**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

**КУРСОВАЯ РАБОТА**

**по дисциплине «Построение и анализ алгоритмов».**

**Тема: «Алгоритмы на графах. Поиск мостов. Исследование»**

Вариант 5.

|  |  |  |
| --- | --- | --- |
| Студент(ка) гр. 5383 |  | Допира В. Е. |
| Преподаватель |  | Фирсов М. А. |

Санкт-Петербург

2017

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студентка Допира В. Е. | | |
| Группа 5383 | | |
| Тема работы: «Алгоритмы на графах. Поиск мостов. Исследование» | | |
| Исходные данные: количество вершин в графе типа int | | |
| Содержание пояснительной записки:  «Содержание»  «Введение»  «Спецификация программы»  «Алгоритм и структуры данных»  «Исследование»  «Тестирование»  «Заключение»  «Список использованных источников»  «Приложение» | | |
| Предполагаемый объем пояснительной записки:  20 страница. | | |
| Дата выдачи задания: | | |
| Дата сдачи реферата: | | |
| Дата защиты реферата: | | |
| Студент |  | Допира В.Е. |
| Преподаватель |  | Фирсов М. А. |

**Содержание**

[ВВЕДЕНИЕ](#_Toc467613480) 5

[1. СПЕЦИФИКАЦИЯ ПРОГРАММЫ](#_Toc467613481) 7

[1.1 Входные данные](#_Toc467613482) 7

[1.2 Выходные данные](#_Toc467613484) 7

[1.3 Описание интерфейса пользователя.](#_Toc467613486) 7

[2. АЛГОРИТМ И СТРУКТУРЫ ДАННЫХ](#_Toc467613488) 8

[2.1 Описание алгоритма](#_Toc467613489) 8

[2.2 Функции, использованные в программе](#_Toc467613490) 9

[3. ТЕСТИРОВАНИЕ.](#_Toc467613493) 11

[4. ИССЛЕДОВАНИЕ.](#_Toc467613493) 12

[ЗАКЛЮЧЕНИЕ И ВЫВОД](#_Toc467613497) 15

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ](#_Toc467613497) 16

[ПРИЛОЖЕНИЕ. Текст программы](#_Toc467613497) 17

**Аннотация**

Курсовая работа генерирует неориентированный граф и находит в нем мосты. Граф задается матрицей смежности его вершин. Генерация осуществляется с учетом насыщенности графа ребрами. Результатом работы программы являются мосты графа. Разработан проект на языке программирования С++ с использованием поиска в глубину.

**Summary**

Course work generates an undirected graph and finds bridges in it. The graph is given by the adjacency matrix of its vertices. Generation is carried out taking into account the saturation of the graph by the edges. The result of the program is the graph's bridges. The project is developed in C ++ programming language using depth-based search.

**введение**

Мосты графа. Мост в связном неориентированном графе - это ребро, удаление которого делает граф несвязным.

**Вариант 5**

**Цель:** реализация и экспериментальное исследование алгоритма поиска мостов в глубину в неориентированном графе.

**Задача:** Реализовать алгоритм нахождения мостов графа (модификацией поиска в глубину). Провести экспериментальный анализ его эффективности на графах. Продумать и предложить механизм генерации необходимых для эксперимента тестовых данных.

**Формальная постановка задачи**

Задано количество вершин в неориентированном графе. Например, 4. Далее задаем плотность матрицы смежности. Допустим, плотность графа 50%. Считаем количество ребер в графе по формуле: . В нашем случаае: = *8.* Матрица смежности имеет вид:

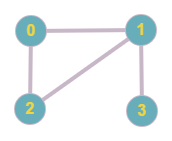
0 1 1 0

1 0 1 1

1 1 0 0

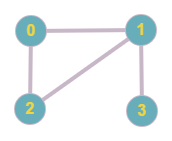
0 1 0 0

Граф имеет вид:



Рассмотрим вершину 0. Из нее мы можем попасть в вершину 1 и 2. Пробуем найти обратные ребра в 0 из других вершин. Они есть, значит, ребра не являются мостами. Из 1 можно попасть в 0, 2, 3. В 0 и 2 есть еще ребра. А в 3 – нет, значит, это мост. Мост: (1, 3). Идем из 2. Больше мостов не находим. И из 3 мост (1,3) уже найден.

Мост: (1 , 3)



**1. Спецификация программы**

**1.1 Входные данные**

**Структуры данных.**

Неориентированный граф задается матрицей смежности.

const int MAXN = 1933;

int mx[MAXN][MAXN];

А потом матрица переводится в список смежности.

vector<int> g[MAXN];

Мосты записываются в вектор.

vector< pair<int, int> > bridges;

**1.2 Выходные данные**

Программа выводит на экран и записывает в файл мосты графа.

**1.3 Описание интерфейса пользователя**

Пользователь выбирает, как будет задаваться матрица: генерироваться программой или вводиться вручную. Далее, как осуществляться ввод: из консоли или из файла «input.txt». Пользователь вводит количество вершин. При генерации следует ввести плотность графа, с учетом которой создается матрица смежности. Программа проверяет корректность введенных данных: типа ввода данных (0 или 1), типа генерации матрицы смежности (0 или 1), количества вершин графа (≤1933), плотность графа (≤100%). При обнаружении некорректности, выводятся соответствующие сообщения. Программа выводит в консоль и записывает в файл «output.txt» результат работы программы.

**2. Алгоритм и структуры данных**

**2.1 Описание алгоритма**

Программа выполняет следующую последовательность действий:

Введенная пользователем матрица смежности конвертируется в список смежности с помощью функции void convertMatrixToAdjacenciesList().

Поиск мостов осуществляется с помощью обхода в глубину. Функция поиска мостов find\_bridges() вызывает для каждой вершины рекурсивно, пока не посетит все вершины, функцию обхода в глубину dfs(), которая описана ниже.

В функции обхода в глубину, просматривая все рёбра из вершины v. Тогда, если текущее ребро (v, to) таково, что из вершины to и из любого её потомка в дереве обхода в глубину нет обратного ребра в вершину v или какого-либо её предка, то это ребро является мостом. В противном случае оно мостом не является. Вводятся времена входа в вершину, вычисляемыми [алгоритмом поиска в глубину](http://www.e-maxx-ru.1gb.ru/algo/dfs).

Итак, пусть tin[v] — это время захода поиска в глубину в вершину v. Теперь введём массив fup[v], который и позволит нам отвечать на вышеописанные запросы. Время fup[v] равно минимуму из времени захода в саму вершину tin[v], времён захода в каждую вершину p, являющуюся концом некоторого обратного ребра (v,p), а также из всех значений fup[to] для каждой вершины to, являющейся непосредственным сыном v в дереве поиска:

Тогда, из вершины v или её потомка есть обратное ребро в её предка тогда и только тогда, когда найдётся такой сын to, что fup[to] ≤ tin[v]. Если fup[to] == tin[v], то это означает, что найдётся обратное ребро, приходящее точно в v; если же fup[to] < tin[v], то это означает наличие обратного ребра в какого-либо предка вершины v. А если для текущего ребра (v, to), принадлежащего дереву поиска выполняется fup[to] > tin[v], то это ребро является мостом; в противном случае оно мостом не является.

Различаются три случая: когда идём по ребру дерева поиска в глубину, когда идём по обратному ребру, и когда пытаемся пойти по ребру дерева в обратную сторону. Это, соответственно, случаи:

* used[to] = false — критерий ребра дерева поиска;
* used[to] — true & to != parent критерий обратного ребра;
* to = parent — критерий прохода по ребру дерева поиска в обратную сторону.

Таким образом, для реализации этих критериев надо передавать в функцию поиска в глубину текущую вершину и вершину-предка.

**2.2 Функции, используемые в программе**

void dfs (int v, int p = -1)

Назначение: поиск в глубину

Параметры: int v – текущая вершина

int p – родительская вершина

void find\_bridges()

Назначение: поиск мостов

void isBridge(int source, int destination)

Назначение: проверка на мост

Параметры: int source – первая вершина

int destination – вторая вершина

void inputData()

Назначение: ввод данных

void printGraphState()

Назначение: печать графа

void randomGenerator()

Назначение: рандомная генерация

void convertMatrixToAdjacenciesList()

Назначение: конвертация из матрицы в список смежности

**3. Тестирование**

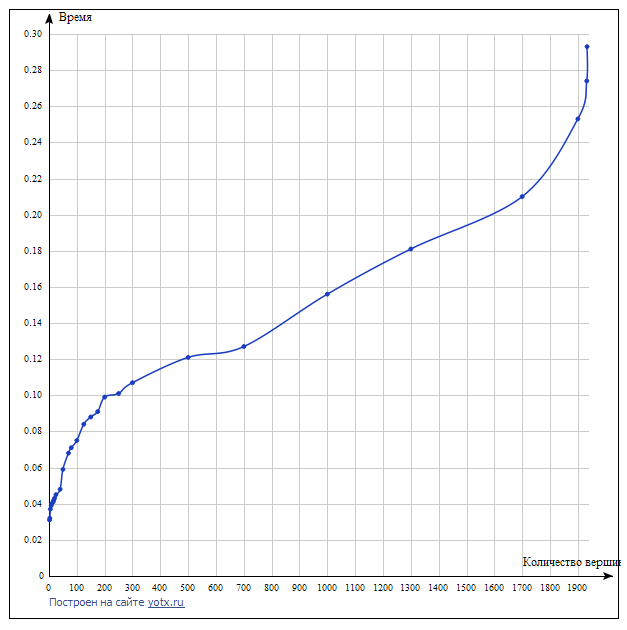
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Входные данные** | | | | **Результат** | |
| **№** | **Число вершин** | **Плот-**  **ность** | **Матрица смежности** | Ожидаемые ответы | **Полученные ответы** |
| 1 | 1 | 101 |  | density can not have this value! | density can not have this value! |
| 2 | 3 | 45 | 0 0 1  0 0 1  1 1 0 | 2 1  0 2 | 2 1  0 2 |
| 3 | 3 | 70 | 0 1 0  1 1 1  0 1 1 | 1 2  0 1 | 1 2  0 1 |
| 4 | 4 | 50 | 1 1 1 1  1 0 0 0  1 0 1 0  1 0 0 0 | 0 1  0 2  0 3 | 0 1  0 2  0 3 |
| 5 | 4 | 50 | 0 1 1 0  1 0 1 1  1 1 0 0  0 1 0 0 | 1 3 | 1 3 |
| 6 | 2 | 75 | 0 1  1 1 | 0 1 | 0 1 |
| 7 | 5 | 10 | 1 0 1 1 0  0 0 1 0 0  1 1 0 0 0  1 0 0 1 1  0 0 0 1 0 | 2 1  0 2  3 4  0 3 | 2 1  0 2  3 4  0 3 |

**4. Исследование**

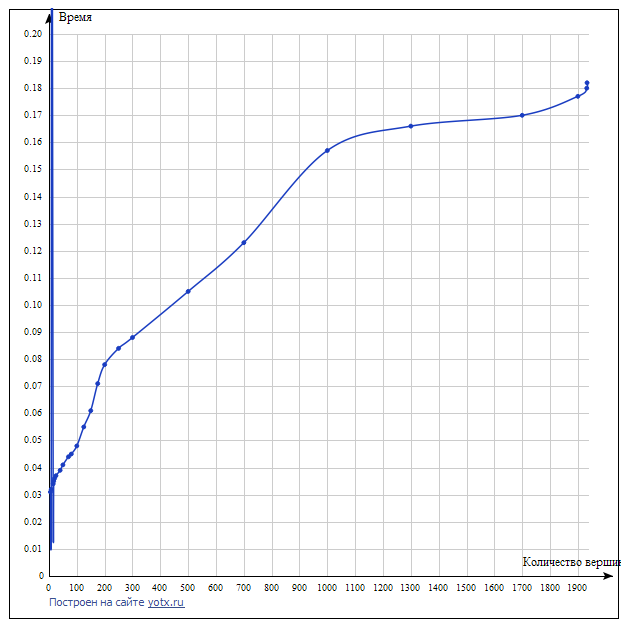
Программа корректно работает на графах с количеством вершин до 1933 (ограничение на входные данные).

Экспериментально получены следующие зависимости.

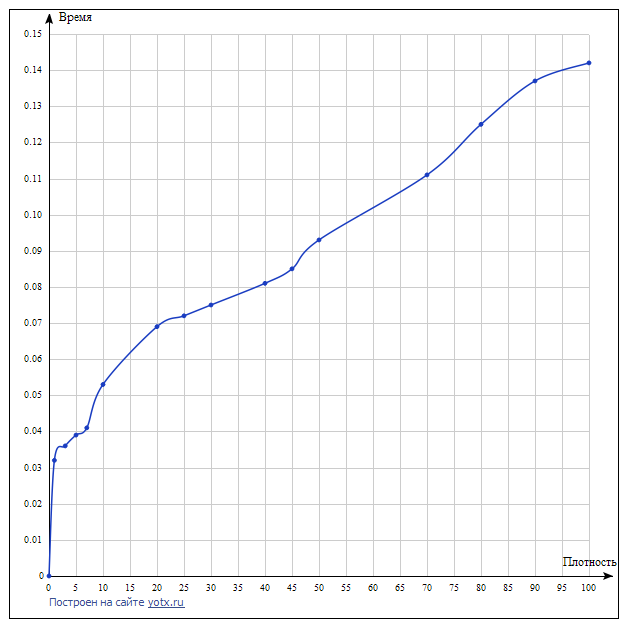
Зависимость времени выполнения от количества вершин при плотности 90%:



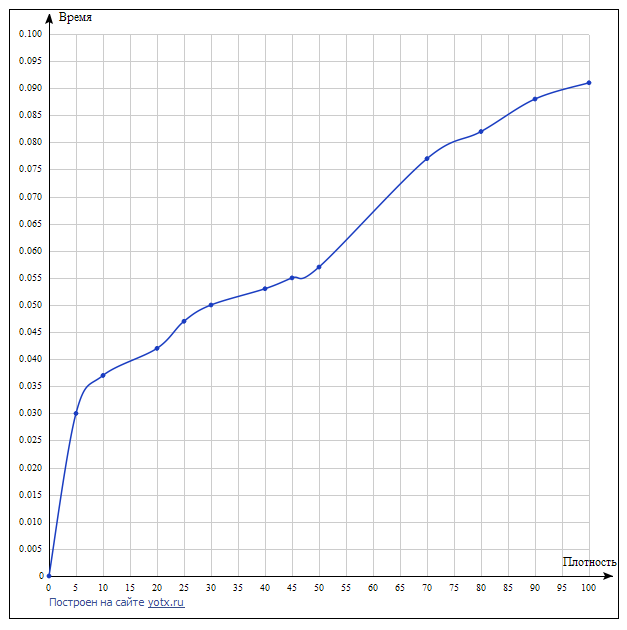
Зависимость времени выполнения от количества вершин при плотности 30%:



Зависимость времени выполнения от плотности (количества ребер) при количестве вершин 50:



Зависимость времени выполнения от плотности (количества ребер) при количестве вершин 5:



Теоритически алгоритм работает за время *O(n+m)*, где *n* — количество вершин, *m* — рёбер в графе.

Полученные на практике зависимости совпадают с теоритическими, то есть при увеличении количества вершин и ребер линейно возрастает время выполнения программы.

**ЗАКЛЮЧЕНИЕ**

**Вывод**

В ходе данной работы был создан генератор матрицы смежности для неориентированного графа и реализован алгоритм поиска мостов на языке программирования С++. При выполнении курсовой работы использовались знания, полученные во время лекций и практических занятий по работе с алгоритмами на графах. Исследованы теоритически и экспериментально зависимости времени выполнения от количества вершин и ребер.

**список использованных источников**

1. [Кнут, Д. Э.](https://ru.wikipedia.org/wiki/%D0%9A%D0%BD%D1%83%D1%82,_%D0%94%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%B4_%D0%AD%D1%80%D0%B2%D0%B8%D0%BD) [Искусство программирования. Том 3. Сортировка и поиск](https://books.google.ru/books?id=92rW-nktlbgC&printsec=frontcover&dq=editions:spjjKVwoQ3QC&hl=ru&sa=X&ved=0ahUKEwiUjLbc88rQAhVCfiwKHUJlBQoQuwUIIjAB#v=onepage&q&f=false). The Art of Computer Programming. Volume 3. Sorting and Searching / под ред. Ю. В. Козаченко. Москва: Вильямс, 2007. Т. 3.— 832 с.

2. [Левитин А. В.](https://www.wikidata.org/wiki/Q21694518) Глава 5. Метод уменьшения размера задачи: Поиск в глубину //[Алгоритмы. Введение в разработку и анализ](https://www.wikidata.org/wiki/Q21694522) — М.: [Вильямс](https://www.wikidata.org/wiki/Q21694521), 2006. — С. 212–215. — 576 с.

3. Кормен Т., Лейзерсон Ч., Ривест Р.: «Алгоритмы: построение и анализ». Вильямс, 2-е издание, 2005.

4. Роберт Седжвик. Фундаментальные алгоритмы на C. Анализ/Структуры данных/Сортировка/Поиск = Algorithms in C. Fundamentals/Data Structures/Sorting/Searching. СПб.: ДиаСофтЮП, 2003. С. 672.

5. Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. Алгоритмы: построение и анализ = INTRODUCTION TO ALGORITHMS. 2-е изд. М.: [«Вильямс»](https://ru.wikipedia.org/w/index.php?title=%D0%92%D0%B8%D0%BB%D1%8C%D1%8F%D0%BC%D1%81_(%D0%B8%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE)&action=edit&redlink=1), 2006. С. 1296.

6. Евстигнеев В. А. Глава 3. Итеративные алгоритмы глобального анализа графов. Пути и покрытия //[Применение теории графов в программировании](http://publ.lib.ru/ARCHIVES/E/EVSTIGNEEV_Vladimir_Anatol'evich/_Evstigneev_V.A..html)/ Под ред. А. П. Ершова. — Москва: [Наука](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D1%83%D0%BA%D0%B0_(%D0%B8%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE)). Главная редакция физико-математической литературы, 1985. — С. 138-150. — 352 с.

**приложение**

**Текст программы**

#include <iostream>

#include <vector>

#include <algorithm>

#include <time.h>

#include <fstream>

using namespace std;

ifstream infile("input.txt"); //файл ввода

ofstream outfile("output.txt"); //файл вывода

const int MAXN = 1933; //максимальное значение

vector<int> g[MAXN]; //вектор смежности

int mx[MAXN][MAXN]; //матрица смежности

bool used[MAXN];

int timer, tin[MAXN], fup[MAXN];

unsigned int n; // количество вершин

vector< pair<int, int> > bridges; //мосты

void dfs (int v, int p = -1); //обход в глубину

void find\_bridges(); //поиск мостов

void isBridge(int source, int destination); //проверка на мост

void inputData(); //ввод данных

void convertMatrixToAdjacenciesList(); //конвертация из матрицы в список смежности

void randomGenerator(); //рандомная генерация

void printGraphState(); //печать графа

int main(/\*int argc, char \*argv[]\*/)

{

cout << "Graph type: 0-input, 1-random" << endl;

unsigned int enter=0; //ввод вручную или генерация

cin >> enter;

while (enter < 0 || enter > 1){

cout << "try again!" << endl;

cin >> enter;

}

if (enter == 0) //ввод графа

{

inputData();

printGraphState(); //печать

}

if (enter == 1){ //генерация графа

randomGenerator();

}

cout << "Bridges is finding" <<endl;

find\_bridges(); //поиск мостов

cout << "Bridges: " << endl;

for(auto it : bridges) //вывод мостов

{

cout << it.first << " " << it.second << endl;

outfile << it.first << " " << it.second << endl;

}

cout << endl;

outfile << endl;

return 0;

}

void dfs (int v, int p) { //поиск в глубину

used[v] = true; //v - начальная вершина

tin[v] = fup[v] = timer++; //tin[v] - время захода в вершину v

for (size\_t i=0; i<g[v].size(); ++i) { //ходим по всему вектору

int to = g[v][i];

if (to == p) continue;

if (used[to])

//время fup[v] == min из времени

//захода в вершину tin[v] и времен захода

//в каждую вершину p

fup[v] = min (fup[v], tin[to]);

else {

dfs (to, v); //поиск в глубину

fup[v] = min (fup[v], fup[to]);

if (fup[to] > tin[v]) //является ли ребро мостом

isBridge(v,to);

}

}

}

void find\_bridges() { //поиск мостов

timer = 0;

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

used[i] = false; //критерий ребра дерева поиска

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

if (!used[i])

dfs (i);

}

void isBridge(int source, int destination) //проверка на мост

{

bridges.push\_back(make\_pair(source, destination));

}

void inputData() //ввод данных

{

char inputType; //выбор типа ввода

cout << "Choosr input type: 0 - from console; 1 - from file" << endl;

cin >> inputType;

if(inputType == '1') //ввод из файла

{

cout << "input type switched to file" << endl;

cout << "The content of input type is:" << endl;

infile >> n; //считывание количества вершин

cout << n << endl;

if(n > MAXN)

{

cout << "Number of vertices must be nonnegative" << endl;

exit(1);

}

cout << "Graph: " << endl; //вывод графа

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

{

vector<int> tmpVector;

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

{

int tmp;

infile >> tmp;

// check item

if(tmp != 0)

{

tmpVector.push\_back(j);

}

}

g[i] = tmpVector;

}

printGraphState(); //печать графа

}

else //ввод из консоли

{

cout << "Input number of vertices: ";

cin >> n; //считывание количества вершин

if(n > MAXN)

{

cout << "Number of vertices must be nonnegative" << endl;

return;

}

cout << "Input graph: " << endl; //ввод графа

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

{

vector<int> tmpVector;

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

{

int tmp;

cin >> tmp;

// check item

if(tmp != 0)

{

tmpVector.push\_back(j);

}

}

g[i] = tmpVector;

}

}

}

void printGraphState() //печать графа

{

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

{

for(unsigned int j = 0; j < g[i].size(); j++)

{

cout << g[i][j] << " "; //печать вектора смежности

}

cout << endl;

}

}

void randomGenerator() //рандомная генерация

{

cout << "Input Vertices count" << endl;

cin >> n; //ввод количества вершин

int density = -1; //ввод плотности

cout << "Input Density" << endl;

while(density < 0)

{

cin >> density;

if(density < 0)

{

cout << "Try again" << endl;

}

}

if (density > 100)

{

cout << "density can not have this value!";

exit(1);

}

unsigned int count = n\*n\*density/100; //подсчет ребер

cout << "Edges count in graph = " << count <<endl;

unsigned int real\_count = 0;

srand(time(0)); //генерация

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

{

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

{

mx[i][j] = 0;

}

}

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

{

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

{

if (mx[i][j]) break;

else{

if (real\_count != count){

mx[i][j] = rand() % 2;

if (mx[i][j]) {

mx[j][i] = 1;

real\_count++;

}

}

}

}

}

convertMatrixToAdjacenciesList();

printGraphState();

}

void convertMatrixToAdjacenciesList() //конвертация из матрицы в список смежности

{

for(unsigned int i = 0; i < MAXN; i++)

{

g[i] = vector<int>(0);

}

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

{

vector<int> tmpVector(0);

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

{

if(mx[i][j] != 0)

{

tmpVector.push\_back(j);

}

}

g[i] = tmpVector;

}

}