**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

отчет

**по лабораторной работе № 3**

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

Тема: **«Деревья»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 1308 |  | Макаров М.В. |
| Студент гр. 1308 |  | Томилов Д. |  | Макаров М.В. |
| Преподаватель |  | Манирагена В. |  | Морозов С.М. |

Санкт-Петербург

2022

**Цель работы.**

Целью работы является изучение деревьев (двоичных и троичных) и их реализация на языке С++.

**Задание (вариант 3)**

Вид дерева: троичное

Разметка: обратная

Способ обхода: внутренний

Нужно вычислить: количество вершин, имеющих хоть одного потомка

**Постановка задачи и описание решения**

Для реализации алгоритмов были созданы 2 класса: класс дерево и класс узел, причем класс узел является приватным классом класса дерево. Сделано это, чтобы методы этого класса не были доступны извне.

Дерево имеет в себе указатель на узел, являющийся корнем дерева. Каждый узел, в свою очередь, имеет указатели на дочерние узлы. Такая реализация была выбрана из-за её очевидности и интуитивной понятности. Также на ней удобно реализовывать рекурсивные алгоритмы.

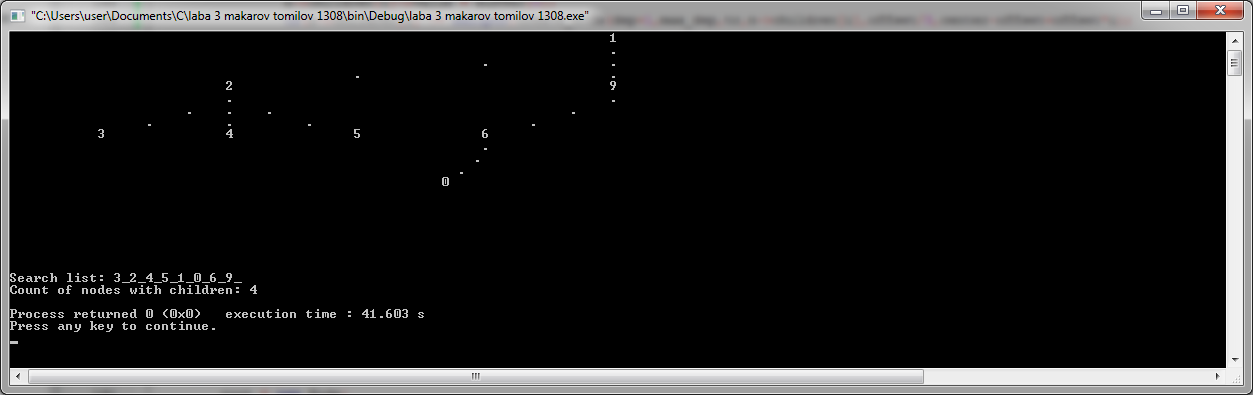
Были реализованы методы создания дерева посредством его ввода с клавиатуры и посредством генерации случайного дерева. При создании случайного дерева разметка производится в обратном порядке. Это значит, что сначала создаются элементы ветви, и только при подъёме вверх им задаются значения, каждый раз увеличивая значение на 1.

Также был реализован внутренний обход. Внутренний обход подразумевает, что элемент обрабатывается либо если он является листом, либо если при проходе вверх до этого узла ранее уже была полностью обработана хотя бы одна ветка, исходящая из этого узла.

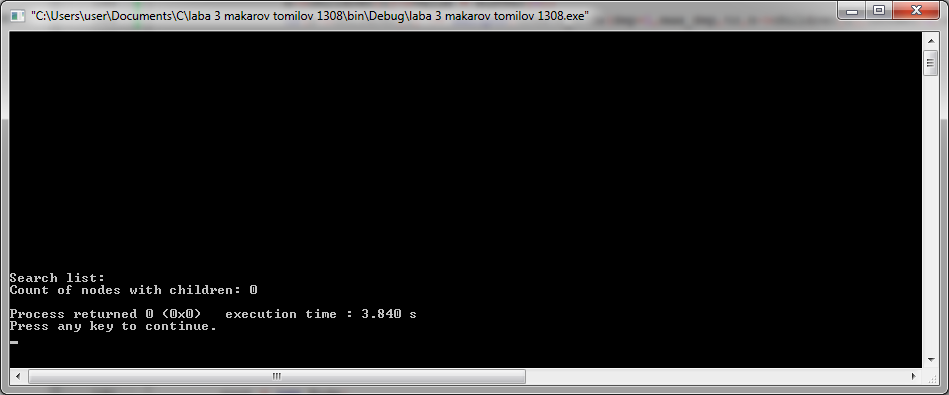
Чтобы вычислить количество вершин, имеющих хоть одного потомка, при внутреннем обходе смотрим, есть ли хоть один потомок у узла, который в данный момент обрабатывается, и соответствующим образом обновляем счетчик.

**Примеры выполнения программы**

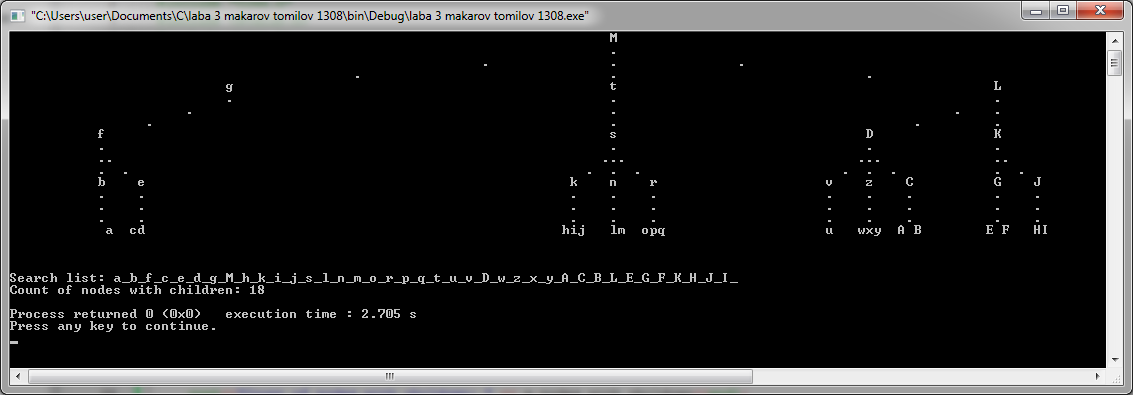
**Пример 1(тестовый):**



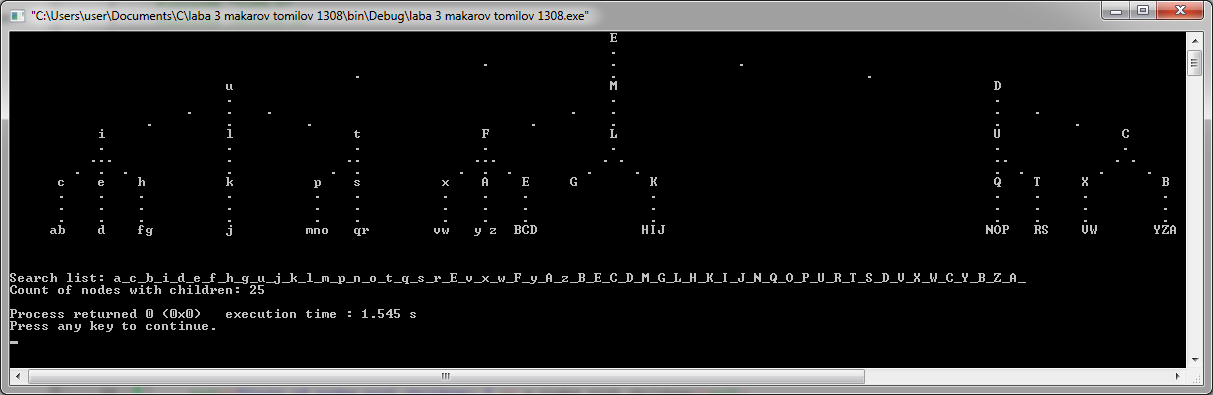
**Пример 2(пустое дерево):**



**Пример 3 (случайное дерево):**



**Пример 4 (случайное дерево):**



**Оценка временной сложности**

Создание дерева выполняется за n шагов, где n – количество вершин. Далее при проходе вверх по дереву тратится ещё n шагов на присвоение узлам значений. Следовательно, сложность создания дерева O(n).

Внутренний обход дерева также проходит по всем элементам по одному разу, следовательно, его сложность также O(n).

На обработку одного элемента требуется константное время, так как просто проверяются 3 поля в элементе (указатели на потомков), следовательно, временная сложность обработки O(1).

Сложность вывода также O(n), поскольку каждый элемент выводится на экран 1 раз.

В целом сложность обработки всего дерева = O внутреннего обхода \* O обработки одного элемента = O(n) \* O(1) = O(n).

**Выводы.**

В результате были изучены способы программного представления деревьев. Алгоритмы, использованные в данной работе, можно считать эффективными, ведь обход дерева невозможен за время, меньшее чем O(n).

Текст программы:

//main.cpp//

#include <iostream>

#include "Tree.h"

#include <stdlib.h>

#include <time.h>

using namespace std;

int main()

{

string buff;

srand(time(0));

Tree a(20,150);

cout << "1 - Random tree\n2-create tree\nYOUR ANSWER: ";

cin >>buff;

if(buff[0] == '2'){

a.create\_tree(5);

}else{

a.create\_random\_tree(5);

system("cls");

}

a.show();

buff = a.inner\_search();

cout<<"Search list: " << buff<<endl;

cout<<"Count of nodes with children: " << a.nodes\_with\_children<<endl;

return 0;

}

//Tree.h//

#pragma once

#include "random"

#include <iostream>

#include <string.h>

class Tree

{

private:

class Node

{

private:

Node\* children[3];

char value;

void node\_show(Tree\* tr,int dep,int offset,int center);

void inner\_add(std::string\* s,Tree\* tr);

public:

Node();

Node(const int);

~Node();

friend class Tree;

};

static int tag;

static const int ch\_count;

const int depth;

const int breadth;

char\*\* screen;

Node\* root;

public:

void scr\_update();

int nodes\_with\_children;

void create\_tree(const int dep);

void create\_node(const int dep,const int max\_dep,Tree\* tr,Node\* n,int offset,int center);

static Node\* random\_node(const int dep,const int max\_dep);

void create\_random\_tree(const int dep);

std::string inner\_search();

void show();

Tree(int depth, int breadth);

~Tree();

};

//Tree.cpp//

#include "Tree.h"

const int Tree::ch\_count = 3;

int Tree::tag = 0;

Tree::Node::Node()

{

int i;

for(i = 0;i<ch\_count;i++){

children[i] = nullptr;

}

}

Tree::Node::Node(const int val)

{

int i;

for(i = 0;i<ch\_count;i++){

children[i] = nullptr;

}

value = val;

}

Tree::Node::~Node()

{

int i;

for(i = 0;i<ch\_count;i++){

if(children[i] != nullptr){

delete children[i];

}

}

//std::cout << "delete node " << value <<std::endl;

}

Tree::Tree(int dep,int br):depth(dep),breadth(br)

{

int i,j;

root = nullptr;

screen = new char\*[depth];

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

screen[i] = new char[breadth+1];

for(j = 0;j<breadth;j++) screen[i][j] = ' ';

screen[i][j] = 0;

}

nodes\_with\_children = 0;

}

Tree::~Tree()

{

int i;

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

delete[] screen[i];

}

delete[] screen;

delete root;

//std::cout << "delete tree\n";

}

Tree::Node\* Tree::random\_node(const int dep,const int max\_dep)

{

Node\* ret = nullptr;

int i;

if((rand()%6>1 &&dep < max\_dep)||dep == 0){

ret = new Node;

}else ret = nullptr;

if(ret != nullptr){

for(i = 0;i<ch\_count;i++){

ret->children[i] = random\_node(dep+1,max\_dep);

}

if(tag <26){

ret->value = 'a' + (tag++)%26;

}else{

ret->value = 'A' + (tag++)%26;

}

}

return ret;

}

void Tree::create\_random\_tree(const int dep)

{

delete root;

root = nullptr;

root = random\_node(0,dep);

}

void Tree::show()

{

scr\_update();

int i;

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

std::cout << screen[i]<<std::endl;

}

}

void Tree::Node::node\_show(Tree\* tr,int dep,int offset,int center)

{

int i,j,up,down;

tr->screen[dep][center] = value;

for(i = 0;i<Tree::ch\_count;i++){

if(children[i] != nullptr){

up = center;

down = center-offset+offset\*i;

for(j = 0;j<3;j++){

tr->screen[dep+j+1][center - ((up-down)\*j)/3] = '.';

}

children[i]->node\_show(tr,dep+4,offset/3,center-offset+offset\*i);

}

}

}

void Tree::scr\_update()

{

if(root != nullptr) root ->node\_show(this,0,breadth/3-2,breadth/2);

}

std::string Tree::inner\_search()

{

std::string ret;

ret = "";

nodes\_with\_children = 0;

if(root != nullptr) root->inner\_add(&ret,this);

return ret;

}

void Tree::Node::inner\_add(std::string\* s,Tree\* tr)

{

int cntr,i,show,a;

cntr = 0;

show = 1;

a = 1;

for(i = 0;i<Tree::ch\_count;i++){

if(children[i] != nullptr){

if(cntr == 1){

if(children[0] !=nullptr||children[1] !=nullptr||children[2] !=nullptr){

tr->nodes\_with\_children++;

//std::cout <<value<<tr->nodes\_with\_children<<" ";

}

\*s+=value;

\*s +="\_";

show = 0;

}

cntr++;

children[i]->inner\_add(s,tr);

}

}

if(show){

if(children[0] !=nullptr||children[1] !=nullptr||children[2] !=nullptr){

tr->nodes\_with\_children++;

//std::cout <<value<<tr->nodes\_with\_children<<" ";

}

\*s+=value;

\*s +="\_";

}

}

void Tree::create\_node(const int dep,const int max\_dep,Tree\* tr,Node\* n,int offset,int center)

{

int i;

std::string buffer;

if(dep <max\_dep){

for(i = 0;i<ch\_count;i++){

buffer = "-000";

system("cls");

screen[dep\*4][center-offset+offset\*i] = '|';

show();

screen[dep\*4][center-offset+offset\*i] = ' ';

std::cout << "Enter node(type '-' if you don't want this node to exist): ";

std::cin >>buffer;

if(buffer[0] != '-'){

n->children[i] = new Node;

n->children[i]->value = buffer[0];

if(n->children[i] != nullptr)create\_node(dep+1,max\_dep,tr,n->children[i],offset/3,center-offset+offset\*i);

}

}

}

}

void Tree::create\_tree(const int dep)

{

Node\* ret = nullptr;

int i;

std::string buffer;

buffer = "-000";

delete root;

root = nullptr;

system("cls");

screen[0][breadth/2] = '|';

show();

screen[0][breadth/2] = ' ';

std::cout << "Enter node(type '-' if you don't want this node to exist): ";

std::cin >>buffer;

if(buffer[0] != '-'){

root = new Node;

root ->value = buffer[0];

create\_node(1,dep,this,root,breadth/3-2,breadth/2);

}else root = nullptr;

system("cls");

}