Министерство Образования и Науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение   
высшего профессионального образования

**“Санкт-Петербургский государственный электротехнический университет “ЛЭТИ” им. В.И. Ульянова (Ленина)”**

**(СПбГЭТУ)**

ФКТИ, кафедра МОЭВМ

**Отчет**

**по лабораторной работе №4**

**по дисциплине «Проектирование и анализ алгоритмов»**

Выполнила: Студентка группы 1381

Шнейдер Екатерина

Проверил: Казаков Б.Б.

г. Санкт-Петербург

2013г.

**Поставленная задача.**

***Мосты графа(Исследование алгоритма)***. Мост в связном неориентированном графе - это ребро, удаление которого делает граф несвязным. Реализовать алгоритм нахождения мостов графа (модификацией поиска в глубину из задачи 2: ребро {v,w} графа будет мостом тогда и только тогда, когда {v,w} является ребром дерева и Low(w)>=NumVert(w)). Провести экспериментальный анализ его эффективности на графах разной структуры. Продумать и предложить механизм генерации необходимых для эксперимента тестовых данных.

**Анализ алгоритма.**

Пусть дан неориентированный граф. Мостом называется такое ребро, удаление которого делает граф несвязным (или, точнее, увеличивает число компонент связности). Требуется найти все мосты в заданном графе.

Неформально эта задача ставится следующим образом: требуется найти на заданной карте дорог все "важные" дороги, т.е. такие дороги, что удаление любой из них приведёт к исчезновению пути между какой-то парой городов.

Алгоритм основан на поиске в глубину и работает за время O(n+m), где n — количество вершин, m — рёбер в графе.

Пусть мы находимся в обходе в глубину, просматривая сейчас 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. (В самом деле, мы этим условием проверяем, нет ли другого пути из v в to, кроме как спуск по ребру (v,to) дерева обхода в глубину.)

Чтобы эффективно реализовать этот факт для каждой вершины. Для этого воспользуемся "временами входа в вершину", вычисляемыми алгоритмом поиска в глубину.

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:

 fup[v] = \min \cases{
tin[v], & \cr
tin[p], & {[...]

(здесь "back edge" — обратное ребро, "tree edge" — ребро дерева)

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] \le 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.)

Таким образом, 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.

В программе были реализованы следующие функции:

int dfs (int v, int p = -1)- сама функция обхода в глубину

void IS\_BRIDGE(int a, int b)- это некая функция, которая будет реагировать на то, что ребро (a,b) является мостом и выводить это ребро на экран

void find\_bridges(int n)- она производит необходимую инициализацию и запуск обхода в глубину для каждой компоненты связности графа.

**Пример работы программы.**

В файле «Графы.txt» необходимо ввести размер матрицы смежности графа и ввести саму матрицу смежности (1 в ней будет означать, что существует ребро между данными вершинами, 0- ребра нет).

Пример1:

Размер матрицы – 7

Матрица:

0 1 1 1 0 0 0

1 0 1 1 0 0 0

1 1 0 0 0 0 0

1 1 0 0 1 0 0

0 0 0 1 0 1 0

0 0 0 0 1 0 1

0 0 0 0 0 1 0

Данный граф будет выглядеть следующим образом:

«3» «4»

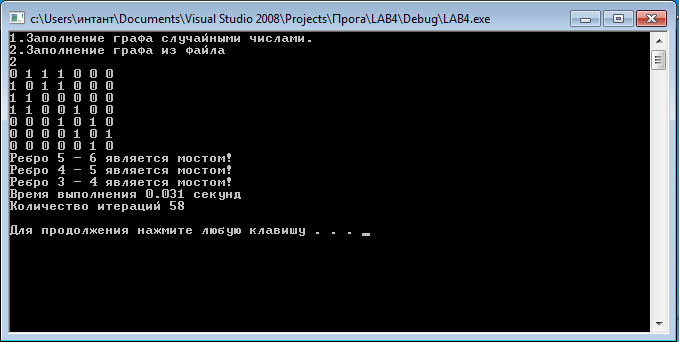
«0» «5»

«1»

«2» «6»

Как можно увидеть из рисунка мостами этого графа будут являться ребра:

(3,4),(4,5),(5,6).



Пример2:

Размер матрицы – 6

Матрица:

0 1 1 0 0 0

1 0 1 0 0 0

1 1 0 1 0 0

0 0 1 0 0 0

0 0 0 0 0 1

0 0 0 0 1 0

Данный граф будет выглядеть следующим образом:

«1» «4»

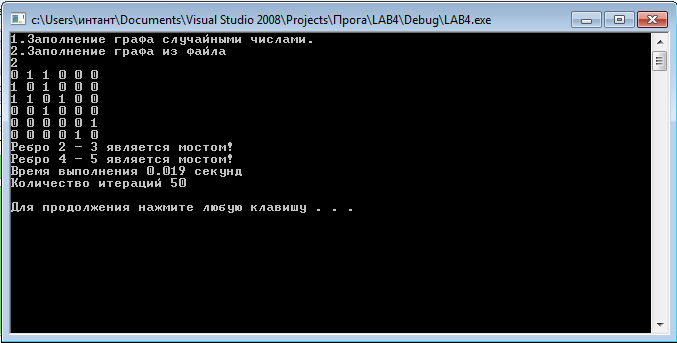
«0» «5»

«2»

«3»

Как можно увидеть из рисунка мостами этого графа будут являться ребра:

(2,3),(4,5).



**Код программы.**

#include <iostream>

#include <fstream>

#include <vector>

#include <time.h>

using namespace std;

const int MAXN = 100;

vector<int> g[MAXN];

bool used[MAXN];

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

int tab[100][100];

int rand (int tab[100][100], vector<int>g[MAXN])

{

int n;

cout << "Введите количество элементов в графе: ";

cin >> n;

srand(time(0));

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

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

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

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

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

tab[j][i]=tab[i][j];

if(i==j) tab[i][j]=0;

if(tab[i][j]==1)

{

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

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

}

}

return n;

}

void IS\_BRIDGE(int a, int b)

{

std::cout << "Ребро " << a << " - " << b << " является мостом!" << endl;

}

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

used[v] = true;

tin[v] = fup[v] = timer++;

for (size\_t i=0; i<g[v].size(); ++i)

{ iter++;

int to = g[v][i];

if (to == p) continue;

if (used[to])

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

else {

dfs (to, v);

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

if (fup[to] > tin[v])

IS\_BRIDGE(v,to);

return iter;

}

}

}

void find\_bridges(int n) {

timer = 0;

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

used[i] = false;

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

if (!used[i])

iter=iter + dfs (i);

}

int main()

{

setlocale(0,"rus");

int menu,n,i,j;

fstream fi;

cout << "1.Заполнение графа случайными числами.\n2.Заполнение графа из файла\n";

cin >> menu;

switch (menu)

{

case 1: n = rand(tab,g);

break;

case 2:

fi.open("Graph.txt", ios::in);

fi >> n;

int j,i;

if (!fi)

{

cerr << "ошибка!\n";

exit(1);

std::system("pause");

}

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

{

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

{

fi>>tab[i][j];

if(tab[i][j]==1)

{

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

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

}

if (tab[i][j] !=1 && tab[i][j] !=0 )

{

std::cout << " Матрица введена некорректно\n";

fi.close();

system ("pause");

return 0;

}

}

}

break;

}

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

{

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

cout << tab[i][j] << " ";

cout<<endl;

}

timex = clock();

find\_bridges(n);

timex = clock() - timex;

cout << "Время выполнения " << (double)timex/CLOCKS\_PER\_SEC << " секунд " << endl;

cout << "Количество итераций " << iter << endl;

fi.close();

cout << endl;

system ("pause");

return 0;

}

**Анализ работы программы.**

|  |  |  |
| --- | --- | --- |
| **Размер матрицы** | **Количество итераций** | **Время работы алгоритма** |
| **2** | **6** | **0.007с.** |
| **3** | **12** | **0.015с.** |
| **4** | **26** | **0.009с.** |
| **5** | **32** | **0.021с.** |
| **6** | **50** | **0.019с.** |
| **7** | **58** | **0.031с.** |
| **8** | **62** | **0.028с.** |

Вывод:

Была реализована программа поиска мостов в неориентированном графе.