Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет

телекоммуникаций и информатики»

Кафедра ПМиК

РГР по дисциплине

Защита Информации

по теме “Доказательство с нулевым знанием”

для задачи Гамильтонов цикл

Выполнил: студент 4 курса

Ф. ИВТ, группа: ИП-711

Мартасов И. О.

Проверил: доцент кафедры ПМиК

Ракитский Антон Андреевич

Новосибирск, 2020

**Содержание**

1. Задание.
2. Описание протокола
3. Текст программы.
4. Результаты работы

**Задание**

Необходимо написать программу, реализующую протокол

доказательства с нулевым знанием для задачи «Гамильтонов цикл»

**Описание протокола**

Рассматриваемая задача имеет важное теоретическое значение. Блюм (Manuel Blum) показал, что, выражаясь неформально, любое математическое утверждение может быть представлено в виде графа, причем доказательство этого утверждения соответствует гамильтонову циклу в этом графе. Поэтому наличие протокола доказательства с нулевым знанием для гамильтонова цикла означает, что доказательство любого математического утверждения может быть представлено в форме доказательства с нулевым знанием.

Гамильтоновым циклом в графе называется непрерывный путь, проходящий через все вершины графа ровно по одному разу.

Нашей задачей будет построение криптографического протокола, с помощью которого Алиса будет доказывать Бобу, что она знает гамильтонов цикл в некотором графе G так, чтобы Боб не получил никаких знаний о самом этом цикле. Иными словами, Алиса будет предоставлять Бобу доказательство с нулевым знанием.

Итак, допустим, что Алиса знает гамильтонов цикл в графе G. Теперь она может это доказывать Бобу и всем, кто имеет граф G, с помощью описываемого ниже протокола. Алиса может использовать это доказательство, например, для идентификации своей личности.

Протокол доказательства состоит из следующих четырех шагов (пояснения будут даны ниже).

Шаг 1. Алиса генерирует перестановку P и строит граф F , являющийся копией исходного графа G, где у всех вершин новые, случайно выбранные номера, и соответствующим образом строит цикл Hf, являющийся Гамильтоновым для графа F . На языке теории графов говорят, что F изоморфен G. Этот граф Алиса передает Бобу.

Шаг 2. Боб, получив граф F , задает Алисе один из двух вопросов.

1. Действительно ли граф H изоморфен G?

2. Каков гамильтонов цикл для графа H?

Шаг 3. Алиса отвечает на соответствующий вопрос Боба.

1. Она передает Бобу перестановку P, после чего Боб переводит граф G по этой перестановке в граф F’.

2. Она передает Бобу Гамильтонов цикл Hf.

Шаг 4. Получив ответ, Боб проверяет правильность и убеждается либо в том, что предъявленная перестановка действительно переводят граф G в граф F, либо в том что Hf действительно является Гамильтоновым циклом для графа F.

Весь протокол повторяется t раз.

**Текст программы**

#include <list>

#include <vector>

#include <sstream>

#include <Windows.h>

#include "LibSignes.h"

using namespace std;

void print\_vector(int\* vector, int n)

{

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

{

cout << vector[i] << " ";

}

cout << endl << endl;

}

void print\_graph(int\*\* g, int n, int \*vertex\_size)

{

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

{

cout << "VERTEX " << i + 1 << ": ";

for (int j = 0; j < vertex\_size[i]; j++)

{

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

}

cout << endl;

}

cout << endl;

}

int check\_izomorph\_graph(int n, int\*\* g, int\*\* f, int\* vertex\_size, int\* p, int\* f\_vertex\_size)

{

int ret = 0;

bool is\_true = true;

int\*\* restore\_graph = new int\* [n];

int\* restore\_vertex\_size = new int[n];

cout << "GRAPH G - " << endl;

print\_graph(g, n, vertex\_size);

cout << "GRAPH F - " << endl;

print\_graph(f, n, f\_vertex\_size);

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

{

restore\_vertex\_size[p[i] - 1] = vertex\_size[i];

restore\_graph[p[i] - 1] = new int[restore\_vertex\_size[p[i] - 1]];

for (int k = 0; k < restore\_vertex\_size[p[i] - 1]; k++)

{

restore\_graph[p[i] - 1][k] = p[g[i][k] - 1];

}

}

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

{

for (int j = 0; j < restore\_vertex\_size[i]; j++)

{

if (f[i][j] != restore\_graph[i][j])

{

is\_true = false;

break;

}

}

}

if (is\_true == false)

{

ret = -4;

goto finally;

}

cout << "RESTORE GRAPH F - " << endl;

print\_graph(restore\_graph, n, restore\_vertex\_size);

finally:

return ret;

}

int check\_hamilton\_cycle(int \*\*f, int\* f\_vertex\_size, int\* hamilton\_cycle\_f, int n)

{

int i = 0;

int ret = 0;

bool is\_true\_cycle;

cout << "GRAPH F - " << endl;

print\_graph(f, n, f\_vertex\_size);

cout << "HAMILTON CYCLE FOR GRAPH F:" << endl;

print\_vector(hamilton\_cycle\_f, n);

while (i < n - 1)

{

is\_true\_cycle = false;

for (int j = 0; j < f\_vertex\_size[hamilton\_cycle\_f[i] - 1]; j++)

{

if (f[hamilton\_cycle\_f[i] - 1][j] == hamilton\_cycle\_f[i + 1])

{

i++;

is\_true\_cycle = true;

break;

}

}

if (is\_true\_cycle == false)

{

ret = -5;

goto finally;

}

}

finally:

return ret;

}

int hamilton(int \*\*a, int\* c, int\* path, int n, int k, int v0)

{

int v, q1 = 0;

for (v = 0; v < n && !q1; v++)

{

if (a[v][path[k - 1]] || a[path[k - 1]][v])

{

if (k == n && v == v0)

{

q1 = 1;

}

else if (c[v] == -1)

{

c[v] = k;

path[k] = v;

q1 = hamilton(a, c, path, n, k + 1, v0);

if (!q1)

{

c[v] = -1;

}

}

else

{

continue;

}

}

}

return q1;

}

int main(void)

{

srand(time(NULL));

int n = 0;

int v0 = 7;

int ret = 0;

int vertex = 0;

int counter = 0;

int buf\_counter = 0;

int\*\* g;

int\* vertex\_size;

int\* hamilton\_cycle;

int\*\* f;

int\* f\_vertex\_size;

int\* hamilton\_cycle\_f;

int\* c;

int\* p;

int\*\* matrix;

int tmp;

int choose;

int rand\_index1;

int rand\_index2 ;

int vertex\_buf[1000] = { 0 };

LibSignes cryptograph;

string s;

stringstream ss;

ifstream fin("graph.txt");

if (fin.good())

{

getline(fin, s);

ss << s;

ss >> n;

c = new int[n];

p = new int[n];

g = new int\* [n];

matrix = new int\* [n];

vertex\_size = new int[n];

hamilton\_cycle = new int[n];

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

{

vertex\_size[i] = 0;

}

ss.clear();

while (!fin.eof())

{

getline(fin, s);

ss << s;

/\*if (n == counter)

{

int j = 0;

while (ss >> vertex)

{

hamilton\_cycle[j] = vertex;

j++;

}

}

else

{\*/

while (ss >> vertex)

{

vertex\_size[counter]++;

vertex\_buf[buf\_counter] = vertex;

buf\_counter++;

}

g[counter] = new int[buf\_counter];

for (int j = 0; j < buf\_counter; j++)

{

g[counter][j] = vertex\_buf[j];

}

counter++;

ss.clear();

buf\_counter = 0;

//}

}

if (n != counter)

{

cout << "WRONG DATA" << endl;

ret = -2;

goto finally;

}

}

else

{

cout << "NO FILE" << endl;

ret = -1;

goto finally;

}

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

{

matrix[j] = new int[n];

}

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

{

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

{

matrix[i][j] = 0;

}

}

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

{

for (int j = 0; j < vertex\_size[i]; j++)

{

matrix[i][g[i][j] - 1] = 1;

}

}

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

{

c[i] = -1;

}

c[v0 - 1] = v0 - 1;

hamilton\_cycle[0] = v0 - 1;

if (hamilton(matrix, c, hamilton\_cycle, n, 1, v0 - 1))

{

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

{

hamilton\_cycle[i]++;

}

}

else

{

cout << "NO HAMILTON CYCLE!" << endl;

ret = -10;

goto finally;

}

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

{

p[i] = i + 1;

}

cout << "GRAPH G - " << endl;

print\_graph(g, n, vertex\_size);

cout << "HAMILTON CYCLE FOR GRAPH G - " << endl;

print\_vector(hamilton\_cycle, n);

for(int t = 0; t < 10; t++)

{

cout << "ITERATION " << t + 1 << ": " << endl;

f = new int\* [n];

f\_vertex\_size = new int[n];

hamilton\_cycle\_f = new int[n];

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

{

rand\_index1 = rand() % n;

rand\_index2 = rand() % n;

tmp = p[rand\_index1];

p[rand\_index1] = p[rand\_index2];

p[rand\_index2] = tmp;

}

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

{

f\_vertex\_size[p[i] - 1] = vertex\_size[i];

f[p[i] - 1] = new int[f\_vertex\_size[p[i] - 1]];

for (int k = 0; k < f\_vertex\_size[p[i] - 1]; k++)

{

f[p[i] - 1][k] = p[g[i][k] - 1];

}

hamilton\_cycle\_f[i] = p[hamilton\_cycle[i] - 1];

}

cout << "P:" << endl;

print\_vector(p, n);

choose = rand() % 2 + 1;

switch (choose)

{

case 1:

cout << "1 - IS F ISOMORPH TO G? " << endl;

Sleep(3000);

ret = check\_izomorph\_graph(n, g, f, vertex\_size, p, f\_vertex\_size);

if (0 != ret)

{

cout << "GRAPHS ARE NOT EQUALS!" << endl;

ret = -5;

goto finally;

}

else

{

cout << "GRAPHS ARE EQUALS!" << endl;

}

break;

case 2:

cout << "2 - GIVE ME HAMILTON CYCLE" << endl;

Sleep(3000);

ret = check\_hamilton\_cycle(f, f\_vertex\_size, hamilton\_cycle\_f, n);

if (0 != ret)

{

cout << "CYCLE IS NOT CORRECT!" << endl;

ret = -6;

goto finally;

}

else

{

cout << "CYCLE IS CORRECT!" << endl;

}

break;

}

}

finally:

return ret;

}

**Результаты работы**

****