ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И.Ульянова (Ленина)»

Кафедра МОЭВМ

**ОТЧЕТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4**

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

Выполнила: студентка группы №1381

Лавринович Т. В.

Проверил: Казаков Б.Б.

Санкт-Петербург

2013 г.

**Цель работы**

Изучить алгоритмы работы с графами, принцип их работы. Составить программу с использованием конкретного алгоритма.

**Задание**

Вариант 2а (двусвязные компоненты)

Реализовать алгоритм нахождения двусвязных компонент графа. Провести экспериментальный анализ его эффективности на графах разной структуры. (Исследование алгоритма.)

**Описание алгоритма**

Двусвязная компонента — это максимальный набор ребер, любые два ребра которого принадлежат общему простому циклу (простой цикл - цикл, не проходящий дважды через одну вершину).

Данный Алгоритм основан на "Алгоритме обхода графа в глубину" с внесением изменений.

Изменения: при попадании в уже обработанную вершину (то есть нахождение цикла), осуществляются следующие действия:

1. все частично найденные двусвязные компоненты, которые пересекаются с найденным циклом хотя бы по одному ребру, объединяются со всеми ребрами цикла все вместе и это множество ребер обозначается как новая двусвязная компонента (старые двусвязные компоненты, принадлежащие этому множеству удаляются);
2. если цикл ни с чем не пересекается, все его ребра обозначаются как принадлежащие к одной новой двусвязной компоненте.

**Исследование алгоритма**

Таблица 1. Результат работы программы

|  |  |  |  |
| --- | --- | --- | --- |
| Количество вершин графа | Количество ребер графа | Время выполнения | Количество итераций |
| 3 | 3 | 0,000144663 | 17 |
| 4 | 4 | 0,000156393 | 22 |
| 5 | 5 | 0,000164701 | 27 |
| 6 | 6 | 0,000186205 | 32 |
| 7 | 10 | 0,000296657 | 49 |
| 10 | 14 | 0,00041053 | 68 |
| 11 | 12 | 0,000355304 | 61 |
| 14 | 32 | 0,000899746 | 153 |

График зависимости времени от количества вершин графа указан на рисунке 1.

Рис.1

На рисунке 2 показана зависимость количества итераций от количества вершин графа.

Рис. 2

По полученным данным можно сделать вывод, что при использовании линейных списков можно добиться линейной скорости относительно количества, и если циклов в графе не очень много, то алгоритм превращается в простой алгоритм обхода в глубину, который работает за время O(m), где m–количество рёбер.

**Исходная программа**

#include <stdio.h>

#include <vector>

#include <algorithm>

#include <time.h>

#include <omp.h>

#include <iostream>

#include <Windows.h>

using namespace std;

typedef vector<int> VInt;

typedef vector<VInt> VVInt;

typedef VInt::iterator VIter;

typedef pair<int, int> PInt;

typedef vector<PInt> VPInt;

typedef vector<VPInt> VVPInt;

typedef VPInt::iterator VPIter;

class Grath {

private:

VVInt graph;

VInt vrem, parents, enter, low, bcc;

int myTime;

int newIndex;

int Cout\_iter;

public:

Grath(){myTime=0;newIndex=0;Cout\_iter=0;}

void visitLow(int u);

void visitBCC(int u);

int getBCC(int u, int v);

void job();

};

/\* Поиск в глубину, выполняющий вычисление enter и low\*/

void Grath::visitLow(int u) {

vrem[u] = 1;

low[u] = enter[u] = ++myTime;

Cout\_iter++;

for(VIter it = graph[u].begin(); it != graph[u].end(); it++)

if(vrem[\*it] == 0)

{

parents[\*it] = u;

visitLow(\*it);

low[u] = min(low[u], low[\*it]);

Cout\_iter++;

}

else

{

if(vrem[\*it] == 1 && \*it != parents[u])

{

low[u] = min(low[u], enter[\*it]);

Cout\_iter++;

}

}

}

/\* Второй поиск в глубину, присвающий идентификаторы bcc\*/

void Grath::visitBCC(int u) {

for(VIter it = graph[u].begin(); it != graph[u].end(); it++)

if(parents[\*it] == u)

{

Cout\_iter++;

if (low[\*it] < enter[u])

bcc[\*it] = bcc[u] ;

else

if (low[\*it] > enter[u])

bcc[\*it] = -1 ;

else

bcc[\*it] = (newIndex++);

visitBCC(\*it);

}

}

/\* Получение номера BCC, которому принадлежит ребро.\*/

int Grath::getBCC(int u, int v) {

Cout\_iter++;

if (enter[u] > enter[v])

return bcc[u];

else

return bcc[v];

}

void Grath::job() {

int n, m, i;

cout << "Введите количество вершин и количество рёбер:"<<endl;

// Прочитаем граф

scanf("%d%d", &n, &m);

graph.resize(n);

cout<< "Вводите рёбра:"<<endl;

while(m--) {

int from, to;

scanf("%d%d", &from, &to);

graph[from - 1].push\_back(to - 1);

graph[to - 1].push\_back(from - 1);

}

// Запустим первый поиск (вычислим enter и low)

vrem.assign(n, 0);

parents.assign(n, -1);

enter.resize(n);

low.resize(n);

double t1 = omp\_get\_wtime(); // системное время до выполнения функции

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

if(vrem[i] == 0)

{ Cout\_iter++;

visitLow(i);

}

// Запустим второй поиск (определение идентификаторов bcc)

// Второй поиск запускается "по следам" первого,

// то есть проходит по уже найденному дереву

bcc.assign(n, -1);

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

if(parents[i] == -1)

{

Cout\_iter++;

visitBCC(i);

}

// Выведем результат

VVPInt comps(newIndex);

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

for(VIter it = graph[i].begin(); it != graph[i].end(); it++)

if(i < \*it)

{

Cout\_iter++;

int id = getBCC(i, \*it);

if (id != -1)

comps[id].push\_back(PInt(i, \*it));

}

double t2 = omp\_get\_wtime(); // время после выполнения функции

bool z=false;

for(i = 0; i < newIndex; i++) {

printf("Компонент %d: ", i+1);

for(VPIter edge = comps[i].begin(); edge != comps[i].end(); edge++)

{

printf("(%d, %d) ", edge->first + 1, edge->second + 1);

z=true;

}

printf("\n");

}

if (z==false)

cout<<"Нет компонентов!"<<endl;

double time = t2 - t1;

cout<<"Время выполнения: "<<time<<endl;

cout<<"Количество базовых операций: "<<Cout\_iter<<endl;

}

int main()

{

setlocale(LC\_ALL,"Russian");

Grath g;

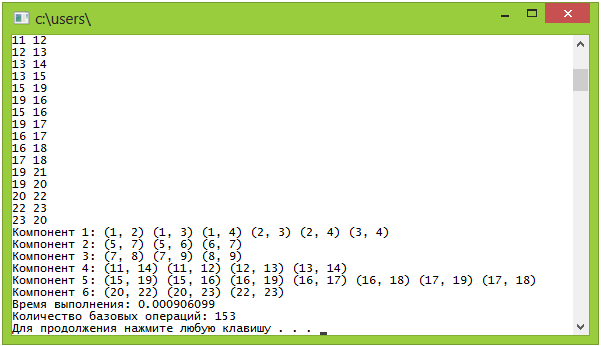
g.job();

system("PAUSE");

return 0;

}

**Результаты выполнения программы**

****

**Вывод**

При выполнении лабораторной работы были получены практические навыки работы с алгоритмами на графах. Была составлена программа, решающая задачу двусвязный компонент графа. При правильно введённых данных ошибок в её работе не обнаружено.