Министерство науки и образования РФ

Федеральное государственное автономное образовательное

учреждение высшего профессионального образования

«Санкт-Петербургский государственный электротехнический

университет «ЛЭТИ» им. В. И. Ульянова (Ленина)»

(СПбГЭТУ «ЛЭТИ»)

Факультет компьютерных технологий и информатики

Кафедра вычислительной техники

**Отчёт по заданию № 3**

**на тему: “Деревья”**

**по дисциплине**

**“Алгоритмы и структуры данных”**

**Вариант 34**

|  |  |
| --- | --- |
| Выполнили студенты гр.: | Дементьев Д.П., Кашапова О.С. |
| Проверил: | Колинько П.Г. |

Санкт-Петербург, 2020 г.

Оглавление

[Введение 3](#_Toc54989561)

[1. Задание 3](#_Toc54989562)

[2. Формализация задания 3](#_Toc54989563)

[3. Пример вывода дерева 4](#_Toc54989564)

[4. Временная сложность функций обхода. 5](#_Toc54989565)

[5. Результаты прогона программы. 5](#_Toc54989566)

[Вывод 7](#_Toc54989567)

[Список используемых источников 8](#_Toc54989568)

[Приложение 1 (Исходный текст программы) 9](#_Toc54989569)

## Введение

Исследование алгоритмов для работы с троичным деревом.

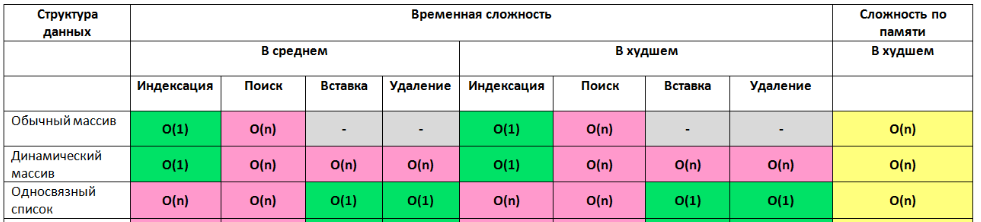
## 1. Задание

 Требуется создать троичное дерево с обратной разметкой узлов. Используя внутренний способ обхода, подсчитать количество правых листьев в исходном дереве.

## 2. Формализация задания

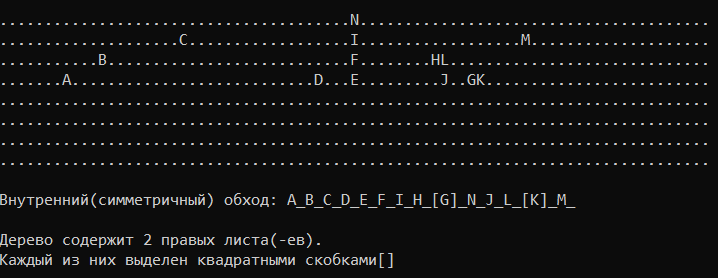
Для обоснования выбора в пользу разветвляющегося приведём таблицу «Временной сложности операций в различных структурах данных».

*Таблица 1. Временная сложность операций различных структур данных*



В нашей работе присутствует и «Поиск», и «Вставка», и «Удаление», поэтому рациональнее выбрать структуру данных, где временные сложности всех операций наименьшие - по таблице это определено список. Такая высокая скорость работы «Вставки» и «Удаления» обуславливается тем, что каждый раз нам не приходится перебирать все элементы, мы работаем со ссылками на конкретные элементы. В динамическом же массиве для этих же операций без аллокации не обойтись (выделение большей/меньшей памяти и последующая перезапись всех элементов), что и порождает сложность O(n).

## 3. Пример вывода дерева

Пример вывода дерева при выполнении программы:  
  


*Рисунок 1. Пример вывода дерева.*

Ввод обозначения узла символом осуществляется обратным образом. Это можно наглядно увидеть, если посмотреть, что самый левый лист обозначается буквой «A» (обозначение узлов начинается с буквы «A»). Чтобы при выводе троичного дерева не создавалось перекрытие элементов, они выводятся на отдалении в один символ от центра(например, H - правый сын I, а L - левый сын M). Так же ниже представлен обратный внутренний обход дерева. Сначала обходится левое поддерево, затем среднее поддерево, после обрабатывается корень, и в последнюю очередь обрабатывается правое поддерево.

## 4. Временная сложность функций обхода.

1. **Создание дерева.**

Сложность: Создание дерева - добавление в изначально пустое дерево *n* элементов имеет временную сложность O(*n*).

1. **Обход дерева**Сложность: О(*n*), так как это рекурсивный обход по n узлам дерева
2. **Обработка дерева**

Сложность: О(*n*), так как правые листья подсчитываются непосредственно в момент обхода дерева по n узлам.

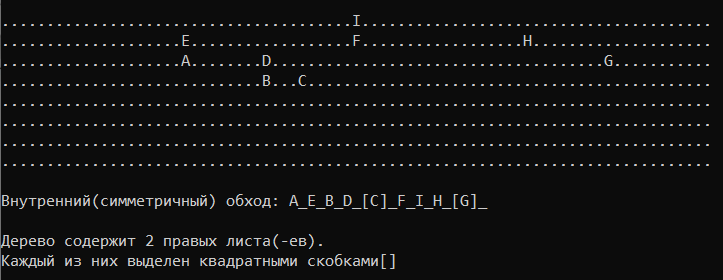
1. **Вывод дерева**

Сложность: О(*n*), так как мы просто обходим дерево.

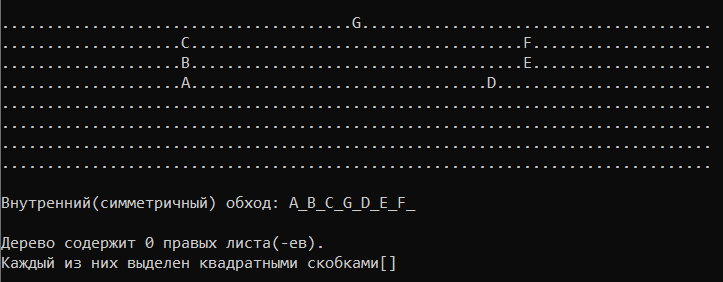
## 5. Результаты прогона программы.

При работе программы выводится количество правых листьев у исходного дерева.

Также выводится порядок обработки элементов дерева при внутреннем(симметричном) обходе.

На скриншотах ниже приведены различные ситуации прогона программы.  
  


*Рис 2. Результат выполнения.*

  
*Рис 3. В дереве отсутствуют правые листья.*

## Вывод

* + - * 1. Структура «Дерево» одна из базовых и необходимых инструментов в программировании. Это довольно простой вариант представления каких-либо связных между собой данных, поиска в этой базе, а что куда важнее для программирования - высокая скорость работы с этими базами. «Дерево» основанное на узлах и указателях мало того, что интуитивно понятно(узел - элемент дерева, указатель - связующая линия двух элементов), так еще и имеет большое преимущество по скорости работы перед «Деревом», основанном на массиве или векторе.

Выполнение данной лабораторной поможет в дальнейшей реализации структуры данных «Граф», так как это всеобъемлющее понятие, включающее в себя в том числе и «Дерево».

## Список используемых источников

1. Колинько П.Г. Пользовательские структуры данных / Методические указания по дисциплине «Алгоритмы и структуры данных» - Санкт-Петербург: СПбГЭТУ «ЛЭТИ», 2020.
2. Деревья. Способы обхода. ProgCPP // <https://prog-cpp.ru/data-tree/>
3. Знай сложность алгоритмов. Habr // https://habr.com/ru/post/188010/

## Приложение 1 (Исходный текст программы)

#include <iostream>

#include <cstring>

#include <time.h>

using namespace std;//класс "узел дерева"

class Node {

char d; //тег узла

Node \*lft;

Node \*mdl;

Node \*rgt;public:

Node():lft(nullptr), mdl(nullptr), rgt(nullptr) {} // конструктор узла по-умолчанию

~Node(){ //деструктор, уничтожающий поддерево

if (lft) delete lft;

if (mdl) delete mdl;

if (rgt) delete rgt; }

friend class Tree;

};

class Tree {

Node \*root; // указатель на корень дерева

char num, maxnum; // счётчик тегов и максимальный тег

int maxrow, offset; // максимальная глубина, смещение корня

char \*\*SCREEN; // память для вывода на экран

void clrscr(); // очистка рабочей памяти

Node\* MakeNode(int depth); // создание поддерева

void OutNodes(Node\* v, int r, int c); // выдача поддерева

//Tree(const Tree&); // фиктивный конструктор копии

//Tree(Tree&&); // копия с переносом(с++11)

//Tree operator = (const Tree&) const = delete; // присваивание

//Tree operator = (Tree&&) const = delete; // присваивание с переносом

void traversal(Node \*, int &, bool); // Продолжение симметричного обхода

// Перегрузкой traversal была решена проблема того, что члены класса не могут использоваться в качестве аргументов по умолчанию

// иначе была бы одна функций void traversal(Node \*node=root);public:

Tree(char num, char maxnum, int maxrow);

~Tree();

void MakeTree() {root = MakeNode(0);} // ввод - генерация дерева

bool exist() {return root != nullptr;} // проверка на пустоту дерева

int traversal(); // Начало симметричного обхода

void OutTree(); // выдача на экран};Tree:: Tree(char nm, char mnm, int mxr): // Конструктор, создающий матрицу символов, необх. для выдачи изображения на экран

num(nm), maxnum(mnm), maxrow(mxr), offset(40), root(nullptr), SCREEN(new char \*[maxrow])

{ for (int i=0;i<maxrow;i++) SCREEN[i] = new char[80]; }Tree:: ~Tree() { // Деструктор дерева уничтожает матрицу символов и запускает деструктор узла для корня

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()%4+1) && (num<=maxnum);

if (Y) {

v = new Node;

//v->d = num++; // Разметка в прямом порядке

v->lft = MakeNode(depth+1);

v->mdl = MakeNode(depth+1);

//v->d = num++; // Разметка во внутреннем порядке

v->rgt = MakeNode(depth+1);

v->d = num++; // Разметка в обратном порядке

}

return v;}

void Tree::OutTree(){

clrscr();

OutNodes(root, 1, offset);

for (int i=0; i<maxrow; i++)

{

SCREEN[i][79]=0;

cout << '\n' << SCREEN[i];

}

cout << '\n';}void Tree::clrscr() {

for (int i=0; i<maxrow; i++) memset(SCREEN[i], '.', 80); }

void Tree::OutNodes(Node \*v, int r, int c){

if (r&&c&&(c<80)) SCREEN[r-1][c-1] = v->d; //вывод метки

if (r<maxrow) {

if (v->lft) OutNodes(v->lft, r+1, c-(offset >> r)+1);

if (v->mdl) OutNodes(v->mdl, r+1, c);

if (v->rgt) OutNodes(v->rgt, r+1, c+(offset >> r)-1);

}}

int Tree::traversal(){

int rgt\_leaves=0;

Node \*node = root;

if (node)

{

if (node->lft) traversal(node->lft,rgt\_leaves, 0);

if (node->mdl) traversal(node->mdl,rgt\_leaves, 0);

cout << node->d << '\_';

if (node->rgt) traversal(node->rgt,rgt\_leaves, 1);

}

return rgt\_leaves;}

void Tree::traversal(Node \*node, int &rgt\_leaves, bool there\_is\_rgt){

if (there\_is\_rgt && !(node->lft) && !(node->mdl) && !(node->rgt))

{rgt\_leaves++;cout << '[' << node->d << "]\_";}

else {

if (node->lft) traversal(node->lft,rgt\_leaves, 0);

if (node->mdl) traversal(node->mdl,rgt\_leaves, 0);

cout << node->d << '\_';

if (node->rgt) traversal(node->rgt,rgt\_leaves, 1);

}}

int main(){

Tree root('A', 'Z', 8);

int rgt\_leaves;

srand(time(NULL));

setlocale(LC\_ALL, "Russian");

root.MakeTree();

if (root.exist())

{

root.OutTree();

cout << '\n' << "Внутренний(симметричный) обход: ";

rgt\_leaves = root.traversal();

cout << "\n\nДерево содержит " << rgt\_leaves << " правых листа(-ев). \nКаждый из них выделен квадратными скобками[]\n";

}

else cout << "Дерево пусто!";

return 0;}