p);

}

return p;

}

node\* find(node\* p, int key)

{

if (p == nullptr)

return nullptr;

if (p->key == key)

return p;

else if (p->key > key)

return find(p->left, key);

return find(p->right, key);

}

node\* insert(node\* p, node\* new\_node)

{

if (!p)

return new\_node;

if (new\_node->key < p->key)

p->left = insert(p->left, new\_node);

else

p->right = insert(p->right, new\_node);

return balance(p);

}

void Tree::addNode(int key, std::string info)

{

if (root == nullptr)

root = new node(key, info);

else

root = insert(root, new node(key, info));

updateHeight(root);

}

void Tree::show()

{

if (root == nullptr)

{

std::cout << "Дерево пустое\n";

return;

}

std::vector<node\*> v;

node\* curr\_node = root;

int max\_height = curr\_node->height;

v.resize((1 << max\_height));

v[0] = root;

int otstup = 1 << max\_height;

for(int i = 0; i < (1<<(max\_height-1)) - 1; i ++)

{

if (v[i] != nullptr)

{

v[2 \* i + 1] = v[i]->left;

v[2 \* i + 2] = v[i]->right;

}

}

for(int j = 0; j < max\_height; j++)

{

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

{

if(j==0)

std::cout << "\033[33m"; // yellow

else if (i % 2 == 0)

std::cout << "\033[32m"; // green

else

std::cout << "\033[31m"; // red

if(v[(1 << j) - 1 + i]!=nullptr)

std::cout << std::setw(otstup) << v[(1<<j) - 1 + i]->key << std::setw(otstup)<< ' ';

else

std::cout << std::setw(otstup) << '-' << std::setw(otstup) << ' ';

}

std::cout << std::endl;

otstup /= 2;

std::cout << "\033[0m";

}

}

node\* Tree::findByKey(int key)

{

return find(root, key);

}

void printLeftRootRight\_rec(node\* n)

{

if (n == nullptr)

return;

printLeftRootRight\_rec(n->left);

std::cout << n->info << ' ';

printLeftRootRight\_rec(n->right);

}

void Tree::printLeftRootRight()

{

printLeftRootRight\_rec(root);

}

void printRootLeftRight\_rec(node\* n)

{

if (n == nullptr)

return;

std::cout << n->info << ' ';

printLeftRootRight\_rec(n->left);

printLeftRootRight\_rec(n->right);

}

void Tree::printRootLeftRight()

{

printRootLeftRight\_rec(root);

}

void printLeftRightRoot\_rec(node\* n)

{

if (n == nullptr)

return;

printLeftRootRight\_rec(n->left);

printLeftRootRight\_rec(n->right);

std::cout << n->info << ' ';

}

void Tree::printLeftRightRoot()

{

printLeftRootRight\_rec(root);

}

node\* findmin(node\* p)

{

return p->left ? findmin(p->left) : p;

}

node\* removemin(node\* p)

{

if (p->left == nullptr)

return p->right;

p->left = removemin(p->left);

return balance(p);

}

node\* remove(node\* p, int k)

{

if (!p) return 0;

if (k < p->key)

p->left = remove(p->left, k);

else if (k > p->key)

p->right = remove(p->right, k);

else

{

node\* q = p->left;

node\* r = p->right;

delete p;

if (!r) return q;

node\* min = findmin(r);

min->right = removemin(r);

min->left = q;

return balance(min);

}

return balance(p);

}

void Tree::removeByKey(int key)

{

remove(root, key);

}

void clear\_rec(node\* p)

{

if (p == nullptr)

return;

clear\_rec(p->left);

clear\_rec(p->right);

delete p;

}

void Tree::clear()

{

clear\_rec(root);

root = nullptr;

}

**///Lab5.cpp**

#include <iostream>

#include "MySpace.h"

#include "Tree.h"

using namespace std;

int individualTask\_rec(node\* p)

{

if (p == nullptr)

return 0;

if (p->left == nullptr && p->right == nullptr)

return 1;

return individualTask\_rec(p->left) + individualTask\_rec(p->right);

}

void individualTask(Tree& tree)

{

if (tree.root == nullptr)

{

std::cout << "Дерево пустое\n";

return;

}

if (tree.root->left == nullptr)

{

std::cout << "У дерева нет левой ветви\n";

return;

}

cout<< "Количество листов в левой ветви: " << individualTask\_rec(tree.root->left)<<endl;

}

int main()

{

setlocale(LC\_ALL, "RU");

Tree tree;

/\*for(int i = 1; i <= 15; i++)

{

tree.addNode(i, to\_string(i));

}

tree.show();\*/

int c;

while (true)

{

cout << "1 - Добавить\n2 - Поиск\n3 - Удалить\n4 - Показать\n5 - Обход\n6 - Очистить\n7 - Individual Task\n0 - Выход\n";

input(c, 0, 7);

system("cls");

switch (c)

{

case 1:

{

cout << "Введите ключ: ";

int key;

input(key);

if (tree.findByKey(key) != nullptr)

{

cout << "Ключ уже существует\n";

system("pause");

break;

}

cout << "Введите модель двери: ";

string s;

cin >> s;

tree.addNode(key, s);

break;

}

case 2:

{

cout << "Введите ключ: ";

int key;

input(key);

node\* n = tree.findByKey(key);

if (n == nullptr)

{

cout << "Ключа не существует\n";

system("pause");

break;

}

cout << "Модель: " << n->info<<endl;

system("pause");

break;

}

case 3:

{

cout << "Введите ключ, чтобы удалить: ";

int key;

input(key);

node\* n = tree.findByKey(key);

if (n == nullptr)

{

cout << "Ключа не существует\n";

system("pause");

break;

}

tree.removeByKey(key);

break;

}

case 4:

{

tree.show();

system("pause");

break;

}

case 5:

{

if (tree.root == nullptr)

{

std::cout << "Дерево пустое\n";

system("pause");

break;

}

cout << "1 - Left-Root-Right\n2 - Root-Left-Right\n3 - Left-Right-Root\n0 - Выход\n";

int choise;

input(choise, 0, 3);

switch(choise)

{

case 1:

tree.printLeftRootRight();

system("pause");

break;

case 2:

tree.printRootLeftRight();

system("pause");

break;

case 3:

tree.printLeftRightRoot();

system("pause");

break;

case 0:

break;

}

break;

}

case 6:

{

if (tree.root == nullptr)

{

std::cout << "Дерево пустое\n";

system("pause");

break;

}

cout << "Вы действительно хотите очистить дерево? y/n\n";

bool b;

input(b);

if (b)

tree.clear();

break;

}

case 7:

{

individualTask(tree);

system("pause");

break;

}

case 0:

{

return 0;

}

}

system("cls");

}

}

***Вывод:*** изучил алгоритмы обработки данных с использованием нелинейных структур в виде дерева.

