Московский авиационный институт

(Национальный исследовательский университет)

институт “Информационные технологии и прикладная математика”

кафедра “Математическая кибернетика”

КУРСОВАЯ РАБОТА

*по курсу* «Дискретная математика»

*2-й семестр*

|  |  |
| --- | --- |
| Тема: | Теория графов, алгебраические структуры, теория алгоритмов.  Построение базы циклов на основе поиска в глубину. |

|  |  |
| --- | --- |
| *Студент*: | Агеева Алиса Ивановна |
| *Группа*: | М8О-110Б |
| *Руководитель*: | Н.С. Алексеев |
| *Оценка*: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| *Дата*: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

### 

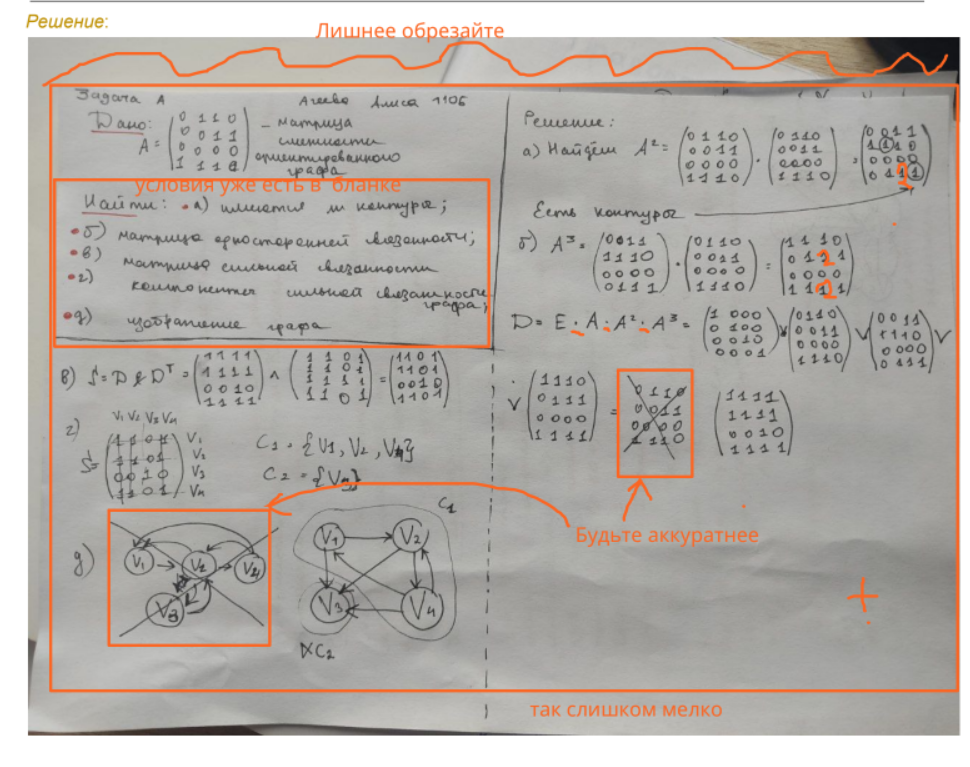
### Введение

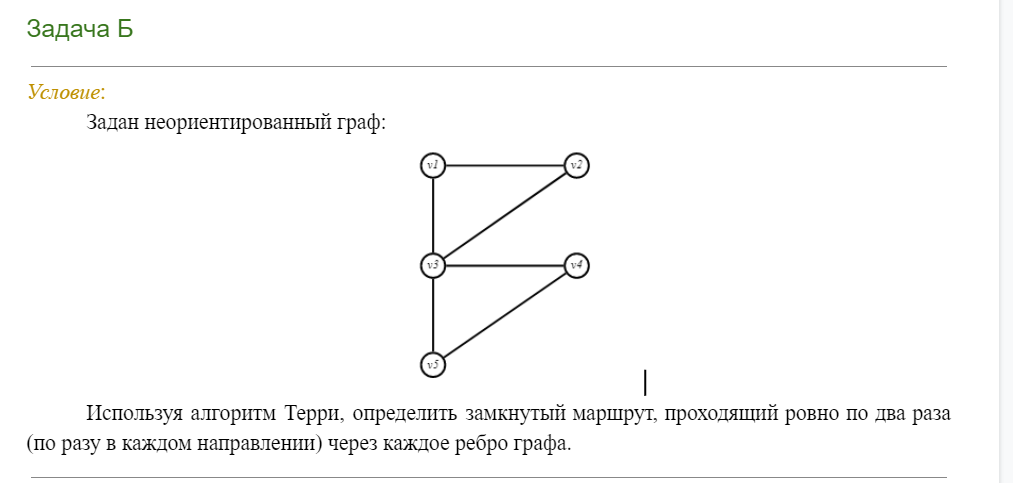
В настоящем отчете по курсовой работе приведены результаты, полученные в рамках изучения курса “Дискретная математика” во втором учебном семестре.

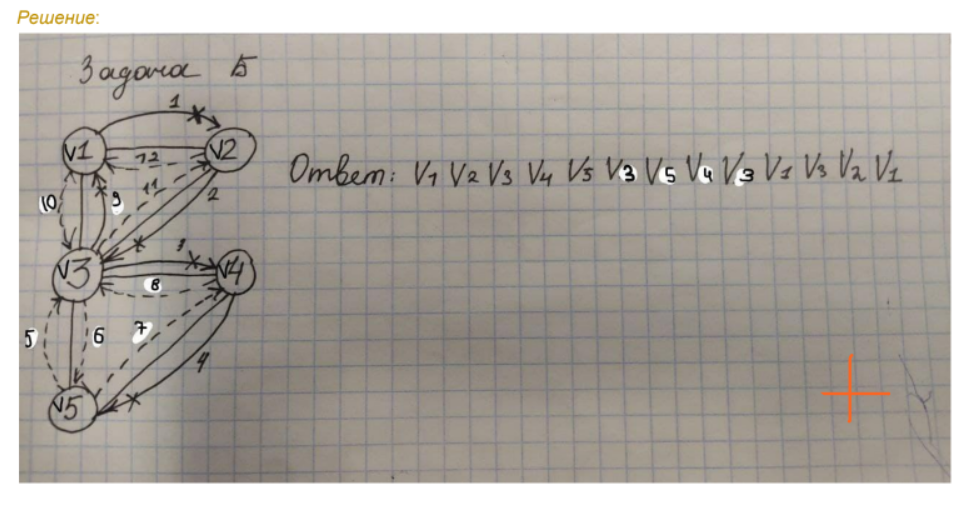
В части I приведены решения типовых задач по теории графов, а также представлена реализация алгоритма построения базы циклов графа на основе поиска в глубину. Текст программы и тестовые примеры вынесены в Приложение.

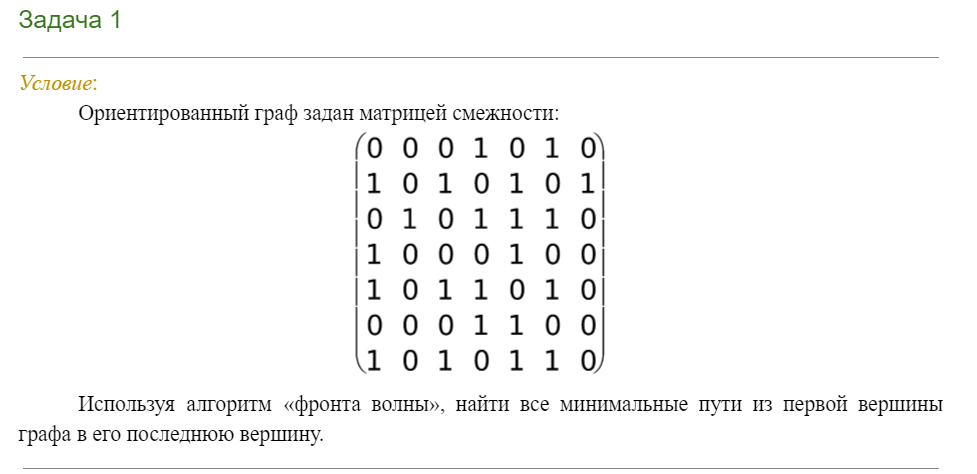
В части II приведены решения типовых задач по темам “Алгебраические структуры” и “Теория алгоритмов”.

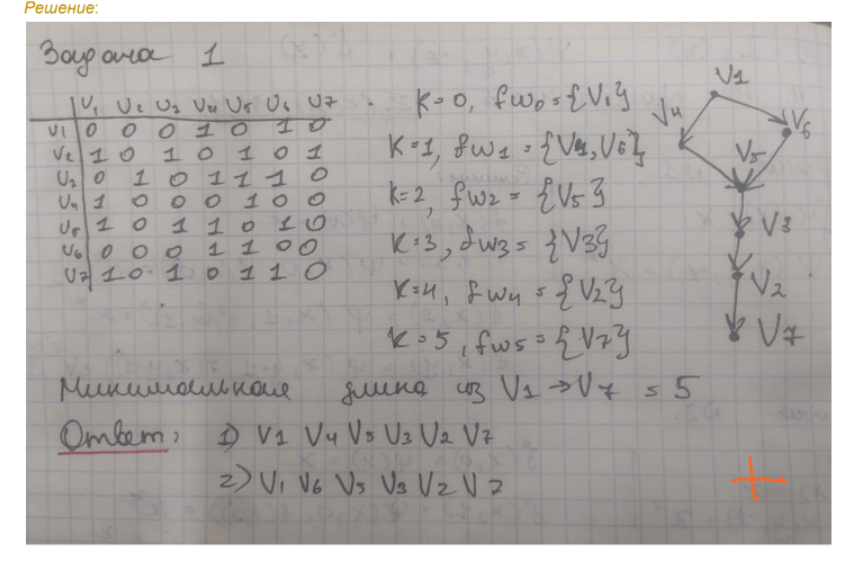
### Часть I. Теория графов

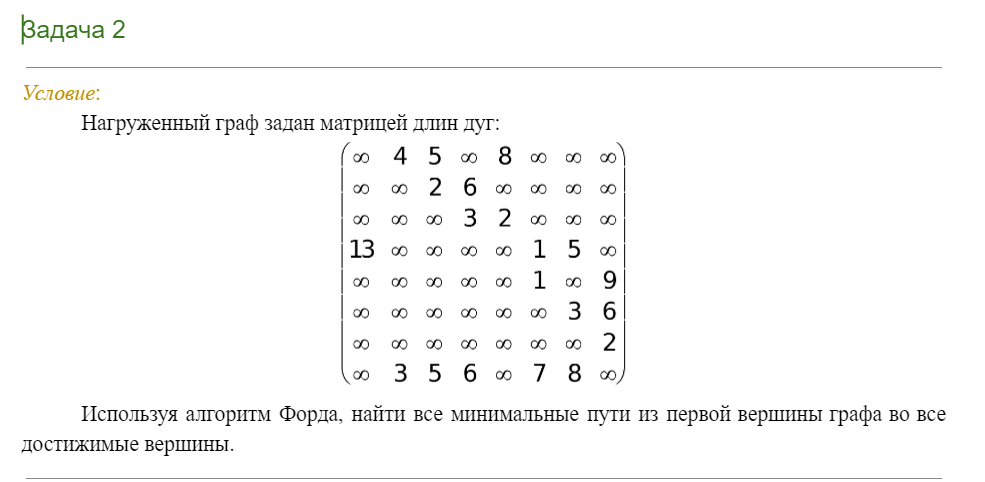


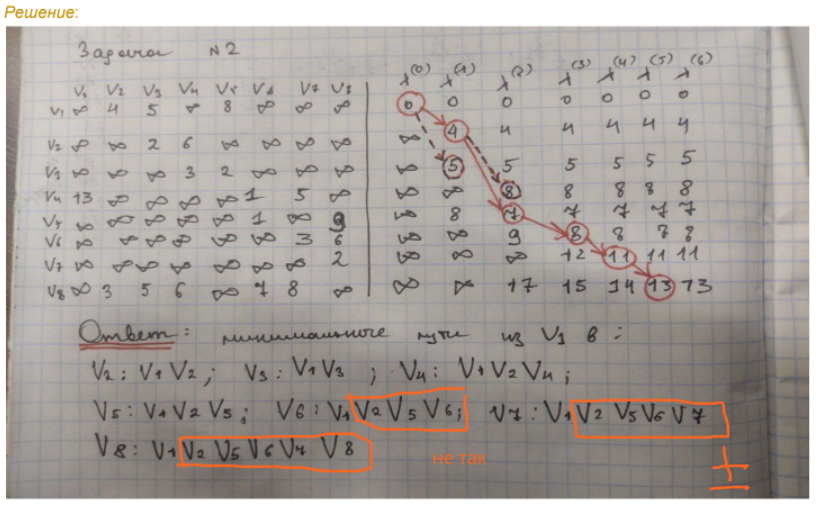


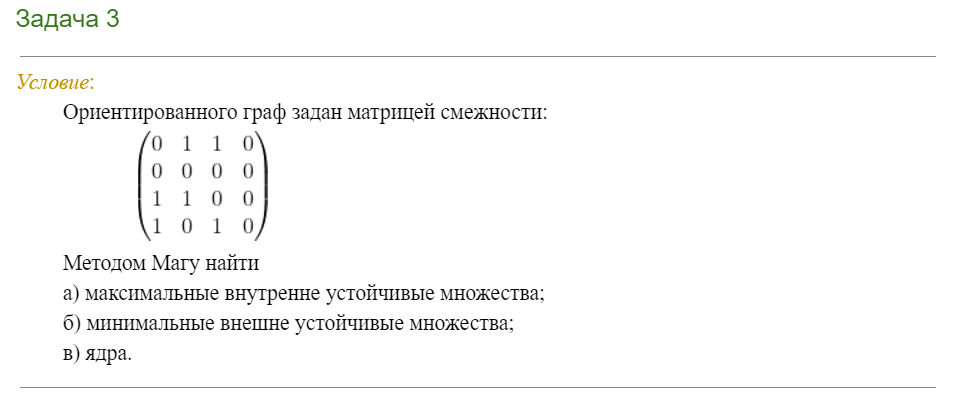


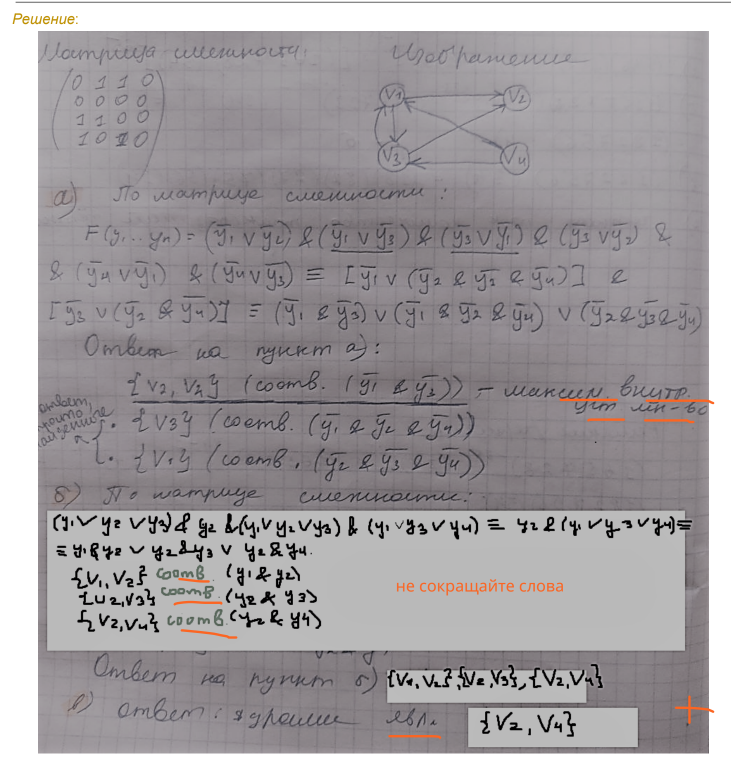


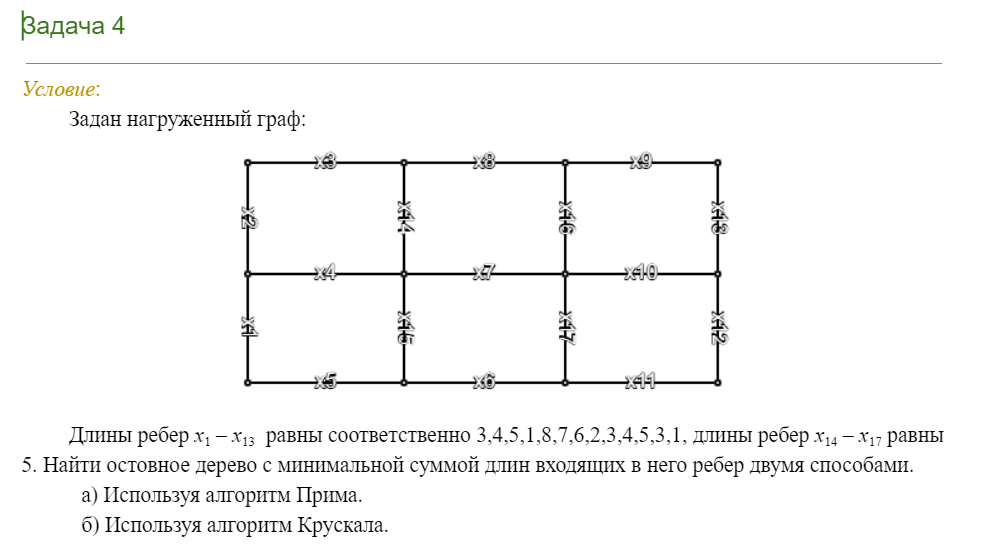




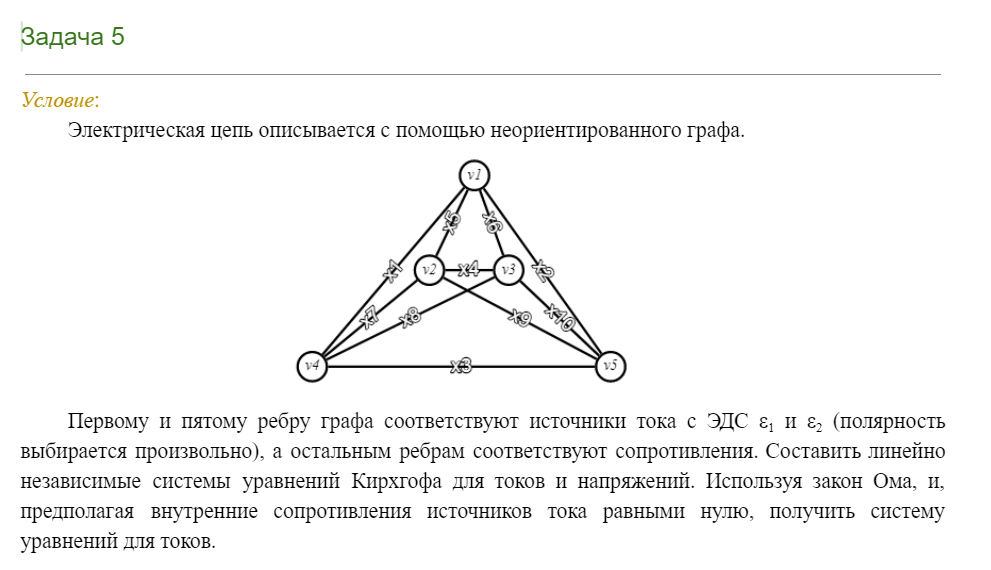


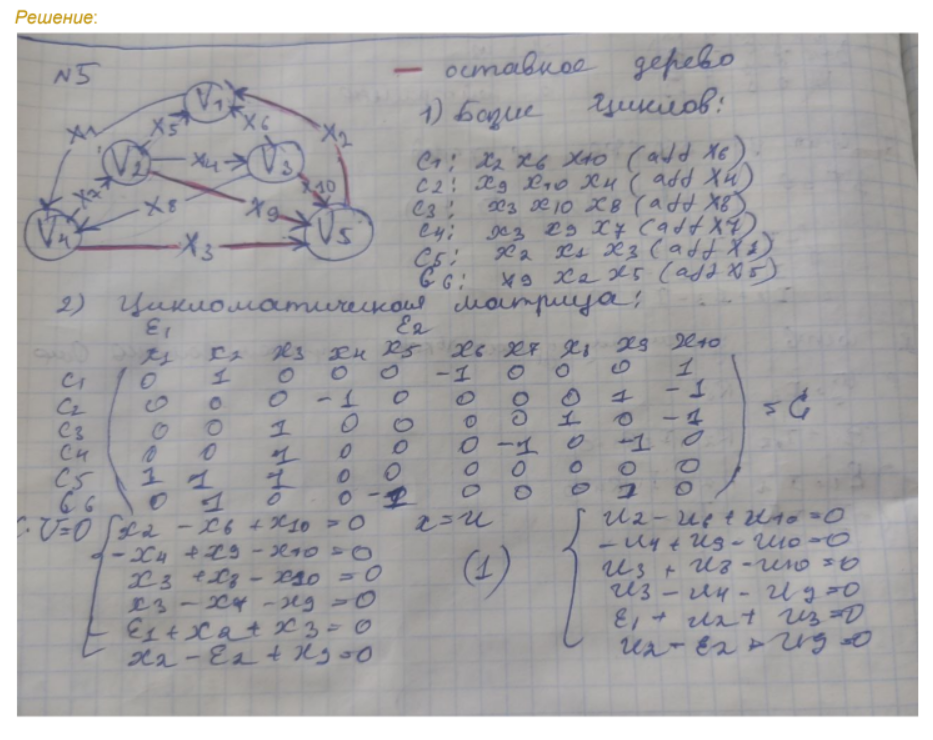


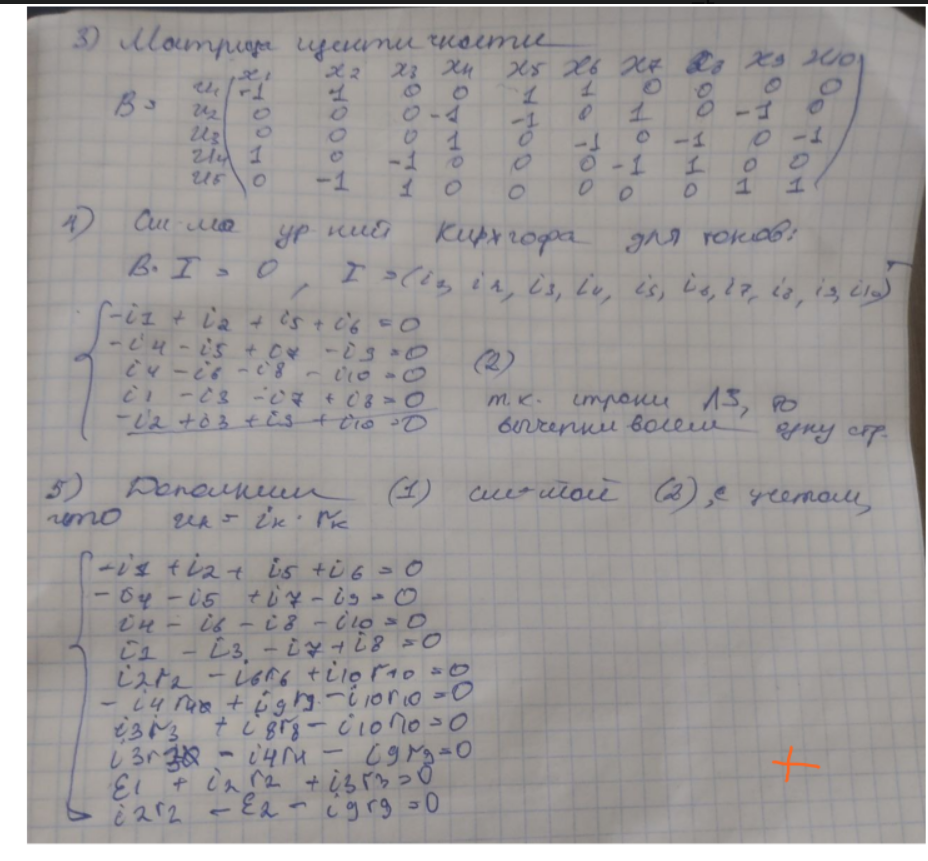


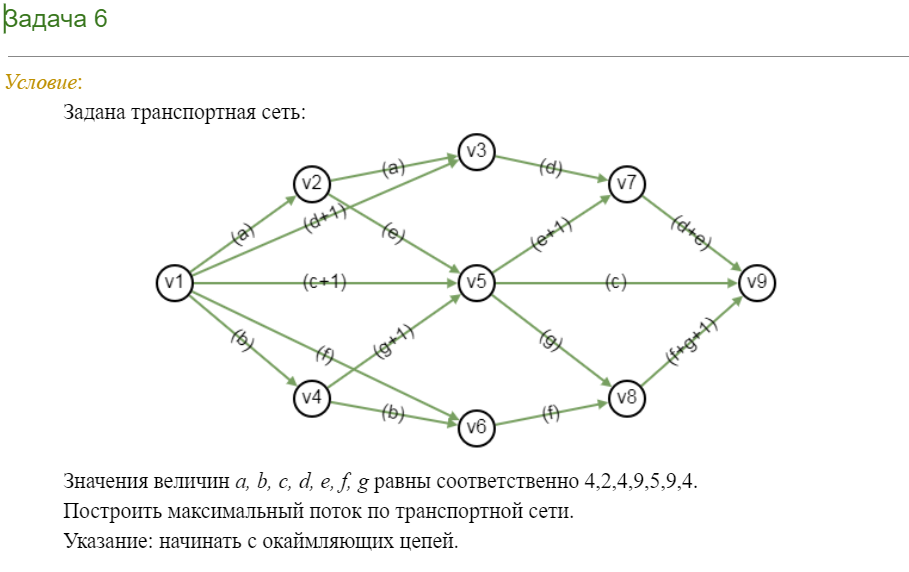


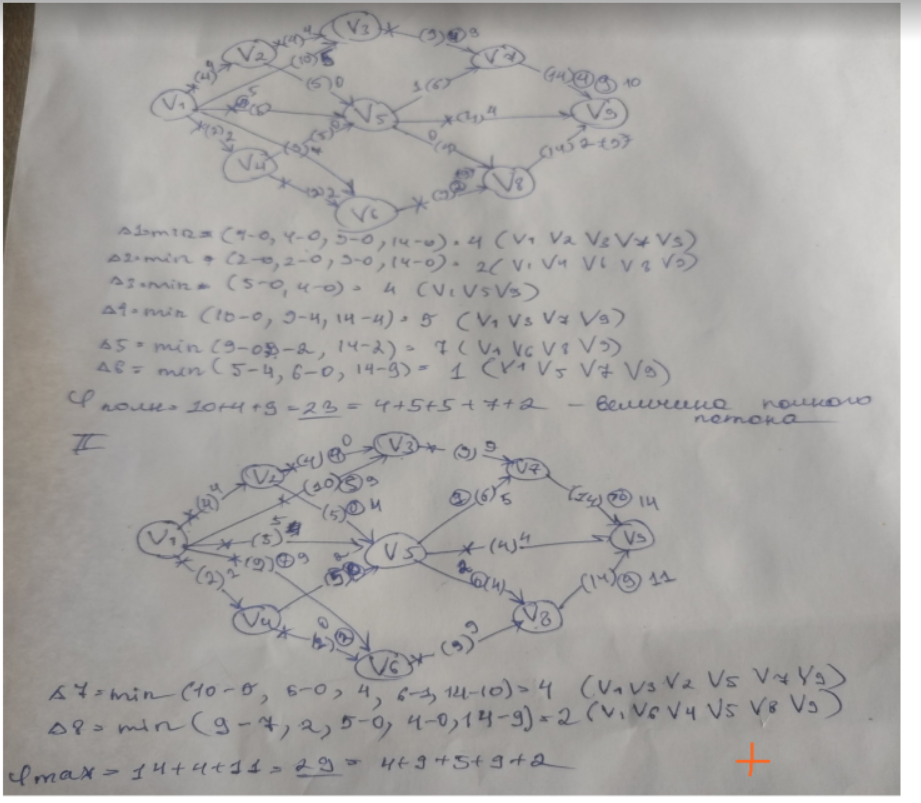












### Часть II. Теория алгоритмов и алгебраические структуры

### 

### 

### 

### 

### Часть III. Программная реализация алгоритма Построение базы циклов на основе поиска в глубину

#### Описание алгоритма.

Построить базис пространства, состоящий из простых циклов, можно следующим образом. Выберем в графе G какой-нибудь каркас T. Пусть e1 … es— все ребра графа G, не принадлежащие T. Если добавить к T ребро ei, то в полученном графе образуется единственный (простой) цикл Zi. Таким образом, получаем семейство из s циклов, они называются фундаментальными циклами относительно каркаса T.

Поиск в глубину для этой задачи особенно удобен благодаря основному свойству DFS-дерева — каждое обратное ребро относительно этого дерева является продольным. Это означает, что из двух вершин такого ребра одна является предком другой в DFS-дереве. Каждое такое ребро в процессе поиска в глубину встретится дважды — один раз, когда активной вершиной будет предок, другой раз, когда ею будет потомок. В этом последнем случае искомый фундаментальный цикл состоит из рассматриваемого обратного ребра и участка пути в DFS-дереве, соединяющего эти две вершины. Но этот путь так или иначе запоминается в процессе обхода в глубину, так как он необходим для последующего возвращения.

#### Описание реализации алгоритма

1. Введём граф из текстового файла;

2. Пометим все вершины графа, как новые, и для каждой вершины графа, если она новая,

вызовем процедуру cycleBase, передавая в неё нужные массивы и текущую вершину а.

3. Покрасим рёбра, входящие в циклы, либо выведем сообщение о том, что циклов в графе нет.

*Процедура cycleBase:*

1. Открываем вершину а и записываем её в качестве своего предка;

2. Записываем значение а в переменную curNode

3. Пока curNode открыта:

a. Посещаем вершину curNode и получаем вершину nextNode, которая связана с

curNode;

b. Если ребро (curNode, nextNode) не исследовано, то

i. Помечаем ребро, как исследованное;

ii. Если nextNode — новая вершина, то

1. Открываем вершину nextNode, устанавливаем ей в качестве родителя curNode и перезаписываем в curNode вершину nextNode.

iii. Иначе

1. Вызываем процедуру newCycle, формирующую базисный цикл.

c. Иначе, если неисследованных рёбер у вершины нет, то

i. Закрываем curNode и перезаписываем в неё её родителя.

Процедура *newCycle,* принимающая в качестве аргументов необходимые массивы, вершину х

и у:

1. Инкрементируем счётчик базисных циклов k;

2. Создаем новый список базисного цикла и добавляем туда curNode = х;

3. Повторяем

a. Перезаписываем в curNode её родителя;

b. Добавляем curNode в список.

4. Пока curNode не равна y;

5. Добавляем полученный список в вектор списков.

#### Описание программы и инструкции по работе с ней

Работа с программой осуществляется при помощи ГРАФОИДа. Необходимо построить в нём неориентированный нагруженный граф, а затем выбрать и запустить исполняемый файл программы. Она покрасит все базовые циклы графа, либо выведет сообщение о том, что их нет.

#### *Оценка вычислительной сложности алгоритма*

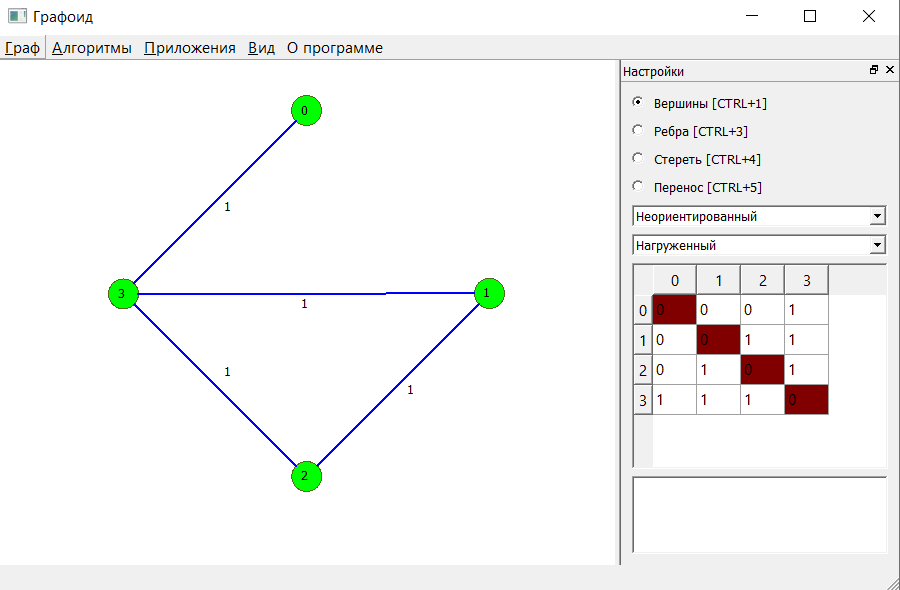
#### Хотя сам поиск в глубину выполняется за линейное от числа вершин и ребер время, решающее влияние на трудоемкость этого алгоритма оказывает необходимость запоминать встречающиеся циклы. Подсчитаем суммарную длину этих циклов для полного графа с n вершинами. DFS-дерево в этом случае является простым путем, относительно него будет n - 2 цикла длины 3, n - 3 цикла длины 4, . . . 1 цикл длины n. Сумма длин всех фундаментальных циклов будет равна



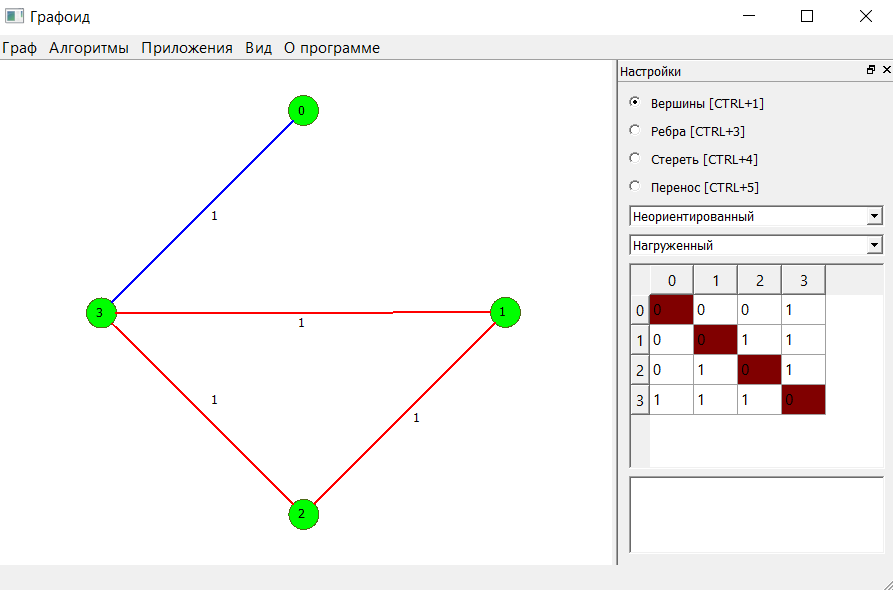
Таким образом, на некоторых графах число операций этого алгоритма будет величиной порядка

**● Тестовые примеры с решением и скриншотами.**

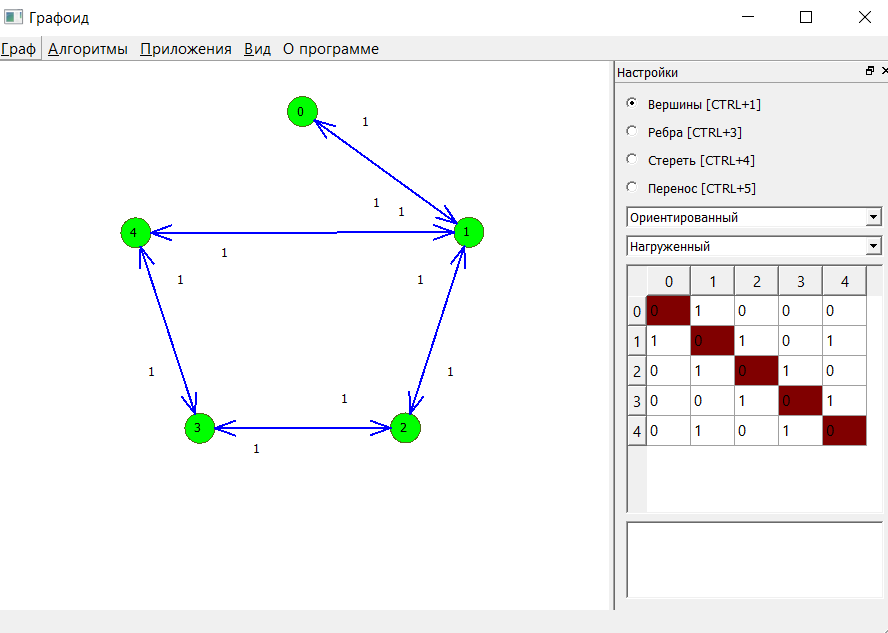
Пример 1

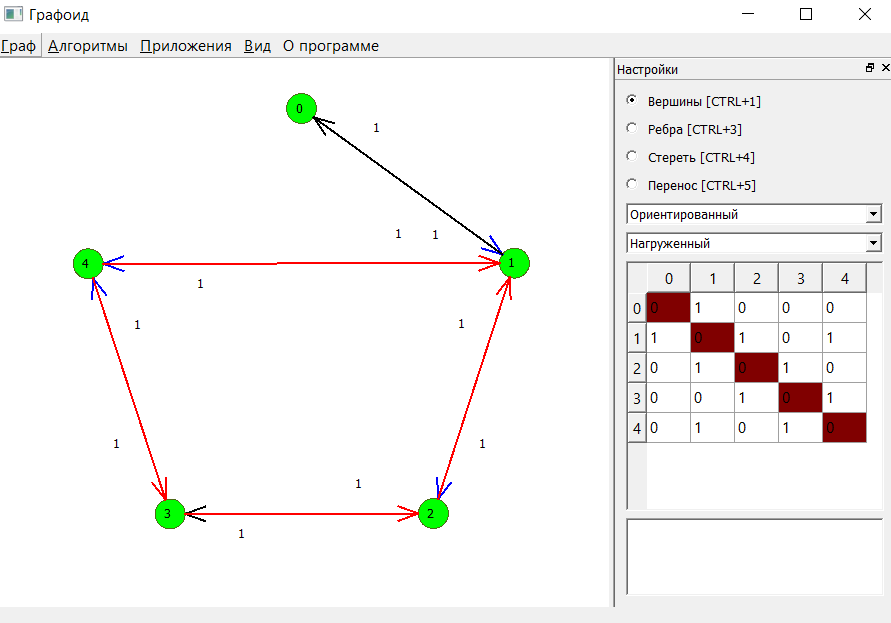
******

#### 

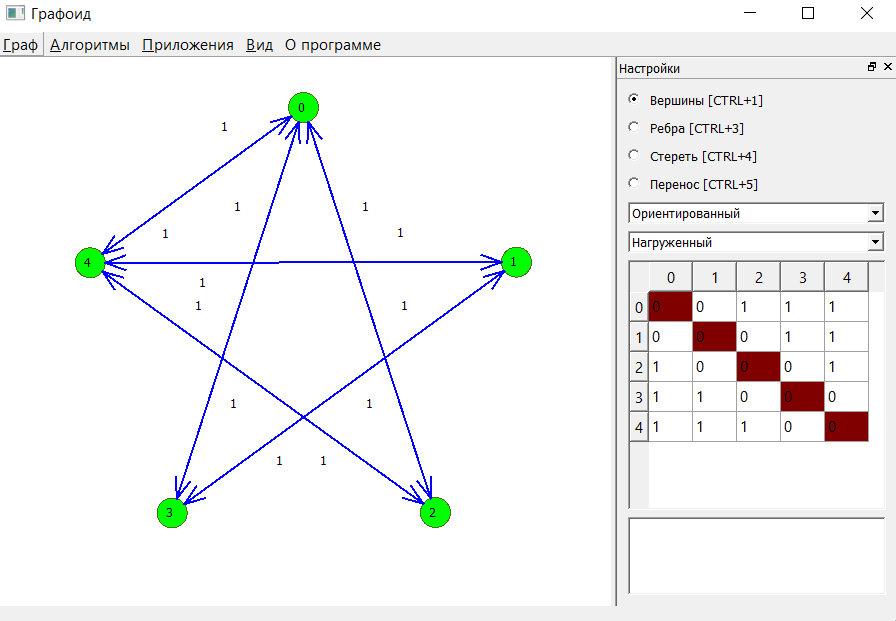


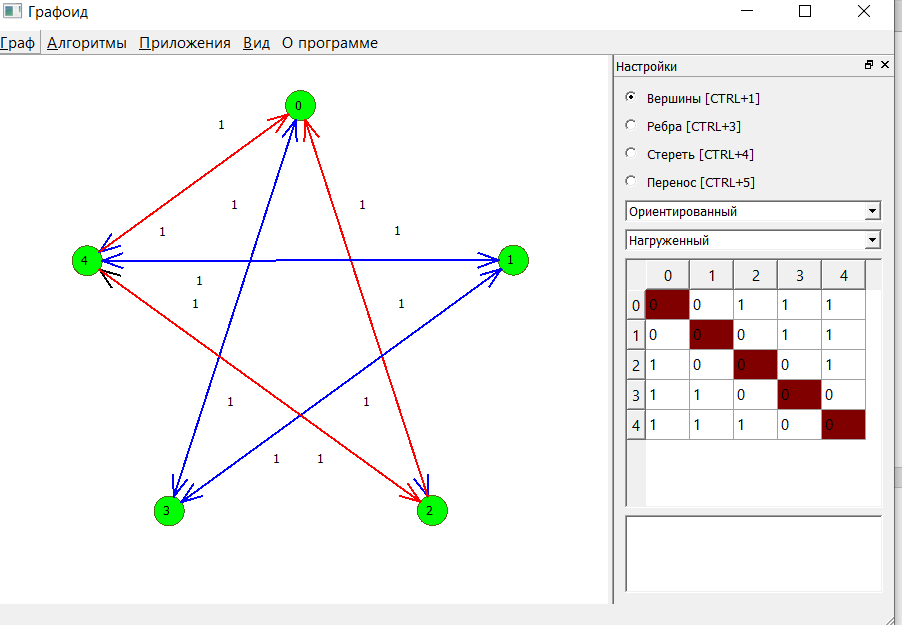
#### Пример 2



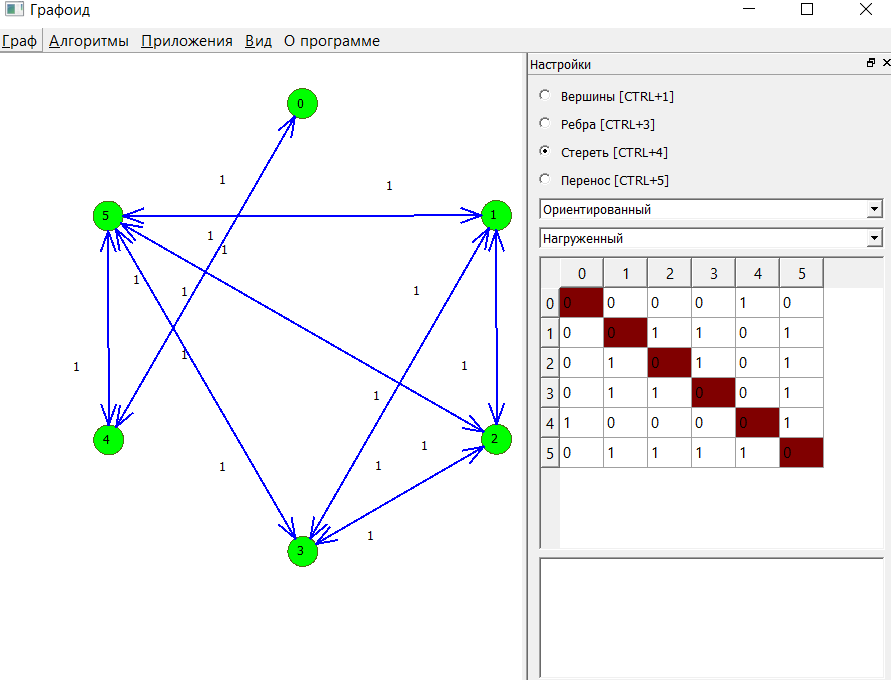


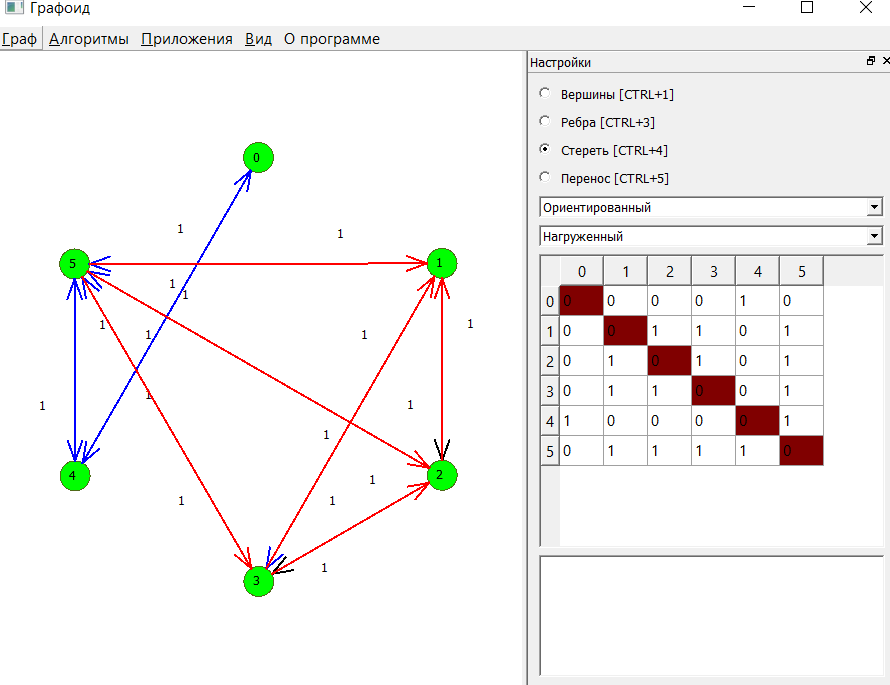
#### Пример 3



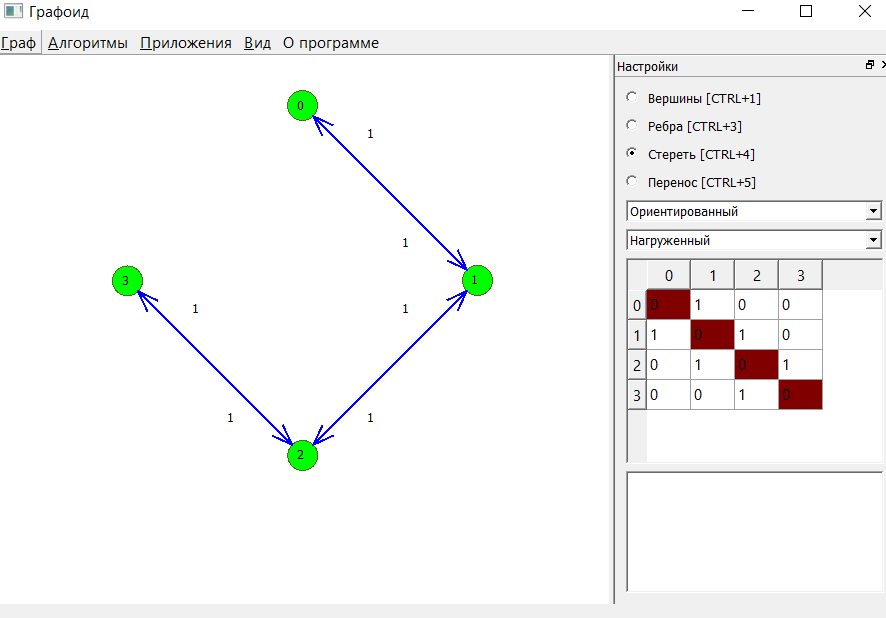


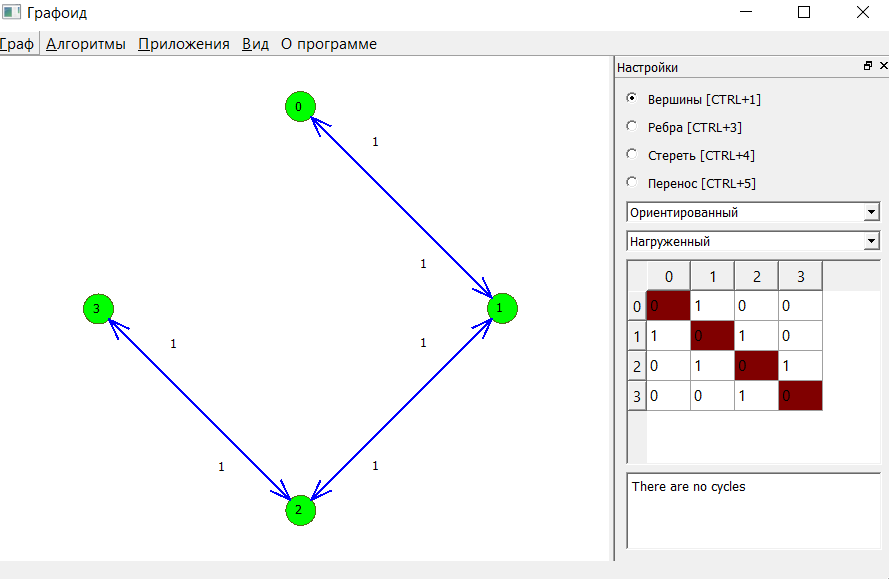
Пример 4





Пример 5





#### Пример прикладной задачи

Алгоритм может быть использован для решения транспортных задач, в которых вершинами графа являются города, а ребрами – дороги (автомобильные, железные и др.)

### Заключение

В ходе выполнения курсовой работы были решены 12 задач по различным разделам курса “Дискретная математика”.

Кроме того, был изучен вопрос о различных методах поиска путей и маршрутов в графах. Была написана и отлажена программа, реализующая построение базы циклов на основе поиска в глубину. Программа написана на языке программирования Си. Программа обеспечивает связь по установленному формату с системой ГРАФОИД, разработанной на кафедре 805, что дает возможность обеспечить графический интерфейс при ее использовании. Эта программа является основным результатом курсового проектирования.

## 

## **Список использованных источников**

* 1. *В.А. Таланов, В.Е. Алексеев.* Графы и алгоритмы. <https://www.intuit.ru> .
* 2. *Нефедов В.Н., Осипова В.А.* Курс дискретной математики. МАИ, 1992

### Приложение 1. Текст программы

#include <cstdlib>

#include <iostream>

#include <fstream>

#include <vector>

using namespace std;

void cycleBase(vector<vector<int>>& graph, vector<vector<bool>>& exploredEdges,

vector<bool>& openedNodes, vector<bool>& newNodes, vector<int>& parents, int a, int&

cycleAmount, vector<vector<int>>& basicCycles, fstream& out,vector<int>& used);

void newCycle(vector<vector<int>>& basicCycles, vector<int>& parents, int& cycleAmount, int x, int y, fstream& out);

int getUnexploredEdges(vector<vector<bool>>& exploredEdges, int node);

vector<vector<int>> matrix;

int main(int argc, char \*argv[])

{

//Loading matrix

ifstream in(argv[1]);

int size=0;

in >> size;

matrix.resize(size);

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

matrix[i].resize(size);

}

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

for(int j=0;j<size;j++) {

in >> matrix[i][j];

}

}

in.close();

//Algorithm

// заполнение списка смежности

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

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

for(int j=0;j<size;j++) {

if (matrix[i][j] > 0) {

        graph[i].push\_back(j);

}

}

}

vector<bool> newNodes(size, true);

vector<bool> openedNodes(size, false);

vector<vector<bool>> exploredEdges(size, vector<bool>(size, false));

vector<int> parents(size);

vector<int> used(size, 0);

vector<vector<int>> basicCycles;

int cycleAmount = 1;

fstream out;

out.open(argv[1]);

out.clear();

char buffer [33];

itoa(size,buffer,10);

out << buffer << "\n";

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

for(int j=0;j<size;j++) {

out << matrix[i][j];

if(j!=size-1) out << " ";

}

out << "\n";

}

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

if (newNodes[i]){

cycleBase(graph, exploredEdges, openedNodes, newNodes, parents, i, cycleAmount,

basicCycles, out,used);

}

}

if (!basicCycles.empty()) {

vector<vector<int>> paintedNodes(size, vector<int>(size, 0)); // 1 -- Blue, 2 -- Red

for (int i = 0; i < basicCycles.size(); ++i) {

for (int j = 0; j < basicCycles[i].size() - 1; ++j) {

if (j == 0){

paintedNodes[basicCycles[i][j]][basicCycles[i][j+1]] = 2;

paintedNodes[basicCycles[i][j+1]][basicCycles[i][j]] = 2;

paintedNodes[basicCycles[i][j]][basicCycles[i][basicCycles[i].size() - 1]] = 2;

paintedNodes[basicCycles[i][basicCycles[i].size() - 1]][basicCycles[i][j]] = 2;

}

else{

paintedNodes[basicCycles[i][j]][basicCycles[i][j+1]] = 2;

paintedNodes[basicCycles[i][j+1]][basicCycles[i][j]] = 2;

}

}

}

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

for (int j = 0; j < size; ++j){

if (matrix[i][j] != 0 && paintedNodes[i][j] != 2){

paintedNodes[i][j] = 1;

paintedNodes[j][i] = 1;

}

}

}

out << "Colors\_Edges:\n";

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

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

if (matrix[i][j] != 0) {

out << i << ' ' << j << ' ';

switch (paintedNodes[i][j]) {

case 1:

out << "Blue" << "\n";

break;

case 2:

out << "Red" << "\n";

break;

}

}

}

}

}

out << "Text:\n";

if (!basicCycles.empty()) {

for (int i = 0; i < basicCycles.size(); ++i) {

out << "c" << i <<": ";

for (int j = 0; j < basicCycles[i].size(); ++j) {

out << basicCycles[i][j];

if (j != basicCycles[i].size() - 1)

out << ' ';

}

out << "\n";

}

}

else

out << "There are no cycles";

out.close();

}

void cycleBase(vector<vector<int>>& graph, vector<vector<bool>>& exploredEdges,

vector<bool>& openedNodes,

vector<bool>& newNodes, vector<int>& parents, int a, int& cycleAmount,

vector<vector<int>>& basicCycles,fstream& out,vector<int>& used){

openedNodes[a] = true;

parents[a] = a;

int curNode = a;

int curIter = 0;

while (openedNodes[curNode]){

++curIter;

if (curIter == 25){

out.close();

return;

}

newNodes[curNode] = false;

int nextNode;

if (!graph[curNode].empty()){

nextNode = graph[curNode].back(); // потенциальная вершина для дальнейшего ДФС (в алгоритме это y)

graph[curNode].pop\_back();

}

else{

return;

}

// здесь нужно сделать curNode не новой. Новая -- ещё не посещённая

if (!exploredEdges[curNode][nextNode]){

exploredEdges[curNode][nextNode] = true;

exploredEdges[nextNode][curNode] = true;

if (newNodes[nextNode]){

openedNodes[nextNode] = true; // открываем вершину у и делаем её не новой

parents[nextNode] = curNode;

curNode = nextNode;

}

else {

newCycle(basicCycles, parents, cycleAmount, curNode, nextNode, out); // здесь nextNode - это начало цикла, из которого можно вернуться к предыдущим вершинам

}

}

else if (getUnexploredEdges(exploredEdges, curNode) == 0) { // если неисследованных рёбер у вершины нет

openedNodes[curNode] = false;

curNode = parents[curNode];

}

}

}

void newCycle(vector<vector<int>>& basicCycles, vector<int>& parents, int& cycleAmount, int x, int

y, fstream& out){

++cycleAmount;

vector<int> newCycle;

newCycle.push\_back(x);

int curNode = x, curIter = 0; // curNode здесь -- это z

do{

if (curIter == 25){

return;

}

curNode = parents[curNode];

//parents.erase(parents.begin());

newCycle.push\_back(curNode);

++curIter;

} while (curNode != y);

basicCycles.push\_back(newCycle);

}

int getUnexploredEdges(vector<vector<bool>>& exploredEdges, int node){

int ans = 0;

int size = matrix[0].size();

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

if (!exploredEdges[node][i] && matrix[node][i] > 0){

++ans;

}

}

return ans;

}