Министерство высшего образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ

УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

Отчет по лабораторной работе №3

«Алгоритмы на графах»

По дисциплине СиАОДвЭВМ

Выполнил студент гр. 438-3:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Канаев О.А.

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Проверил

Доктор технических наук,

доцент каф. АСУ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Горитов А.Н

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Томск 2020

**Цель работы:**

Получить первичные навыки создания алгоритмов на графах.

**Задание:**

ЗАДАНИЕ №15. Напишите программу, которая с помощью алгоритма поиска в глубину находит множество фундаментальных циклов связного неориентированного графа, заданного списками смежности. После завершения работы с динамическими структурами данных необходимо освободить занимаемую ими память.

**Теория:**

**Граф** - совокупность точек, соединенных линиями. Точки называются вершинами, или узлами, а линии – ребрами, или дугами.

**Поиск в глубину -** DFS следует концепции «погружайся глубже, головой вперед» («go deep, head first»). Идея заключается в том, что мы двигаемся от начальной вершины (точки, места) в определенном направлении (по определенному пути) до тех пор, пока не достигнем конца пути или пункта назначения (искомой вершины). Если мы достигли конца пути, но он не является пунктом назначения, то мы возвращаемся назад (к точке разветвления или расхождения путей) и идем по другому маршруту.

**Списоки смежности** - один из способов представления графа в виде коллекции списков вершин. Каждой вершине графа соответствует список, состоящий из "соседей" этой вершины.

**Отыскание фундаментального множества циклов в графе**

Если к стягивающему дереву <V, T> графа G = <V, E> мы добавим произвольную хорду eE\T, то возникший при этом подграф C = <V, T{e}> содержит в точности один цикл (под циклом мы здесь будем понимать элементарный цикл). Обозначим этот цикл через Ce. Множество C = {Ce : eE \ T} называют фундаментальным множеством циклов графа G (относительно стягивающего дерева <V, T>). Название "фундаментальное" связано с тем, что каждый цикл графа G можно достаточно просто получить из циклов множества G.

Нахождение фундаментального множества циклов имеет прикладное значение, например, при анализе электрических цепей. А именно, каждому фундаментальному циклу в графе, соответствующему данной электрической цепи, мы можем поставить в соответствие закон Кирхгофа для напряжений: сумма падения напряжений вдоль цикла равна 0. Нахождение фундаментального множества циклов позволяет выделить в математическом описании цепи независимые уравнения.

Поэтому, если анализируемое нами ребро {v, u} замыкает цикл (то есть WGN[v]

> WGN[u] > 0 и u не находится непосредственно под верхним элементом стека), то

вершина u находится в стеке и цикл, замыкаемый ребром {v, u}, представлен группой

элементов стека, начиная с v и кончая вершиной u.

Алгоритм Cycle(v);

1 begin

2 d := d+1; Стек[d] := v; num := num+1; WGN[v] := num;

3 for u  List[v] do

4 if WGN[u] = 0 then Cycle(u)

5 else if (u  Стек[d–1]) and (WGN[v] > WGN[u])

then

6 выписать цикл с вершинами

7 Стек[d], Стек[d–1], ..., Стек[c], где Стек[c] = u

8 d := d–1

9 end;

Здесь переменные d, num, СТЕК, List, WGN – глобальные.

Перед началом работы алгоритма необходимо выполнить инициализацию

рабочих переменных и структур данных:

1 for vV do WGN[v] := 0; num := 0; {инициализация}

2 d := 0; Стек[0] := 0; {d – число элементов в стеке}

3 for rV do

4 if WGN[r] = 0 then Cycle(v)

**Алгоритм:**

1. Считать граф из input.txt

2. Структурировать его

3. Проинициализировать Stack и массив WGN для путей от корня до вершины

4. Если WGN для вершины равен нулю выполнить Cycle (проверяемую вершина).

5. Занести вершину в Stack. Проверяя все ребра данной вершины, если WGN[конец ребра] равен 0, выполнить Cycle(конец ребра). Иначе, если WGN[проверяемой вершины]>WGN[конца ребра] и конец ребра в стеке не находится под вершиной стека вывести стек пока вершина стека не станет равна концу ребра. Перейти на следующее ребро

6. После проверки всех ребер убрать вершину из стека и перейти к другой

**Решение:**

**main.cpp:**

#include <iostream>

#include <fstream>

#include <cstdlib>

#include <string>

#include "List.h"

using namespace std;

int main()

{

system("chcp 1251");

ifstream file;

file.open("input.txt");

int\* Size\_of\_vertex= nullptr, \*\*arr=nullptr;

int Size\_of\_graph, R\_max = 10;

if (file.is\_open())

{

file >> Size\_of\_graph;

arr = (int\*\*)malloc(sizeof(int\*)\*Size\_of\_graph);

Size\_of\_vertex = (int\*)malloc(sizeof(int)\*Size\_of\_graph);

char \*str= new char[R\_max];

file.getline(str, R\_max \* 2);

for (int i = 0, strSize = R\_max \* 2; i < Size\_of\_graph; i++)

{

Size\_of\_vertex[i] = 1;

file.getline(str, strSize);

string str2 = str;

Size\_of\_vertex[i] += (str2.size() / 2);

arr[i] = (int\*)malloc(sizeof(int)\*Size\_of\_vertex[i]);

for (int j = 0, k = 0; k < Size\_of\_vertex[i]; k++, j += 2)

{

arr[i][k] = atoi(&(str2[j]));

}

}

cout << endl;

}

else

cout << "Нельзя открыть файл!" << endl;

List\*\* start = new List\*[Size\_of\_graph];//Фоимирование графа в структурy

for (int i = 0; i < Size\_of\_graph; i++)

{

start[i] = form(arr[i], Size\_of\_vertex[i]);

}

free(Size\_of\_vertex);

cout << "Graph" << endl;//Cout граф

for (int i = 0; i < Size\_of\_graph; i++)

{

List\* p = start[i];

out(p);

}

free(arr);

int i = Size\_of\_graph;//Init

int \*WGN= new int[Size\_of\_graph - 1];

fill(WGN, WGN + Size\_of\_graph, 0);

int \*Stack= new int[Size\_of\_graph];

Stack[0] = 0;

for (int i = 0; i < Size\_of\_graph; i++)

{

List\* p = start[i];

if (WGN[i] == 0)

Cycle(start, p, WGN, Stack);

}

system("pause");

return 0;

}

**List.h:**

#pragma once

#include <iostream>

struct List {

int value;

List\* next;

};

List\* form(int\* mas, int sizeLine);

void out(List\* p);

void Cycle(List\*\* start, List\* p, int\* wgn, int \*Stack);

**List.cpp:**

#include "List.h"

using namespace std;

int d = 0, num = 0;

List \* form(int \* mas, int sizeLine)

{

List\* p = (List\*)malloc(sizeof(List));

p->value = mas[0];

p->next = nullptr;

List\* start = p;

List\* pk;

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

{

pk = p;

p = (List\*)malloc(sizeof(List));

p->value = mas[i];

p->next = nullptr;

pk->next = p;

}

return start;

}

void out(List \* p)

{

cout << p->value << " ";

if (p->next)

out(p->next);

else

cout << endl;

}

void Cycle(List \*\* start, List \* p, int \* wgn, int \* Stack)

{

int v = p->value;//Номер вершины

d++;

Stack[d] = v;//Заносим в стек вершину

num++;

wgn[v] = num;

while (p->next)

{

int u = p->next->value;//Вершина- конец ребра

if (wgn[u] == 0)

{

//Проверяем другую вершину

List\* pk = start[u];

Cycle(start, pk, wgn, Stack);

}

else

{

if (u != Stack[d - 1] && wgn[v] > wgn[u])

{

//Выводим цикл

cout << "Cycle" << endl;

int c = d;

do

{

cout << Stack[c] << " ";

c--;

} while (Stack[c] != u);

cout << Stack[c];

cout << endl;

}

}

p = p->next;//Сменяем вершину

}

d--;//Удаляем из стека вершину

}

**Результат работы:**

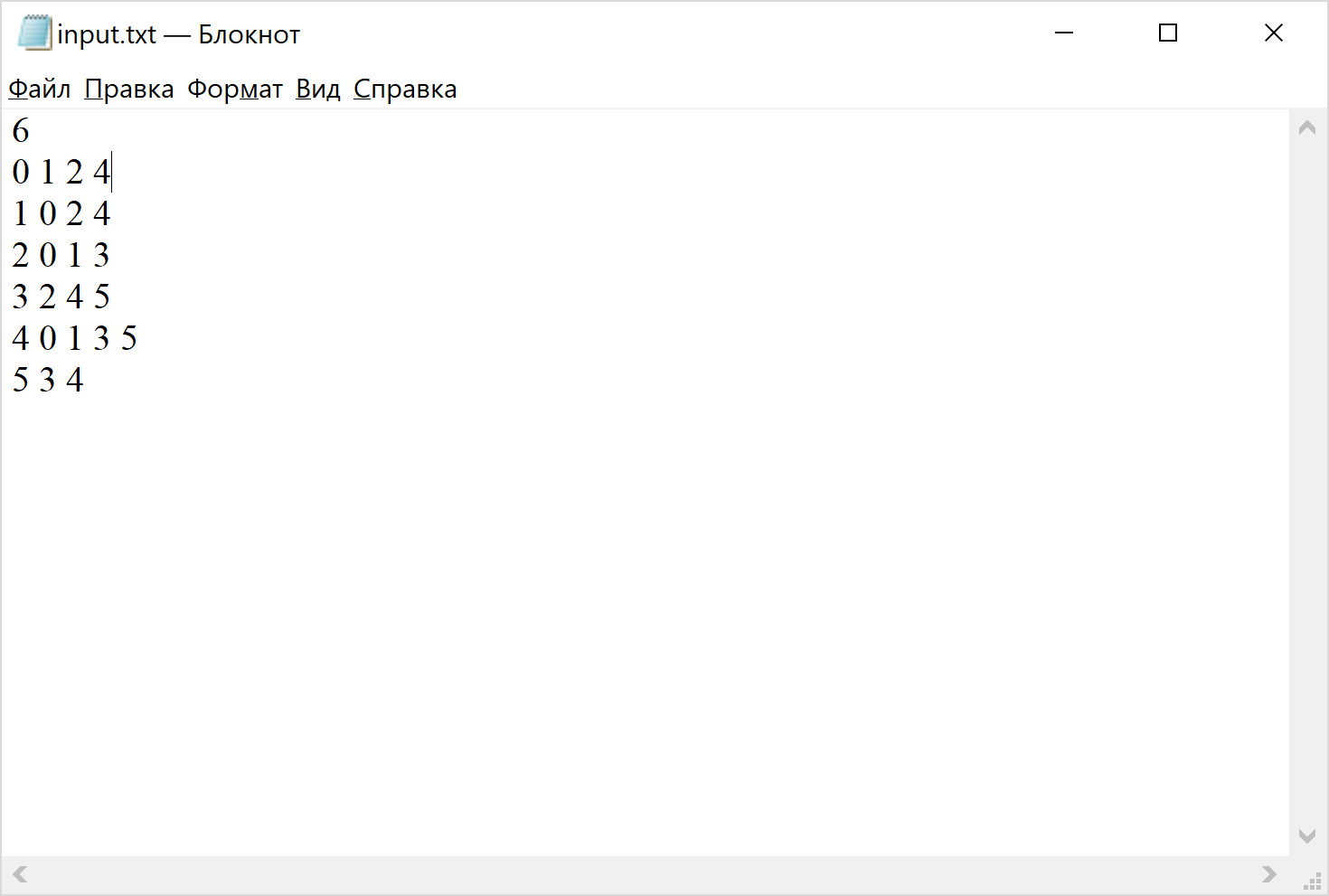


Рисунок №1 – Исходные данные.

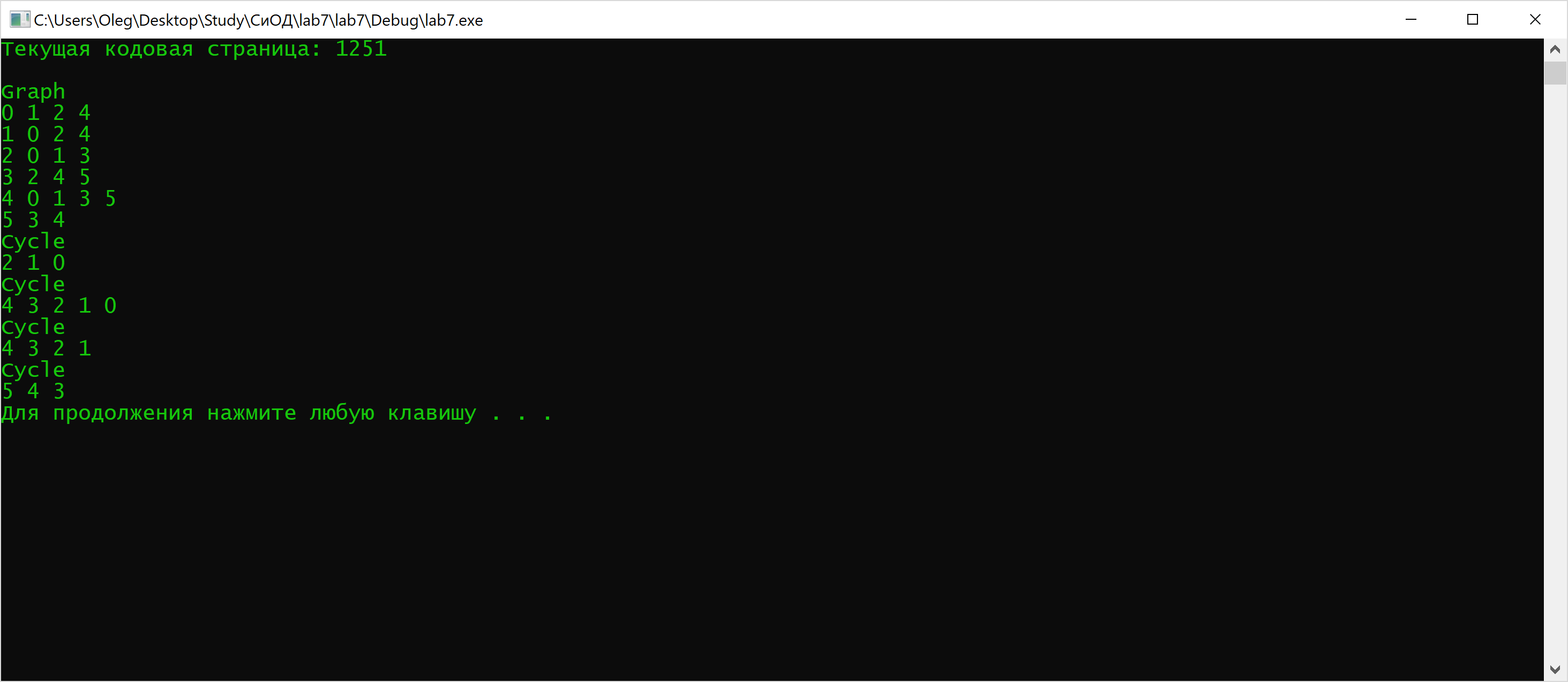


Рисунок №2 – Результат работы программы.

**Вывод**

Я научилась работать с графами. Общая сложность алгоритма O(nm+n).