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 26**

|  |  |
| --- | --- |
| Выполнили студент гр.9308: | Яловега Н.В. |
| Проверил: | Колинько П.Г. |

Санкт-Петербург, 2020 г.

**Оглавление**

[​ Введение 3](#__RefHeading___Toc2711_661619119)

[​ 1. Задание 3](#__RefHeading___Toc2713_661619119)

[​ 2. Формализация задания 3](#__RefHeading___Toc2715_661619119)

[​ 3. Примеры работы программы 5](#__RefHeading___Toc2717_661619119)

[​ 4. Оценки временной сложности 7](#__RefHeading___Toc2723_661619119)

[​ Вывод 8](#__RefHeading___Toc2725_661619119)

[​ Список используемых источников 9](#__RefHeading___Toc2727_661619119)

[​ Приложение 1 (Исходный текст программы) 10](#__RefHeading___Toc2729_661619119)

## Введение

Исследование алгоритмов для работы с троичным деревом.

## 1. Задание

Требуется создать троичное дерево с симметричной разметкой узлов. Используя способ обхода в глубину, подсчитать количество листьев не на самом нижнем уровне исходного дерева.

## 2. Формализация задания

Было принято представлять дерево в памяти ЭВМ с помощью разветвляющегося списка. Узлы дерева — объекты, связи между которыми осуществляется через указатели.

Данный метод предствления дерева был выбран, потому что он имеет большое количество преимуществ по сравнению с массивом.

Можно представить дерево в виде одномерного массива следующим образом: вершины можно пронумеровать слева направо последовательно по уровням и использовать эти номера в качестве индексов в одномерном массиве. Такой способ достаточно эффективен для хранения полных деревьев, т.е тех, которые имеют всех потомков. Однако далеко не все деревья являются полными, поэтому возникает проблема чрезмерного использования памяти, связанная с отсутствующими узлами, которым должны соответствовать неиспользуемые элементы массива. Т.е чем менее полным является дерево, тем менее рационально используется память. Кроме того, существенным недостатком рассмотренного способа представления дерева является то, что структура данных предполагает последовательное хранение элементов в памяти. Если в программе потребуется добавить больше узлов, то потребуется создание массива большего размера и копирование туда всех данных из старого массива. Если использовать для представления дерева массив, то это обеспечит линейное время удаления и вставок, логарифмическое время выполения двоичного поиска (причем массив должен быть отсортирован, а учитывая то, что при хранении дерева предложенным способом позиция важна для получения потомков, хранить единожды отсортированный массив не получится (а если хранить, то придется тратить большое количество памяти)) или линейное время выполнения линейного поиска.

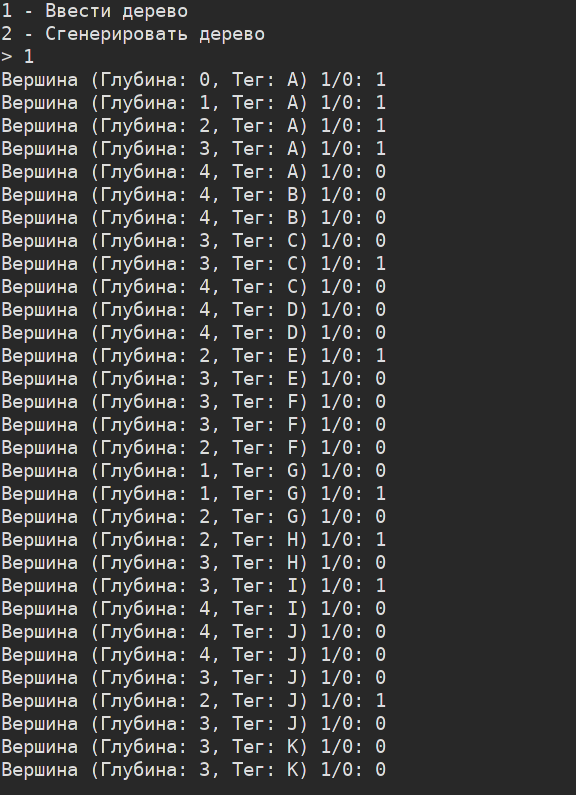
Метод представления дерева в виде разветвляющегося списка лишен недостатков массива. Связные списки хранят значения в узлах, разбросанных по памяти и связанных между собой указателями. За счет этого не используется лишняя память для хранения несуществующих узлов, для добавления новых элементов требуется выделять столько памяти, сколько необходимо для данной структуры. В неотсортированных односвязных списках для операций вставки и удаления требуется время O(1), а для операций поиска — линейное время в худшем случае.

## 3. Примеры работы программы

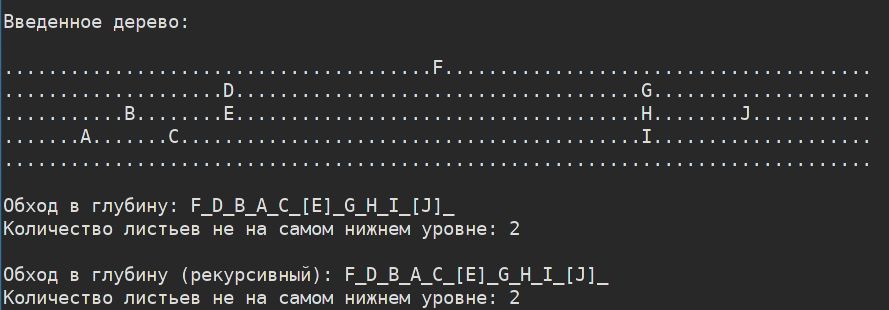
1) Ручной ввод дерева

Для ввода дерева использовалась следующая последовательность:

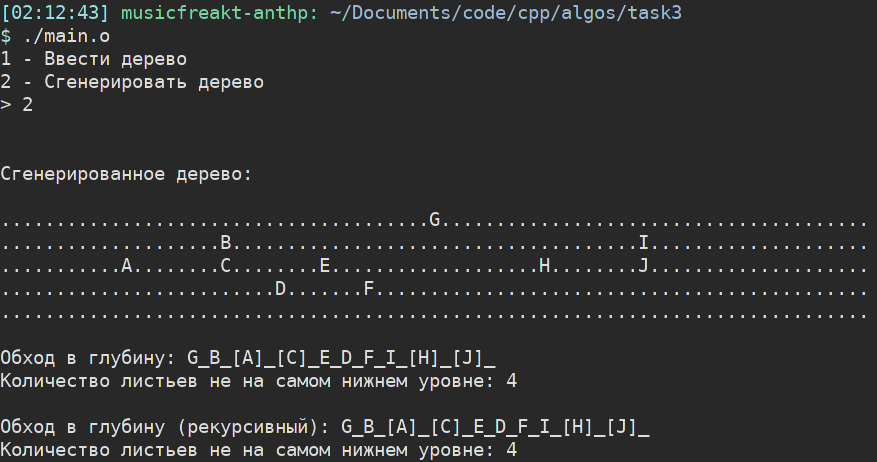
1111000010001000001010100001000



Дерево, которое получилось в результате:



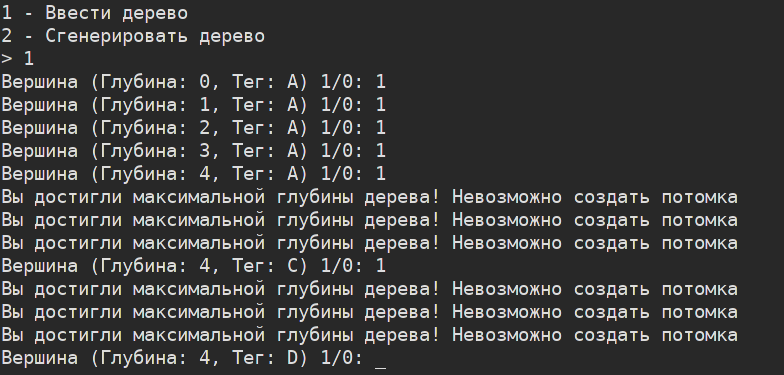
2) Генерация случайного дерева

****

Для разметки узлов используется симметричный обход. Это можно наглядно увидеть, ведь номер любого узла, больше номера в левом поддереве и меньше любого номера в среднем и правом поддереве.Так же ниже представлена последовательность вершин, используемых при обходе дерева в глубину. В квадратных скобках отмечены вершины, удовлетворяющие условию задачи.

При работе программы выводится количество листьев у исходного дерева, которые находятся не на самом нижнем уровне.

Программа не даст возможности ввести потомки узла, если превышено ограничение на максимально допустимый уровень дерева.



## 4. Оценки временной сложности

**1) Создание дерева.**

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

**2) Обработка дерева (обход в глубину)**

Сложность: O(n), где n — количество вершин дерева, т.к в стек добавляем всех сыновей узла, следовательно посетим каждую вершину один раз.

**3) Вывод дерева**

Сложность: O(n), где n — количество вершин дерева, т.к рекурсивно обходим дерево.

## Вывод

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.

* 1. 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. 1. Колинько П.Г. Пользовательские структуры данных / Методические указания по дисциплине «Алгоритмы и структуры данных» - Санкт-Петербург: СПбГЭТУ «ЛЭТИ», 2020.

## Приложение 1 (Исходный текст программы)

#include <iostream>

#include <cstring>

#include <ctime>

using namespace std;

template<class Item> class STACK

{

Item \*S;

int t;

public:

STACK(int maxt):S(new Item[maxt]), t(0){}

int empty() const{return t==0;}

void push(Item item) {S[t++]=item;}

Item pop(){return(t? S[--t]:0);}

~STACK(){delete [] S;}

};

class Node

{

char tag;

Node \*left;

Node \*middle;

Node \*right;

int depth;

public:

Node():left(nullptr), middle(nullptr), right(nullptr) {}

~Node()

{

if (left) delete left;

if (middle) delete middle;

if (right) delete right;

}

bool is\_leaf()

{

if (!left && !middle && !right)

return true;

return false;

}

friend class Tree;

};

class Tree {

Node \*root;

char num, maxnum;

int maxrow, offset;

int max\_depth;

char \*\*SCREEN;

void clrscr();

Node\* MakeNode(int, char);

void OutNodes(Node\*, int, int);

void RDFS(Node\*, int\*);

public:

Tree(char, char, int);

~Tree();

void MakeTree() {root = MakeNode(0, 0);}

void InpTree() {root = MakeNode(0, 1);}

bool exist() {return root != nullptr;}

void OutTree();

int DFS();

void RDFS(int\*);

};

Tree::Tree(char nm, char mnm, int mxr):

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()

{

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

delete[] SCREEN[i];

delete[] SCREEN;

delete root;

}

Node \*Tree::MakeNode(int depth, char flag)

{

Node \*v = nullptr;

int Y;

if (flag)

{

if (depth < maxrow)

{

cout << "Вершина (Глубина: " << depth << ", Тег: " << num << ") 1/0: ";

cin >> Y;

}

else

{

cout << "Вы достигли максимальной глубины дерева! Невозможно создать потомка\n";

Y = 0;

}

}

else

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

if (Y)

{

v = new Node;

v->depth = depth;

v->left = MakeNode(depth+1, flag);

v->tag = num++;

v->middle = MakeNode(depth+1, flag);

v->right = MakeNode(depth+1, flag);

}

if(depth-1 > max\_depth)

max\_depth = depth-1;

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->tag;

if (r < maxrow) {

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

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

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

}

}

int Tree::DFS()

{

const int MaxS = 20;

int count = 0;

STACK <Node \*> stack(MaxS);

stack.push(root);

while (!stack.empty())

{

Node \*v = stack.pop();

if (v->depth != max\_depth && v->is\_leaf())

{

count++;

cout << '[' << v->tag << "]\_";

}

else

cout << v->tag << '\_';

if (v->right)

stack.push(v->right);

if (v->middle)

stack.push(v->middle);

if (v->left)

stack.push(v->left);

}

return count;

}

void Tree::RDFS(int\* c)

{

RDFS(root, c);

}

void Tree::RDFS(Node\* v, int\* count)

{

if (v)

{

if (v->depth != max\_depth && v->is\_leaf())

{

(\*count)++;

cout << '[' << v->tag << "]\_";

}

else

cout << v->tag << '\_';

RDFS(v->left, count);

RDFS(v->middle, count);

RDFS(v->right, count);

}

}

int main()

{

srand(time(NULL));

Tree root('A', 'Z', 5);

int menu\_item,

num\_leaves;

do

{

cin.sync();

cout << "1 - Ввести дерево\n";

cout << "2 - Сгенерировать дерево\n";

cout << "> ";

cin >> menu\_item;

switch(menu\_item)

{

case 1:

root.InpTree();

cout << "\n\nВведенное дерево:\n";

break;

case 2:

root.MakeTree();

cout << "\n\nСгенерированное дерево:\n";

break;

default:

cout << "Такого пункта не существует, повторите ввод!\n";

}

if(cin.fail())

{

cin.clear();

cin.ignore(32767,'\n');

}

}

while(menu\_item > 2 || menu\_item < 1 || cin.fail());

if (root.exist())

{

root.OutTree();

cout<< '\n' << "Обход в глубину: ";

num\_leaves = root.DFS();

cout << "\nКоличество листьев не на самом нижнем уровне: " << num\_leaves << '\n';

num\_leaves = 0;

cout<< '\n' << "Обход в глубину (рекурсивный): ";

root.RDFS(&num\_leaves);

cout << "\nКоличество листьев не на самом нижнем уровне: " << num\_leaves << '\n';

}

else cout << "Дерево пусто!\n";

return 0;

}