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

**Пояснительная записка к курсовой работе по теме «Графы»**

**по дисциплине**

**«Алгоритмы и структуры данных»**

**Вариант 34**

Выполнили студенты гр.9308: Дементьев Д.П., Кашапова О.С.

Проверил: Колинько П.Г.

Санкт-Петербург, 2020 г.

**Оглавление**

[Введение 3](#_bookmark0)

1. [Задание 3](#_bookmark1)
2. [Формализация задания 3](#_bookmark2)
3. [Обоснование выбора способа представления графа в памяти ЭВМ 4](#_bookmark3)
4. [Временная сложность функций 5](#_bookmark4)
5. [Контрольные примеры 6](#_bookmark5)
6. [Тестирование программы 8](#_bookmark6)

[Вывод 11](#_bookmark7)

[Список используемых источников 12](#_bookmark8)

[Приложение 1 (Исходный текст программы) 13](#_bookmark9)

# Введение

Исследование алгоритмов, реализуемых с помощью графов.

# Задание

Проверка на изоморфизм произвольных корневых деревьев.

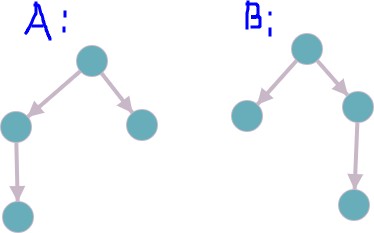
# Формализация задания

Дерево называется *корневым*, если оно ориентированно, и из какой-то вершины (называемой *корнем*) можно попасть во все остальные.

*Изоморфизм* графов G1 и G2 — это биекция (взаимно-однозначное соответствие) f из V(G1) в V(G2) такая, что ребро (uv) в графе G1 существует тогда и только тогда, когда в графе G2 существует ребро (f(u)f(v)). Иными словами, изоморфизм — это способ перенумеровать вершины графа G1 так, чтобы получился в точности граф G2. Соответственно, два графа называются изоморфными, если существует изоморфизм.

В задании рассматривается частный случай, когда оба графа являются деревьями. Понятно, что если у них разный размер, то они неизоморфны. Поэтому будем рассматривать корневые деревья с одинаковым количеством вершин .

Два дерева называются изоморфными, если они имеют одну структуру. К примеру, эти два дерева будут изоморфны друг другу.



*Рисунок 1: Пример изоморфных деревьев*

Пусть каждое дерево имеет свой код, который создается следующим образом: если вершина является листом - ей присваивается код «10», а если нет, то ее кодом будет являться конкатенация кодов ее детей и приписанные в начало «1» и «0» в конец. Таким образом, пусть у вершины k есть два ребенка с кодами str1 и str2 соответственно, тогда код вершины k будет выглядеть следующим образом: k = 1 + str1 + str2 + 0. Получается код всего дерева будет содержаться в его корне. Можно заметить, что у двух изоморфных деревьев будет одинаковый код, если его отсортировать. Отсортированы кодом назовем код, полученный поэтапной сортировкой кодов, расположенных от листов к корню. Т.е. к примеру, мы имеем 3 кода: 1100, 10 и 110100, тогда в код текущей вершины сначала запишется 10, затем 1100 и уже после 110100 и мы получим: code = 1 + 10 + 1100 + 110100 + 0.

Для обхода деревьев и получения их кодов будем использовать поиск в глубину. Как только мы приходим в лист, присваиваем ему код «10», и возвращаем этот код как результат функции. Тогда для k — ой вершины мы получим, после обхода n ее детей — n кодов, которые нужно прост отсортировать и приписать «1» в начало и

«0» в конец. Таким образом мы получили коды всех деревьев, которые осталось только сравнить.

Данный алгоритм носит название AHU, в честь трёх ученых Aho, Hopcroft и Ullman. Работает за время O(n\*logn), так как обход в глубину O(n), и быстрая сортировка O(n\*logn).

# Обоснование выбора способа представления графа в памяти ЭВМ

Граф в памяти представлен в виде списка смежности. На это есть свои причины:

Во-первых, алгоритм основывается на обходе графа. Для такого представления обход графа имеет линейную сложность O(n). Если же использовать матрицу смежности, то сложность будет O(n^2).

Во-вторых, данный способ представления удобен для графов, в которых количество ребер не очень велико. Максимальное количество вершин дерева, которое может быть в программе — 26, количество букв латинского алфавита. Тогда максимальное количество ребер может будет m = n-1 = 25. Получается, что для хранения такого дерева понадобится n+m памяти 26+25 = 51, что меньше, чем при использовании матрицы смежности, для которой необходимо n^2 памяти 26^2 = 676.

# Временная сложность функций

1. **Ввод графа**

Сложность: O(n+m), так как заполняем списки смежности

## Генерация случайного графа.

Сложность: O(n^2), так как для генерации графа матрица смежности заполняется случайными 1 и 0, после чего она преобразуется в списки смежности.

## Построение кода:

Сложность: O(n\*logn), так как обход в глубину O(n), и быстрая сортировка O(n\*logn).

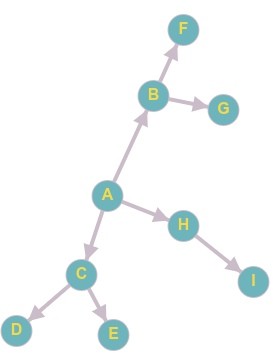
# Контрольные примеры

В таблице 1 представлены контрольные примеры, которые использовались для тестирования программы.

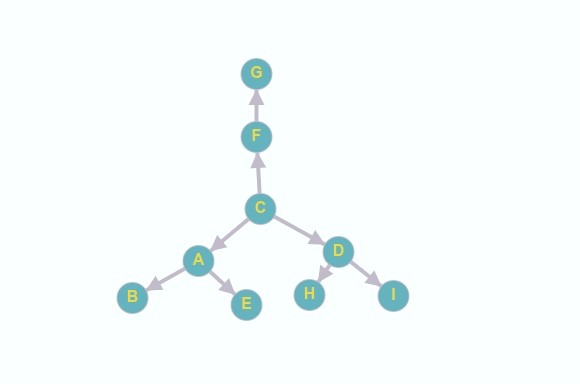
Таблица 1: Контрольные примеры

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Входные данные | | | | | Результат |
| Кол-во вершин | Корень | Списки смежности | Корень | Списки смежности |
| 1 | 9 | a | a: -bc h-  b: -----fg--  c: ---de----  d: ---------  e: ---------  f: ---------  g: ---------  h i  i: --------- | c | a: -b--e----  b: ---------  c: a--d-f---  d hi  e: ---------  f: ------g--  g: ---------  h: ---------  i: --------- | Деревья изоморфны |
| 2 | 9 | f | a: --c h-  b: a-----g--  c: ---de----  d: ---------  e: ---------  f: -b-------  g: ---------  h i  i: --------- | g | a: -b--e----  b: ---------  c: a--d-----  d hi  e: ---------  f: --c------  g: -----f---  h: ---------  i: --------- | Деревья не изоморфны |
| 3 | 4 | a | a: -bc-  b: ---d  c: ----  d: ---- | a | a: -bc- b: ----  c d  d: ---- | Деревья изоморфны |
| 4 | 3 | a | a: -bc b: ---  c: --- | a | a: -b-  b: --c  c: --- | Деревья не изоморфны |
| 5 | 6 | a | a: -b-d--  b: --c-e-  c: ------  d f  e: ------  f: ------ | a | a: -b-de-  b: --c--f  c: ------  d: ------  e: ------  f: ------ | Деревья не изоморфны |

На рисунках 2 и 3 представлено графическое представление исходного графа и результата для примера 1. Данный пример используется в программе, как демонстрационный.



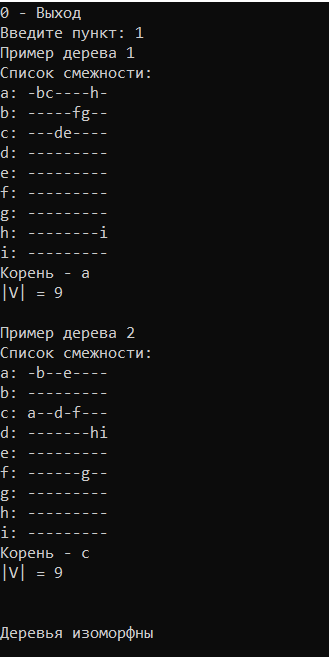
*Рисунок 2: Визуализация первого дерева из примера 1*

**

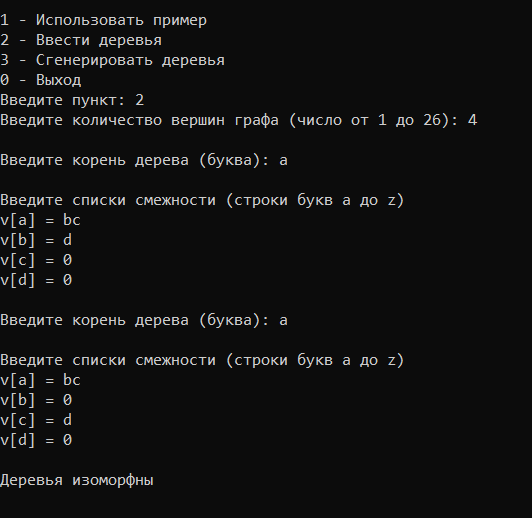
*Рисунок 3: Визуализация второго дерева из примера 1*

# Тестирование программы

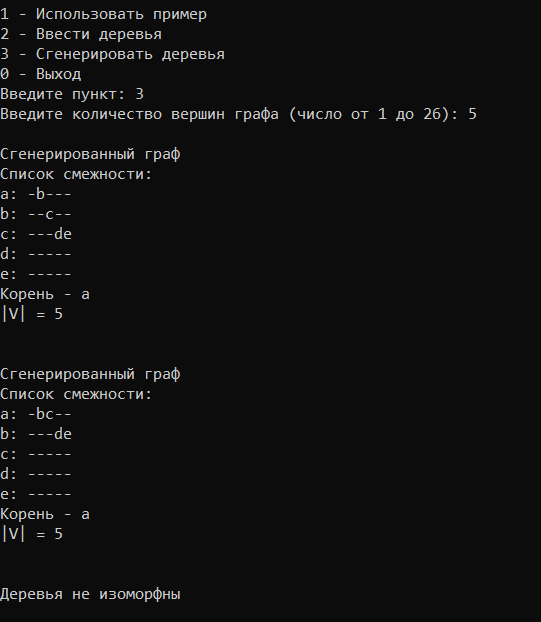
На скриншотах ниже представлены результаты прогона программы для различных тестов из таблицы 1.



*Рисунок 4: Контрольный пример работы*



*Рисунок 5: Пример ввода произвольного дерева*

**

*Рисунок 6: Пример генерации деревьев*

# Вывод

В ходе выполнения работы был исследован алгоритм проверки изоморфизма корневых деревьев. При тестировании программы с различными данными ошибки не были обнаружены.

На практике необходимость проверки изоморфизма деревьев возникает при решении задач хемоинформатики, математической или компьютерной химии, автоматизации проектирования электронных схем (верификация различных представлений электронной схемы) и оптимизации программ (выделение общих подвыражений).

# Список используемых источников

* + 1. Колинько П.Г. Пользовательские структуры данных / Методические указания по дисциплине «Алгоритмы и структуры данных» - Санкт-Петербург: СПбГЭТУ «ЛЭТИ», 2020
    2. Гирш Э.А. «Информатика». Алгоритмы на графах (II). URL: https://logic.pdmi.ras.ru/~hirsch/students/cs-math/lecture6.pdf

Дата обращения 23.12.2020

* + 1. METANIT.COM Сайт о программировании// Итераторы. URL: <https://metanit.com/> Дата обращения: 24.12.2020

# Приложение 1 (Исходный текст программы)

# Tree.h (структура данных Корневого дерева)

#define TREE\_H

#include <vector>

#include <string>

#include <algorithm>

#include <iostream>

using namespace std;

class Tree

{

int n; // количество вершин

int root; // корень дерева

vector <vector<int> > adj; // список смежности

string dfs(int, vector<bool>&); // обход дерева в глубину

string dfs\_exec(); // оболочка обхода в глубину

public:

Tree(); // пример дерева

Tree(char); // пример другого дерева (изоморфного 1)

Tree(int); // ввод дерева

Tree(int, char); // генерирование случайного дерева

bool is\_isomorph(Tree\*); // проверка деревьев на изоморфность

};

#endif

# 

**Tree.cpp (описание методов)**

#include "tree.h"

// Возвращает символ алфавита c'ый по порядку

char name(char c) {return c+'a';}

// Сравнение строк (для сортировки)

bool cmp(const string &lhs, const string &rhs)

{

return lhs < rhs;

}

// Заготовленный пример дерева

Tree :: Tree()

{

root = 0;

n = 9;

adj.resize(n);

adj[0].push\_back(1);

adj[0].push\_back(2);

adj[0].push\_back(7);

adj[1].push\_back(5);

adj[1].push\_back(6);

adj[2].push\_back(3);

adj[2].push\_back(4);

adj[7].push\_back(8);

cout << "Пример дерева 1\nСписок смежности:\n";

cout << "a: -bc----h-\n";

cout << "b: -----fg--\n";

cout << "c: ---de----\n";

cout << "d: ---------\n";

cout << "e: ---------\n";

cout << "f: ---------\n";

cout << "g: ---------\n";

cout << "h: --------i\n";

cout << "i: ---------\n";

cout << "Корень - " << name(root) << "\n";

cout << "|V| = 9\n\n";

}

// Пример второго дерева. (Изоморфен 1)

Tree :: Tree(char c)

{

root = 2;

n = 9;

adj.resize(n);

adj[2].push\_back(0);

adj[2].push\_back(5);

adj[2].push\_back(3);

adj[0].push\_back(1);

adj[0].push\_back(4);

adj[3].push\_back(7);

adj[3].push\_back(8);

adj[5].push\_back(6);

cout << "Пример дерева 2\nСписок смежности:\n";

cout << "a: -b--e----\n";

cout << "b: ---------\n";

cout << "c: a--d-f---\n";

cout << "d: -------hi\n";

cout << "e: ---------\n";

cout << "f: ------g--\n";

cout << "g: ---------\n";

cout << "h: ---------\n";

cout << "i: ---------\n";

cout << "Корень - " << name(root) << "\n";

cout << "|V| = 9\n\n";

}

// Ввод дерева списками смежности

Tree :: Tree(int maxV): n(maxV)

{

string s;

char r;

cout << "\nВведите корень дерева (буква): ";

cin >> r;

root = (int)(tolower(r) - 'a');

adj.resize(n);

cout << "\nВведите списки смежности (строки букв a до z)\n";

for (int k = 0; k < n; ++k)

{

cout << "v[" << name(k) << "] = ";

cin >> s;

for (auto u: s)

if (isalpha(u))

{

int j = tolower(u) - 'a';

adj[k].push\_back(j); // орграф

}

}

}

// Генерация случайного дерева (для простоты корнем будет нулевая вершина)

Tree :: Tree(int maxV, char c): n(maxV), root(0)

{

int G[n][n];

// обнуление матрицы

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

for (int j = 0; j < n; ++j)

G[i][j] = 0;

// генерация случайной матрицы смежности

for (int i = n - 1; i > 0; i--)

G[rand() % i][i] = 1;

cout << "\nСгенерированный граф\nСписок смежности:";

// заполнение списка смежности

adj.resize(n);

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

{

cout << '\n' << name(i) << ": ";

for (int j = 0; j < n; ++j)

if (G[i][j])

{

adj[i].push\_back(j);

cout << name(j);

}

else cout << '-';

}

cout << "\nКорень - " << name(root) << "\n";

cout << "|V| = " << n << "\n\n";

}

// Проверка деревьев на изоморфность. Сравнение кодов деревьев.

// Возвращает истину, если графы изоморфны

bool Tree :: is\_isomorph(Tree \*other)

{

// cout << dfs\_exec() << "\n" << other.dfs\_exec() << "\n";

if (dfs\_exec() == other->dfs\_exec())

return true;

else

return false;

}

// Функция оболочка для обхода дерева в глубину

// Возвращает код дерева

string Tree :: dfs\_exec()

{

string code;

vector<bool> used;

used.resize(n);

code = dfs(root, used);

return code;

}

// Обход дерева в глубину с составлением кода дерева.

// Возвращает код дерева

string Tree :: dfs(int node, vector<bool> &used)

{

used[node] = true;

vector<string> tmp;

string str;

if (adj[node].empty()) // если вершина является листом

return "10";

for (vector<int>::iterator i = adj[node].begin(); i != adj[node].end(); ++i)

if (!used[\*i])

tmp.push\_back(dfs(\*i, used));

sort(tmp.begin(), tmp.end(), cmp);

for (int j = 0; j < tmp.size(); ++j)

str += tmp[j];

return '1' + str + '0';

}

# Main.cpp

#include <iostream>

#include <cstdlib>

#include <ctime>

#include "tree.h"

using namespace std;

const int MAXV = 26;

void example(); // заготовленный контрольный пример

void input(); // ввод дерева

void generate(); // генерация деревьев

short get\_num\_nodes(); // ввод количество вершин графа

int main()

{

srand(time(NULL));

setlocale(LC\_ALL, "Russian");

int menu;

do

{

cout << "1 - Использовать пример\n";

cout << "2 - Ввести деревья\n";

cout << "3 - Сгенерировать деревья\n";

cout << "0 - Выход\n";

cout << "Введите пункт: ";

cin >> menu;

switch(menu)

{

case 1:

example();

break;

case 2:

input();

break;

case 3:

generate();

break;

case 0:

cout << "До свидания!\n";

break;

default:

cout << "Такого пункта не существует!\n";

}

}

while(menu);

return 0;

}

void example()

{

Tree t1;

Tree t2('0');

if (t1.is\_isomorph(&t2))

cout << "\nДеревья изоморфны\n\n\n";

else

cout << "\nДеревья не изоморфны\n\n\n";

}

void input()

{

int n = get\_num\_nodes();

Tree t1(n), t2(n);

if (t1.is\_isomorph(&t2))

cout << "\nДеревья изоморфны\n\n\n";

else

cout << "\nДеревья не изоморфны\n\n\n";

}

void generate()

{

int n = get\_num\_nodes();

Tree t1(n, '0'), t2(n, '0');

if (t1.is\_isomorph(&t2))

cout << "\nДеревья изоморфны\n\n\n";

else

cout << "\nДеревья не изоморфны\n\n\n";

}

short get\_num\_nodes()

{

int n;

do

{

cout << "Введите количество вершин графа (число от 1 до " << MAXV << "): ";

cin >> n;

if (cin.fail())

{

cin.clear();

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

cout << "\nНекорректный ввод. Попробуйте еще раз\n";

}

else

{

if (n > MAXV)

cout << "\nЧисло вершин не должно превышать " << MAXV << '\n';

if (n < 1)

cout << "\nЧисло вершин не должно быть отрицательным или 0\n";

}

} while(cin.fail() || n < 1 || n > MAXV);

return n;

}