**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ.УЛЬЯНОВА (ЛЕНИНА)**

Кафедра вычислительной техники

Отчет по лабораторной работе №3 на тему:

«Деревья»

по дисциплине «Алгоритмы и структуры данных»

Колинько П.Г.

Санкт-Петербург

2018

Торопов В. А.

Преподаватель

Студент гр. 7307

Оглавление

[Цель работы 3](#_Toc527985075)

[Задание 3](#_Toc527985076)

[Измерения времени 4](#_Toc527985077)

[Результат эксперимента с отслеживанием вызова функций-членов. 4](#_Toc527985078)

[Вывод 6](#_Toc527985079)

[Код программы 7](#_Toc527985080)

# **Цель работы**

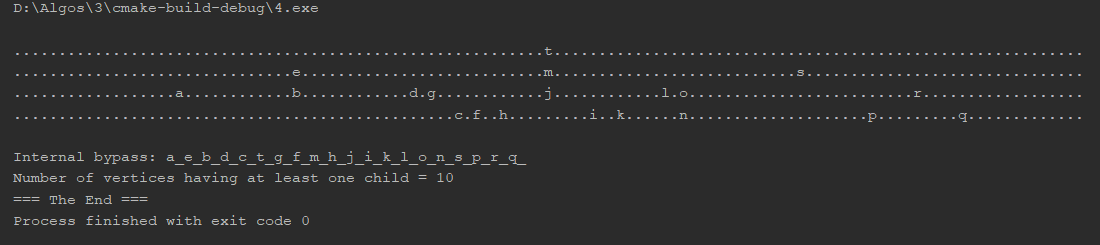
Исследование алгоритмов обхода дерева.

# **Задание**

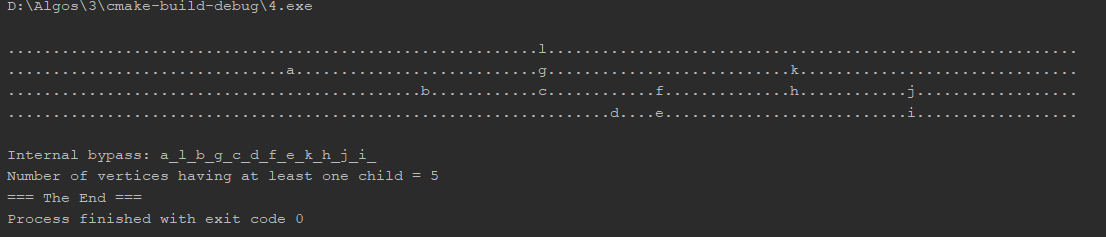
Написать и отладить программу для работы с троичным деревом с обратной разметкой. Вычислить количество вершин, имеющих хотя бы одного потомка внутренним способом обхода.

Для представления дерева в памяти используется разветвляющийся список, так как это самый удобный и эффективный способ работы с деревьями.

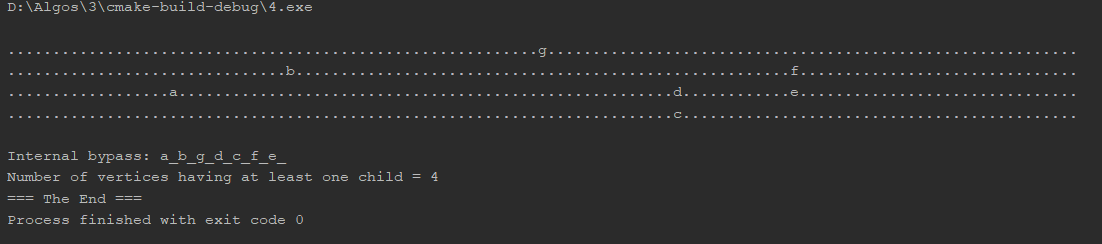
# Результаты прогона программы с генерацией случайного дерева



Последовательность битов: 0001000100111001110111001101010000101111



Последовательность битов: 0000000000001000100011101100000001010111



Последовательность битов: 0001000000001000000000000001010001000011

# Оценка временной сложности

* Создание дерева: O(n)
* Обработка: O(n)
* Вывод: O(n)

# Вывод

В ходе данной работы были получены практические знания по работе с деревьями, а также с алгоритмами работы с деревьями. Время работы алгоритмов обхода дерева внутренним способом и обратным одинаково.

# **Код программы**

Main:

#include <iostream>  
#include "Tree.h"  
  
**int** main()  
{  
 **int** n = 0;  
 Tree Tr('a','z', 4);  
 srand(time(**nullptr**));  
 Tr.MakeTree();  
 **if**(Tr.exist()){  
 Tr.OutTree();  
 cout << '\n' << "Internal bypass: ";  
 n = Tr.bypass();  
 cout << '\n' << "Number of vertices having at least one child = " << n;  
 }  
 **else** cout << "Tree Is Empty!";  
 cout << '\n' << "=== The End ===";  
 //\_getwch();  
 **return** 0;  
}

Tree.h:

#ifndef INC\_4\_TREE\_H  
#define INC\_4\_TREE\_H  
#include <iostream>  
#include <time.h>  
#include <cstring>  
  
**using namespace** std;  
  
#endif //INC\_4\_TREE\_H  
  
**class** Node {  
 **char** d;  
 Node \*left;  
 Node \*right;  
 Node \*middle;  
**public**:  
 Node(): d('\0'), left(**nullptr**), right(**nullptr**), middle(**nullptr**){}  
 ~Node(){  
 **delete** right;  
 **delete** left;  
 **delete** middle;  
 }  
 **friend class** Tree;  
};  
  
**class** Tree{  
 Node \*root;  
 **char** num, maxnum;  
 **int** maxrow, offset;  
 **char** \*\*SCREEN;  
 **void** clrscr();  
 Node \*MakeNode(**int** depth);  
 **void** OutNodes(Node \*v, **int** r, **int** c);  
 **int** restask = 0;  
**public**:  
 Tree(**char** num, **char** maxnum, **int** maxrow);  
 ~Tree();  
 **void** MakeTree(){  
 root = MakeNode(0);  
 }  
 **bool** exist(){**return** root != **nullptr**;}  
 **void** OutTree();  
 **void** task(Node \*ob);  
 **int** bypass(){  
 task(root);  
 **return** restask;  
 }  
};  
  
Tree::Tree(**char** nm, **char** mnm, **int** mxr):  
 num(nm), maxnum(mnm), maxrow(mxr), offset(60), root(**nullptr**), SCREEN(**new char** \*[maxrow])  
 {  
 **for**(**int** i = 0; i < maxrow; ++i) SCREEN[i] = **new char**[120];  
 }  
  
Tree :: ~Tree(){**for**(**int** i = 0; i < maxrow; ++i) **delete** []SCREEN[i];  
 **delete** []SCREEN; **delete** root;}  
  
Node \*Tree::MakeNode(**int** depth)  
{  
 Node \*v = **nullptr**;  
 **int** Y = (depth < rand()%maxrow + 1) && (num <= 'z');  
 **if**(Y){  
 v = **new** Node;  
 v->left = MakeNode(depth + 1);  
 v->middle = MakeNode(depth + 1);  
 v->right = MakeNode(depth + 1);  
 v->d = num++;  
 }  
 **return** v;  
}  
  
**void** Tree::OutTree(){  
 clrscr();  
 OutNodes(root, 1, offset);  
 **for**(**int** i = 0; i < maxrow; ++i){  
 SCREEN[i][119] = 0;  
 std::cout << '\n' << SCREEN[i];  
 }  
 std::cout << '\n';  
}  
  
**void** Tree::clrscr() {  
 **for**(**int** i = 0; i < maxrow; ++i) memset(SCREEN[i],'.',120);  
}  
  
**void** Tree::OutNodes(Node \*v, **int** r, **int** c)  
{  
 **if**(r && c && (c < 120)) SCREEN[r - 1][c - 1] = v->d;  
 **if** (r < maxrow){  
 **if**(v->left) OutNodes(v->left,r + 1,c - (offset >> r) + 2);  
 **if**(v->middle) OutNodes(v->middle,r + 1, c);  
 **if**(v->right) OutNodes(v->right, r + 1, c + (offset >> r) - 2);  
 }  
}  
  
**void** Tree::task(Node \*ob){  
 **if**(ob != **nullptr**){  
 task(ob->left);  
 std::cout << ob->d << "\_";  
 task(ob->middle);  
 task(ob->right);  
 **if** (ob->left || ob->middle || ob->right) restask++;  
 }  
}