МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра прикладной математики

Рекурсия. N-деревья

ОТЧЕТ

По лабораторной работе

по дисциплине

Технология программирования

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Санников А.Н.

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Зырянов Е.А.

22-ПМ-1

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Задание на лабораторную работу:

1. Дано N-дерево. Найти в дереве самое широкое (узкое) поддерево имеющее заданную высоту.

Цель работы:

Познакомиться с .N-деревьями.

Ход работы:

1) Написали программу:

#include <iostream>

template <typename type>

type min(type a, type b) {

if (a > b) {

return b;

}

return a;

}

template <typename type>

type max(type a, type b) {

if (a > b) {

return a;

}

return b;

}

template <typename type>

class sptr {

private:

type\* ptr = NULL;

size\_t length = 0;

public:

sptr() {}

sptr(size\_t n) {

ptr = new type[n];

length = n;

}

~sptr() {

std::cout << ptr << std::endl;

if (ptr) { delete[] ptr; }

}

sptr& resize(size\_t n) {

type\* nptr = new type[n];

for (size\_t i = 0; i < min(length, n); i++) {

nptr[i] = ptr[i];

}

if (ptr) delete[] ptr;

ptr = nptr;

length = n;

return \*this;

}

sptr& add(type a) {

resize(length + 1);

ptr[length - 1] = a;

return \*this;

}

size\_t len() {

return length;

}

type& operator [] (size\_t i) {

if (!(i < length)) {

std::cout << "OPERATOR[] ERROR" << '\n';

exit(-1);

}

return ptr[i];

}

};

template <typename type>

std::ostream& operator << (std::ostream& stream, sptr<type>& ptr) {

for (size\_t i = 0; i < ptr.len(); i++) {

std::cout << ptr[i] << ' ';

}

return stream;

}

struct node {

int data = 0;

node\* pnode = NULL;

sptr<node\*> cnodes;

};

class tree {

friend std::ostream& operator << (std::ostream& stream, tree& tr);

private:

node\* root = NULL;

node\* tmp = NULL;

size\_t level = 0;

public:

tree() {}

tree& gnode(size\_t i) {

if (!i) {

if (!tmp->pnode) {

std::cout << "GO PNODE ERROR" << '\n';

exit(-1);

}

tmp = tmp->pnode;

level--;

}

else if (i <= tmp->cnodes.len()) {

tmp = tmp->cnodes[i - 1];

level++;

}

return \*this;

}

tree& add(int a) {

node\* n = new node;

n->data = a;

n->pnode = tmp;

tmp->cnodes.add(n);

return \*this;

}

tree& sroot(int a) {

tmp = new node;

root = tmp;

tmp->data = a;

return \*this;

}

tree& res() {

tmp = root;

return \*this;

}

node\* get\_root() {

return root;

}

int get\_val() {

return tmp->data;

}

size\_t get\_level() {

return level;

}

};

std::ostream& operator << (std::ostream& stream, tree& tr) {

std::cout << tr.tmp->data;

return stream;

}

bool is\_num(char\* s) {

for (size\_t i = 0; s[i] != '\0'; i++)

{

if (not(int(s[i]) >= 48 and int(s[i]) < 58)) {

return false;

}

}

return true;

}

void move(size\_t& a, char\* s) {

a = 0;

size\_t k = 0;

while (s[k] != '\0') {

k++;

}

for (size\_t i = 0; i < k; i++)

{

a += pow(10, k - i - 1) \* (int(s[i]) - 48);

}

}

void read\_tree(tree& t) {

using namespace std;

char tmp[10];

size\_t data;

for (size\_t i = 0; true; i++)

{

cout << "Введите ребёнка " << i + 1 << " узла " << t.get\_val() << " - ";

gets\_s(tmp, 10);

if (tmp[0] != '\0') {

while (!is\_num(tmp)) {

for (size\_t i = 0; i < 10; i++)

{

tmp[i] = 0;

}

cout << "Нужно вводить числа - ";

gets\_s(tmp, 10);

if (tmp[0] == '\0') {

goto metka;

}

}

move(data, tmp);

t.add(data);

t.gnode(i + 1);

read\_tree(t);

t.gnode(0);

}

else {

metka:

break;

}

}

}

void print\_subtree(node\* n, int indent = 0) {

if (n == nullptr) {

return;

}

for (int i = 0; i < indent; i++) {

std::cout << " ";

}

std::cout << n->data << std::endl;

for (int i = 0; i < n->cnodes.len(); i++) {

print\_subtree(n->cnodes[i], indent + 1);

}

}

int height(node\* n) {

if (n == nullptr) {

return 0;

}

int max\_height = 0;

for (int i = 0; i < n->cnodes.len(); i++) {

max\_height = max(max\_height, height(n->cnodes[i]));

}

return max\_height + 1;

}

void find\_subtrees(tree& t, int target\_height, node\* current\_node = nullptr, int current\_height = 0) {

static node\* widest = nullptr;

static node\* narrowest = nullptr;

static int max\_width = -1;

static int min\_width = INT\_MAX;

if (current\_node == nullptr) {

current\_node = t.get\_root();

}

int num\_children = current\_node->cnodes.len();

if (current\_height == target\_height) {

if (num\_children > max\_width) {

max\_width = num\_children;

widest = current\_node;

}

if (num\_children < min\_width) {

min\_width = num\_children;

narrowest = current\_node;

}

}

for (int i = 0; i < current\_node->cnodes.len(); i++) {

find\_subtrees(t, target\_height, current\_node->cnodes[i], current\_height + 1);

}

if (current\_node == t.get\_root() && widest != nullptr && narrowest != nullptr) {

std::cout << "Поддерево, начинающееся " << widest->data << ", имеет самый широкий узел с " << max\_width << " детьми и высотой: " << target\_height << std::endl;

print\_subtree(widest, 0);

std::cout << "Поддерево, начинающееся с " << narrowest->data << ", имеет самый узкий узел с " << min\_width << " детьми и высотой: " << target\_height << std::endl;

print\_subtree(narrowest, 0);

}

}

int main() {

using namespace std;

setlocale(LC\_ALL, "Russian");

tree Tree;

int data, height;

cout << "Введите корень дерева - ";

cin >> data;

getc(stdin);

Tree.sroot(data);

read\_tree(Tree);

cout << '\n';

cout << "Введите высоту поиска - ";

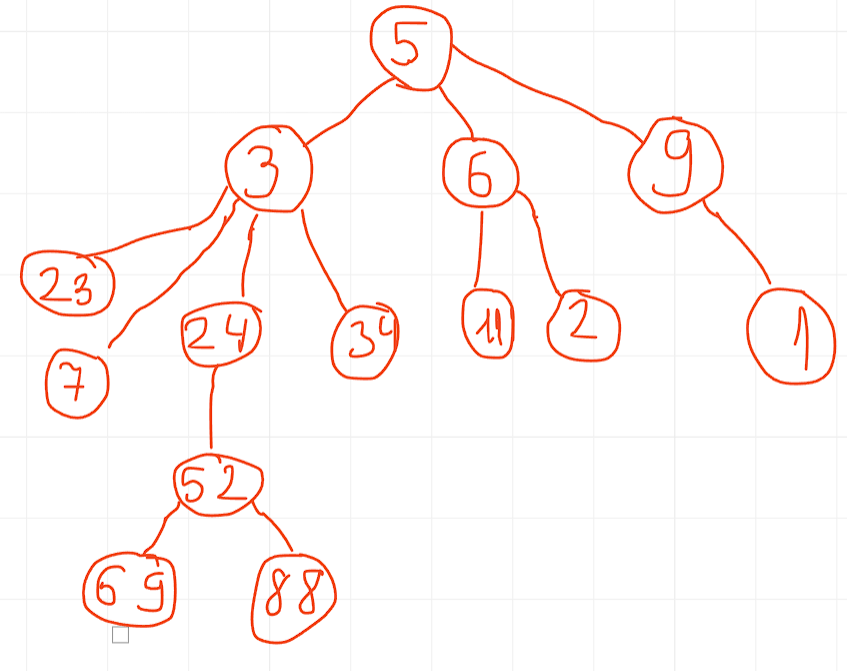
cin >> height; cout << '\n';

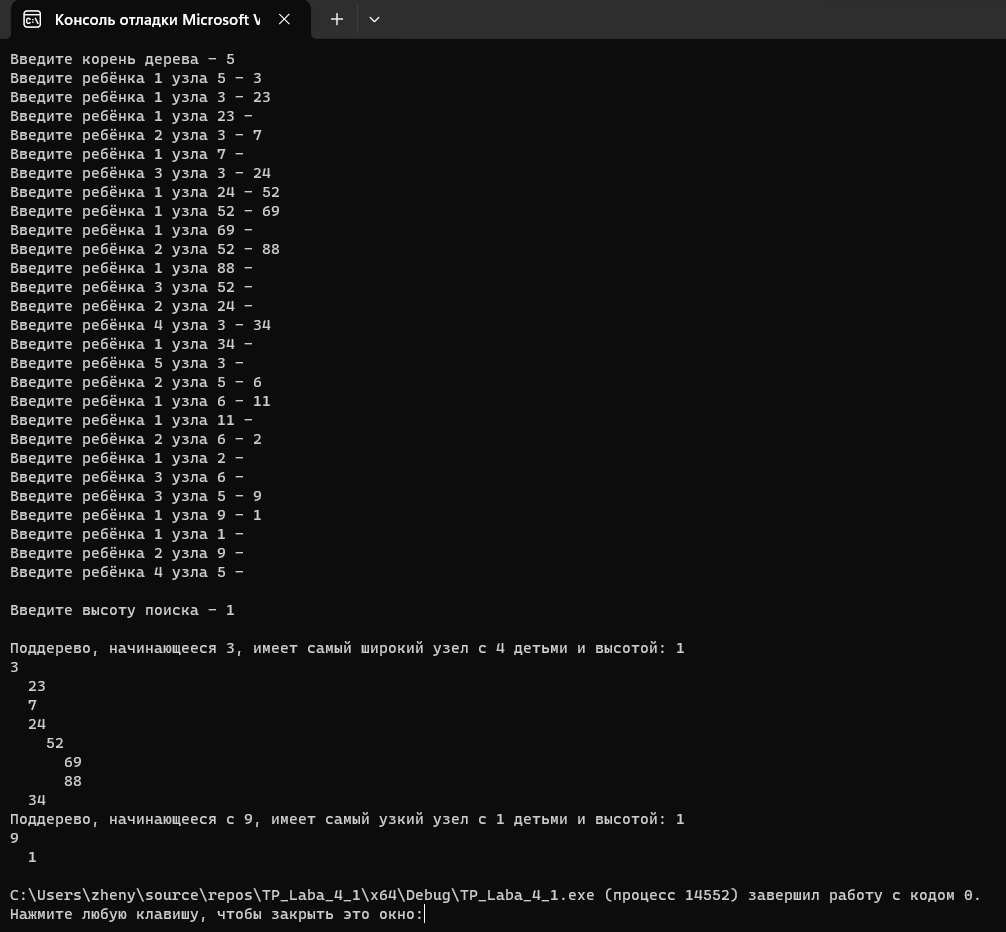
find\_subtrees(Tree, height, Tree.get\_root());

return 0;

}

Результат:





Вывод: Познакомились с .N-деревьями.