Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

**Расчетно-графическое задание**

По дисциплине: «Защита информации»

Выполнил:  
ст. группы ИП-411

Выдрин Андрей.Э.

Проверил:  
доцент кафедры ПмиК  
Ракитский А.А.

Новосибирск 2017 г.

**Содержание**

[**1 Задание** 3](#_Toc469496065)

[**1.1 Постановка задачи** 3](#_Toc469496066)

[**1.2 Реализация (листинг)** 3](#_Toc469496067)

[**1.3 Скриншоты** 9](#_Toc469496068)

[**1.3.1 Входные данные** 6](#_Toc469496069)

[**1.3.2 Выходные данные** 7](#_Toc469496070)

# **1 Задание**

### **1.1 Постановка задачи**

Реализация алгоритма «Доказательство с нулевым знанием». Задание «Гамильтонов цикл». Граф задаётся в файле в следующем формате: В первой строке файла два числа n и m – количество вершин и количество рёбер графа соответственно. Числа большие, порядка 106. В следующих n строках идёт перечисление рёбер графа в виде двух чисел (номера вершин, которые соединяет ребро).

### **1.2 Реализация (листинг)**

### 

#include <bits/stdc++.h>

using namespace std;

vector<int> search\_h\_cycle(vector< vector<int> > graph) {

vector<int> path;

map<int, vector<int> > N;

path.push\_back(0);

N[0] = graph[0];

while(path.size()) {

int x = path.back();

if(N[x].size()) {

int y = N[x].back();

N[x].pop\_back();

auto it = find(path.begin(), path.end(), y);

if(it == path.end()) {

path.push\_back(y);

N[y] = graph[y];

if(path.size() == graph.size()) {

if(find(N[y].begin(), N[y].end(), 0) != N[y].end()){

path.push\_back(0);

return path;

}

}

}

} else {

path.pop\_back();

}

}

}

vector< vector<int> > make\_vv(set< pair<int,int> > edges, int n) {

vector< vector<int> > graph(n);

for(auto t: edges) {

graph[t.first - 1].push\_back(t.second - 1);

graph[t.second - 1].push\_back(t.first - 1);

}

return graph;

}

vector<int> get\_perm(int n) {

vector<int> out(0);

out.push\_back(0);

for(int i=1; i <= n; i++)

out.push\_back(i);

for(int i=1; i <= n; i++) {

swap(out[i], out[rand() % n + 1]);

}

return out;

}

set< pair<int,int> > get\_ism( set< pair<int,int> > edges, vector<int> perm) {

set< pair<int,int> > out;

for(auto t: edges) {

out.insert(make\_pair(perm[t.first], perm[t.second]));

}

return out;

}

vector<int> get\_r\_perm(vector<int> perm) {

vector<int> out(perm.size());

for(int i=1; i < perm.size(); i++) {

out[perm[i]] = i;

}

return out;

}

bool duplicates(vector<int> vec) {

vec.pop\_back();

sort(vec.begin(), vec.end());

for(int i=0; i<vec.size() - 1; i++) {

if(vec[i] == vec[i+1])

return true;

}

return false;

}

void print\_graph(set< pair<int,int> > graph) {

for(auto t: graph) {

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

}

cout << endl << endl;

}

void print\_vec(vector<int> vec) {

for(auto t: vec)

cout << t << " ";

cout << endl << endl;

}

void print\_perm(vector<int> perm) {

for(int i=1; i < perm.size(); i++) {

cout << i << "->" << perm[i] << "\t";

}

cout << endl << endl;

}

class Freelancer{

set< pair<int,int> > edges;

int n, m;

vector<int> h\_cycle;

vector<int> perm;

public:

Freelancer(set< pair<int,int> > ed, int n, int m) {

edges = ed;

this->n = n;

this->m = m;

h\_cycle = search\_h\_cycle(make\_vv(edges,n));

for(int i=0; i<h\_cycle.size(); i++)

h\_cycle[i]++;

cout << "Гамильтонов цикл: " << endl;

print\_vec(h\_cycle);

perm.resize(n+1);

}

set< pair<int,int> > get\_ism\_graph() {

perm = get\_perm(n);

return get\_ism(edges,perm);

}

vector<int> get\_pperm() {

return perm;

}

vector<int> get\_p\_cycle() {

vector<int> out;

for(auto t: h\_cycle) {

out.push\_back(perm[t]);

}

return out;

}

vector<int> get\_h\_cycle() {

return h\_cycle;

}

};

class Customer{

set< pair<int,int> > graph;

int n, m;

set< pair<int,int> > ism\_graph;

public:

Customer(string filename) {

ifstream in(filename.c\_str());

in >> n >> m;

cout << "n: " << n << "\t m: " << m << endl;

for(int i=0; i<m; i++) {

int v,u;

in >> v >> u;

graph.insert(make\_pair(v,u));

}

cout << "Изначальный граф: " << endl;

print\_graph(graph);

}

set< pair<int,int> > get\_graph() {

return graph;

}

int getn(){

return n;

}

int getm() {

return m;

}

void set\_ism\_graph(set< pair<int,int> > gr) {

ism\_graph = gr;

cout << "Изоморфный граф: " << endl;

print\_graph(ism\_graph);

}

bool check\_ism(vector<int> perm) {

auto rperm = get\_r\_perm(perm);

auto temp\_graph = get\_ism(ism\_graph, rperm);

if(temp\_graph.size() == graph.size()) {

if(equal(graph.begin(), graph.end(), temp\_graph.begin()))

return true;

}

return false;

}

bool check\_cycle(vector<int> cycle) {

if(cycle.size() != n + 1) {

cout << "Неверный размер!" << endl;

return false;

}

if(duplicates(cycle)) {

cout << "Есть повторения!" << endl;

return false;

}

if(cycle[0] != cycle[cycle.size() - 1]) {

cout << "Это не цикл!" << endl;

return false;

}

for(int i=0; i<cycle.size() - 1; i++) {

if(!ism\_graph.count(make\_pair(cycle[i], cycle[i+1])) &&

!ism\_graph.count(make\_pair(cycle[i+1], cycle[i]))) {

cout << "Нет ребра!" << endl;

return false;

}

}

return true;

}

};

int main() {

setlocale(LC\_ALL,"");

srand(time(NULL));

Customer Bob("graph.txt");

Freelancer Alice( Bob.get\_graph(), Bob.getn(), Bob.getm() );

cout << "=======================================================" << endl;

for(int i=0; i<10; i++) {

Bob.set\_ism\_graph( Alice.get\_ism\_graph() );

if(rand() % 2) {

cout << "Проверка на изоморфность. " << endl;

cout << "Перестановка: ";

print\_perm( Alice.get\_pperm() );

if( Bob.check\_ism(Alice.get\_pperm()) )

cout << "OK" << endl;

else

cout <<"FALSE" << endl;

} else {

cout << "Проверка цикла. " << endl;

cout << "Изоморфный цикл: " << endl;

print\_vec(Alice.get\_p\_cycle());

if( Bob.check\_cycle(Alice.get\_p\_cycle()) )

cout << "OK" << endl;

else

cout <<"FALSE" << endl;

}

cout << "=======================================================" << endl;

}

}

### **1.3 Скриншоты**

### **1.3.1 Входные данные**

### **1.3.2 Выходные данные**



