Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

**ГРАФЫ**

**Отчет по лабораторной работе №7**

**По дисциплине**

**«Структуры и алгоритмы обработки данных в ЭВМ»**

Студентка гр. 430-2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.А. Лузинсан

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Проверил: профессор кафедры АСУ, д.т.н.

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

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Томск 2022

# Задание на лабораторную работу

# Напишите программу, которая будет находить Эйлеров путь в графе. Для представления графа в программе использовать списки смежности. Данные о графе вводятся из файла. Программа должна вывести последовательность вершин графа, составляющих Эйлеров путь. После завершения работы с динамическими структурами данных необходимо освободить занимаемую ими память.

# Алгоритм решения задачи

В ходе реализации алгоритма поиска Эйлерова пути необходимо было использовать АДТ «Стек», поэтому было решено переписать уже реализованный код в прошлом семестре в более подобающий вид. Но перед этим необходимо обработать входные данные так, чтобы способ задания не зависел от результата. В качестве базового представления графа было решено взять списки смежностей. Поэтому и здесь пришлось брать АДТ «Списки», которая была реализована в предыдущем семестре, и немного её модифицировать.

Алгоритм решения самой задачи состоит в следующем:

1. Создаются объекты стек1 и стек2. В стек1 заносится первая вершина;
2. Если в списке смежностей этой вершины есть элементы, то это значит, что существуют рёбра(дуги), по которым мы можем пройти, а значит;
3. Запоминаем вершины, из которой мы выходим, и в которую мы переходим;
4. Добавляем в стек1 вершину, в которую мы переходим;
5. Обращаясь к вершине, из которой мы выходили, удаляем первую смежную с ней вершину, то есть ту, в которую мы входили. Таким образом удаляется дуга из вершины, из которой мы выходили в вершину, в которую мы перешли;
6. Обращаясь к вершине, в которую мы перешли, удаляем вершину, из которой мы вышли. Для этого применяется поиск элемента в списке, так как не каждая вершина, в которую мы бы перешли, содержала в списке смежностей первым элементом именно вершину, из которой мы перешли. Это увеличивает сложность алгоритма;
7. Иначе, если у данной вершины больше нет рёбер(дуг), то перекладываем из стека1 в стек2 элемент.
8. Если стек1 не пуст, переходим к пункту 2.

# Листинг программы

Graph.h:

#pragma once

#include <vector>

#include "ListNode.h"

#include "Stack.h"

namespace luzinsan

{

template <class T>

class Graph

{

private:

std::vector<ListNode<T>> AdjacencyLists; // список смежностей

public:

Graph(const std::vector<ListNode<T>>& List)

{

for (unsigned i = 0; i < List.size(); i++)

AdjacencyLists.push\_back(List[i]);

}

void Euler()

{

Stack<T> stack1, stack2;

stack1.push(AdjacencyLists[0][0]);

int iter = 0;

while (!stack1.isEmpty())

{

if (AdjacencyLists[stack1.getHead() - 1].getSize() != 1)

{

T v = AdjacencyLists[stack1.getHead() - 1][0];

T w = AdjacencyLists[stack1.getHead() - 1][1];

stack1.push(w);

AdjacencyLists[v - 1].deleteNode(&AdjacencyLists[v - 1]);

AdjacencyLists[w - 1].deleteNode(AdjacencyLists[w - 1].find(v));

}

else

{

stack2.push(stack1.getHead());

stack1.pop();

}

std::cout << "\n\nИтерация[" << iter++ << "]: \n";

for (unsigned i = 0; i < AdjacencyLists.size(); i++)

std::cout << AdjacencyLists[i];

std::cout << "\nStack #1:\t";

Stack<T>::print(std::cout, stack1);

std::cout << "\nStack #2:\t";

Stack<T>::print(std::cout, stack2);

}

}

};

}

Stack.h:

#pragma once

#include <iostream>

namespace luzinsan

{

template <class T>

class Stack

{

private:

Stack(const Stack&);

Stack& operator=(const Stack&);

struct Node

{

Node\* next;

T value;

Node(Node\* next, const T& value) : next(next), value(value) {}

};

Node\* base;

size\_t size;

public:

Stack() : base(NULL), size(0) {}

virtual ~Stack() { clear(); }

bool isEmpty() const { return !base; }

size\_t getSize() const { return size; }

const T& getHead() const { return base->value; }

void pop()

{

Node\* node = base;

base = base->next;

delete node;

--size;

}

const T& push(const T& value)

{

base = new Node(base, value);

++size;

return base->value;

}

void clear()

{

while (!isEmpty())

pop();

size = 0;

}

// Печать элементов списка

static std::ostream& print(std::ostream& out, Stack<T>& stack)

{

if (stack.getSize())

{

Stack<T> temp;

do

{

std::cout << stack.getHead() << ' ';

temp.push(stack.getHead());

stack.pop();

} while (!stack.isEmpty());

std::cout << "\n";

do

{

stack.push(temp.getHead());

temp.pop();

} while (!temp.isEmpty());

}

return out;

}

};

}

ListNode.h:

#pragma once

#include <iostream>

#include <cassert>

namespace luzinsan

{

template <class T>

class ListNode

{

private:

T\* \_info; // информационное поле

ListNode<T>\* \_next; // следующий элемент списка

ListNode<T>\* \_endList; // конец списка

size\_t \_size;

public:

ListNode() : \_info{ NULL }, \_next{ NULL }, \_endList{ NULL } { }

ListNode(T info) : ListNode<T>()

{

\_info = new T;

\*\_info = info;

\_size++;

}

ListNode(const ListNode<T>& List) : ListNode<T>()

{

const ListNode<T>\* temp = &List;

while (temp != NULL)

{

append(\*(temp->\_info));

temp = temp->\_next;

}

}

~ListNode()

{

while (this->\_next != NULL) deleteNode(this);

delete \_info;

}

bool isEmpty() { return !\_info; }

ListNode\* getNext() { return \_next; }

size\_t getSize() const { return \_size; }

// Вставка в конец списка

ListNode\* append(T item)

{

ListNode<T>\* currentItem = new ListNode<T>(item);

\_size++;

if (!\_info)

{

\*this = \*currentItem;

\_endList = this;

}

else

{

\_endList->\_next = currentItem; // добавляем после последнего элемента в списке новый элемент

\_endList = \_endList->\_next; // перемещаем указатель на новый последний элемент в списке

}

return this;

}

// Вставка нового элемента списка после текущего, либо вставка первого элемента в начало

ListNode\* insert(ListNode<T>\* p, T i)

{

\_size++;

ListNode<T>\* q = new ListNode<T>;

assert(q && "Память не выделилась!!!");

q->l\_info = i;

if (!p) // если список ещё не заполнен ни одним элементом

p = q;

else

{

q->l\_next = p->l\_next;

p->l\_next = q;

}

return this;

}

// Поиск элемента списка по его значению. Возвращается указатель на предыдущий перед найденным элементом элемент.

// Если найдённый элемент первый по списку, то возвращается указатель на него самого.

ListNode<T>\* find(const T& element)

{

ListNode<T>\* prev\_temp = this;

if (\*(prev\_temp->\_info) == element)

return prev\_temp;

const ListNode<T>\* curr\_temp = this;

for (int i = 1; i < \_size; i++)

{

curr\_temp = curr\_temp->\_next;

if(\*(curr\_temp->\_info) == element)

return prev\_temp;

prev\_temp = prev\_temp->\_next;

}

return NULL;

}

T& operator[](int index) const

{

const ListNode<T>\* temp = this;

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

temp = temp->\_next;

return \*(temp->\_info);

}

//Удаление следующего элемента после текущего

T deleteNode(ListNode<T>\* p)

{

if (p && p->\_next)

{

\_size--;

T val = \*(p->\_next->\_info);

delete p->\_next->\_info;

p->\_next = p->\_next->\_next;

return val;

}

else return -1;

}

// Печать элементов списка

friend std::ostream& operator<<(std::ostream& out, const ListNode<T>& node)

{

const ListNode<T>\* temp = &node;

std::cout << "\_|" << \*(temp->\_info) << "| -> ";

temp = temp->\_next;

while (temp)

{

std::cout << \*(temp->\_info) << " -> ";

temp = temp->\_next;

}

std::cout << "NULL\n\_|\_\n";

return out;

}

const ListNode<T>& operator=(T item)

{

if (!\_info) \_info = new T;

\*\_info = item;

return \*this;

}

};

}

lr7.cpp:

#include <iostream>

#include <fstream>

#include <conio.h>

#include <vector>

#include "Graph.h"

using namespace luzinsan;

char getSymbol(std::initializer\_list<char> list,

std::string notification\_message = "",

std::string error\_message = "Недопустимое значение, попробуйте ещё раз.\n->");

int main()

{

setlocale(LC\_ALL, "Rus");

std::cout

<< "\n\t-> Файл: IncidentMatrixForADirectedGraph.txt \t\t-> Матрица инциденций для ориентированного графа \t- 1;"

<< "\n\t-> Файл: IncidentMatrixForAnUndirectedGraph.txt \t-> Матрица инциденций для неориентированного графа \t- 2;"

<< "\n\t-> Файл: AdjacencyMatrixForADirectedGraph.txt \t\t-> Матрица смежности для ориентированного графа \t- 3;"

<< "\n\t-> Файл: AdjacencyMatrixForAnUndirectedGraph.txt \t-> Матрица смежности для неориентированного графа \t- 4;"

<< "\n\t-> Файл: CoupleListsForADirectedGraph.txt \t\t-> Списки пар для ориентированного графа \t\t- 5;"

<< "\n\t-> Файл: CoupleListsForAnUndirectedGraph.txt \t\t-> Списки пар для неориентированного графа \t\t- 6;"

<< "\n\t-> Файл: AdjacencyListsForADirectedGraph.txt \t\t-> Списки смежности для ориентированного графа \t\t- 7;"

<< "\n\t-> Файл: AdjacencyListsForAnUndirectedGraph.txt \t-> Списки смежности для неориентированного графа \t- 8.";

char choice = getSymbol({ '1','2','3','4','5','6','7','8' }, "\n\n\tВыберите способ задания графа:\n->");

int numberOfVertices, // количество вершин

numberOfEdges; // количество рёбер/дуг

std::ifstream fin;

std::vector<ListNode<int>> AdjacencyLists;

// Открытие файла

switch (choice)

{

case '1':// Матрица инциденций для ориентированного графа

fin.open("IncidentMatrixForADirectedGraph.txt");

break;

case '2':// Матрица инциденций для неориентированного графа

fin.open("IncidentMatrixForAnUndirectedGraph.txt");

break;

case '3'://Матрица смежности для ориентированного графа

fin.open("AdjacencyMatrixForADirectedGraph.txt");

break;

case '4'://Матрица смежности для неориентированного графа

fin.open("AdjacencyMatrixForAnUndirectedGraph.txt");

break;

case '5'://Списки пар для ориентированного графа

fin.open("CoupleListsForADirectedGraph.txt");

break;

case '6'://Списки пар для неориентированного графа

fin.open("CoupleListsForAnUndirectedGraph.txt");

break;

case '7'://Списки смежности для ориентированного графа

fin.open("AdjacencyListsForADirectedGraph.txt");

break;

case '8'://Списки смежности для неориентированного графа

fin.open("AdjacencyListsForAnUndirectedGraph.txt");

break;

}

fin >> numberOfVertices >> numberOfEdges;

switch (choice)

{

case '1':case '2':case '3':case '4':

{

int\* matrix;

if (choice == '1' || choice == '2')

{

matrix = new int[numberOfVertices \* numberOfEdges];

for (unsigned i = 0; i < numberOfVertices \* numberOfEdges; i++)

fin >> matrix[i];

if(choice == '1') // Матрица инциденций для ориентированного графа

for (unsigned i = 0; i < numberOfVertices; i++)

{

AdjacencyLists.push\_back(ListNode<int>(i + 1));

for (unsigned j = 0; j < numberOfEdges; j++)

if (matrix[i \* numberOfEdges + j] == 1)

for (unsigned k = 0; k < numberOfVertices; k++)

if (matrix[k \* numberOfEdges + j] == -1)

{

AdjacencyLists[i].append(k + 1);

break;

}

}

else // Матрица инциденций для неориентированного графа

for (unsigned i = 0; i < numberOfVertices; i++)

{

AdjacencyLists.push\_back(ListNode<int>(i + 1));

for (unsigned j = 0; j < numberOfEdges; j++)

if (matrix[i \* numberOfEdges + j] == 1)

for (unsigned k = 0; k < numberOfVertices; k++)

if (matrix[k \* numberOfEdges + j] == 1 && k != i)

{

AdjacencyLists[i].append(k + 1);

break;

}

}

}

else // Матрица смежности для (не)ориентированного графа

{

matrix = new int[numberOfVertices \* numberOfVertices];

for (unsigned i = 0; i < numberOfVertices \* numberOfVertices; i++)

fin >> matrix[i];

for (unsigned i = 0; i < numberOfVertices; i++)

{

AdjacencyLists.push\_back(ListNode<int>(i + 1));

for (unsigned j = 0; j < numberOfVertices; j++)

if (matrix[i \* numberOfVertices + j] == 1)

AdjacencyLists[i].append(j + 1);

}

}

delete[] matrix;

break;

}

case '5': case '6': // Списки пар для ориентированного графа

for (unsigned i = 0; i < numberOfVertices; i++)

AdjacencyLists.push\_back(ListNode<int>(i + 1));

for (unsigned i = 0; i < numberOfEdges; i++)

{

unsigned index, vertex;

fin >> index >> vertex;

AdjacencyLists[index-1].append(vertex);

if(choice == '6')

AdjacencyLists[vertex - 1].append(index); // Симметричная пара для неориентированного графа

}

break;

case '7':case '8': // Списки смежности для (не)ориентированного графа

for(unsigned i = 0; i < numberOfVertices; i++)

{

unsigned index, vertex;

fin >> index;

AdjacencyLists.push\_back(ListNode<int>(index));

while (fin >> vertex && vertex != 0)

AdjacencyLists[index - 1].append(vertex);

}

break;

default: std::cerr << "\n\t\t\tУпс, что-то пошло не так...";

}

if (fin)

{

std::cout << "\n\tСформированный список смежностей:\n";

for (auto& element : AdjacencyLists)

std::cout << element;

}

else std::cerr << "Ошибка открытия файла.";

Graph<int> graph(AdjacencyLists);

graph.Euler();

return 0;

}

char getSymbol(std::initializer\_list<char> list,

std::string notification\_message,

std::string error\_message)

{

char choice;

std::cout << notification\_message;

bool flag = true;

do {

choice = \_getche();

std::cout << std::endl;

for (auto it = list.begin(); it != list.end(); it++)

if (it[0] == choice) { flag = false; break; }

if (flag) std::cerr << error\_message;

} while (flag);

return choice;

}

# Пример решения

На выбор пользователю доступно 8 вариантов задания графа в ЭВМ, представленных в файлах (рис 4.1):

1. IncidentMatrixForADirectedGraph.txt -> Матрица инциденций для ориентированного графа ->1;
2. IncidentMatrixForAnUndirectedGraph.txt -> Матрица инциденций для неориентированного графа -> 2;
3. AdjacencyMatrixForADirectedGraph.txt -> Матрица смежности для ориентированного графа -> 3;
4. AdjacencyMatrixForAnUndirectedGraph.txt -> Матрица смежности для неориентированного графа -> 4;
5. CoupleListsForADirectedGraph.txt -> Списки пар для ориентированного графа -> 5;
6. CoupleListsForAnUndirectedGraph.txt -> Списки пар для неориентированного графа -> 6;
7. AdjacencyListsForADirectedGraph.txt -> Списки смежности для ориентированного графа -> 7;
8. AdjacencyListsForAnUndirectedGraph.txt -> Списки смежности для неориентированного графа -> 8.

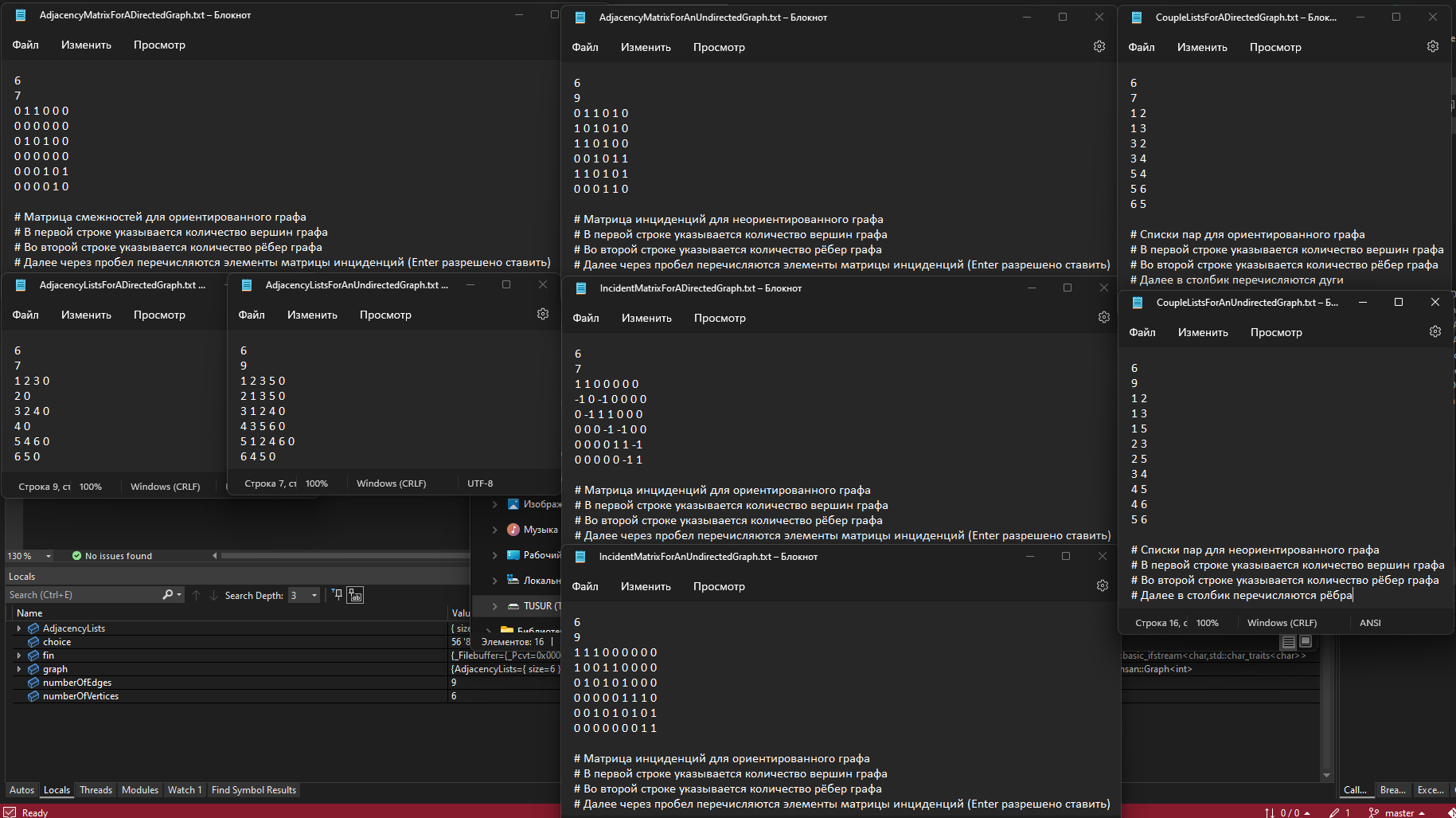


Рисунок 4.1 – Входные данные программы всех файлов

В результате многочисленных тестирований, было подтверждено, что исходные данные, если они были заданы правильно, верно преобразовываются в списки смежностей. Далее, на рисунке 4.2, показан результат работы программы при заданной матрице инциденций неориентированного графа.

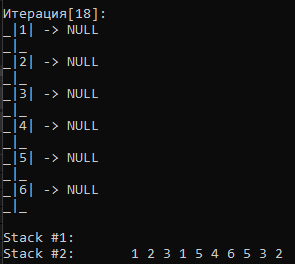


Рисунок 4.2 – Выходные данные программы при входном файле IncidentMatrixForAnUndirectedGraph.txt

# Вывод

В результате данной лабораторной работы я изучила АТД «Граф» и реализовала алгоритм поиска Эйлерова пути.