**ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ**

**ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**"ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ"**

ОТЧЕТ

по индивидуальному заданию

по дисциплине КДМ

Тема: «Поиск фундаментальных циклов графа»

Руководитель: Выполнил:

acc. ПИ ст. гр. ПИ-18б

Незамова Л.В. Мелещенко Н.В.

\_\_\_\_.\_\_\_\_.2019г. \_\_\_\_.\_\_\_\_.2019г.

Донецк – 2019

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc25623878)

[1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 4](#_Toc25623879)

[1.1 Описание алгоритма поиска фундаментальных циклов графа 4](#_Toc25623880)

[1.2 Исходные данные 7](#_Toc25623881)

[1.3 Алгоритм работы программы 12](#_Toc25623882)

[1.4 Изменения в алгоритме 15](#_Toc25623883)

[2 ПРАКТИЧЕСКАЯ ЧАСТЬ 16](#_Toc25623884)

[2.1 Ручной расчет контрольного примера 16](#_Toc25623885)

[2.2 Решение контрольного примера в программе 18](#_Toc25623886)

[2.3 Решение дополнительных контрольных примеров 20](#_Toc25623887)

[3 ОПИСАНИЕ ПРОГРАММЫ 22](#_Toc25623888)

[ВЫВОДЫ 23](#_Toc25623889)

[ПЕРЕЧЕНЬ ССЫЛОК 24](#_Toc25623890)

[ПРИЛОЖЕНИЕ А 25](#_Toc25623891)

[ПРИЛОЖЕНИЕ Б 27](#_Toc25623892)

[ПРИЛОЖЕНИЕ В 29](#_Toc25623893)

# ВВЕДЕНИЕ

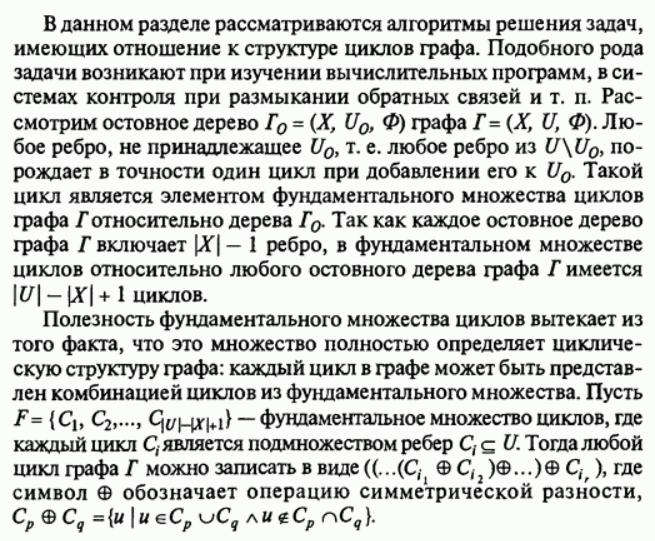
Дискретная математика — это раз­дел ма­те­ма­ти­ки, изу­чаю­щий свой­ст­ва дис­крет­ных струк­тур, ко­то­рые воз­ни­ка­ют как в са­мой ма­те­ма­ти­ке, так и в её при­ло­же­ни­ях. При этом дис­крет­ны­ми струк­ту­ра­ми на­зы­ва­ют­ся объ­ек­ты, для ко­то­рых важ­ней­шие ха­рак­те­ри­сти­ки при­ни­ма­ют ко­неч­ное или счёт­ное чи­сло зна­че­ний. К чис­лу та­ких струк­тур от­но­сят­ся, напр., ко­неч­ные груп­пы, ко­неч­ные гра­фы, не­ко­то­рые ма­те­ма­тические мо­де­ли пре­об­ра­зо­ва­те­лей ин­фор­ма­ции, ко­неч­ные ав­то­ма­ты. Это при­ме­ры струк­тур фи­нит­но­го (ко­неч­но­го) ха­рак­те­ра. Часть дискретной математики, изу­чаю­щая их, ино­гда на­зы­ва­ет­ся ко­неч­ной ма­те­ма­ти­кой. По­ми­мо фи­нит­ных струк­тур, дискретная математика изу­ча­ет так­же дис­крет­ные бес­ко­неч­ные струк­ту­ры (напр., бес­ко­неч­ные ал­геб­ра­ические сис­те­мы, бес­ко­неч­ные гра­фы, бес­ко­неч­ные ав­то­ма­ты) [1].

Дискретная математика представляет собой важный раздел математики, в котором можно выделить характерные предмет и методы исследования. Наряду с выделением Д. м. путем указания ее предмета можно также определить Д. м. посредством перечисления составляющих ее разделов. К ним относятся комбинаторный анализ, графов теория, теория кодирования, теория функциональных систем, теория управляющих систем, теория автоматов, алгоритмы и анализ их сложности. При более широком толковании к Д. м. могут быть отнесены как целые разделы математики, например, математическая логика, так и части таких разделов, как теория чисел, алгебра, вычислительная математика, теория вероятностей, в которых изучаемый объект носит дискретный характер [1].

# ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## 1.1 Описание алгоритма поиска фундаментальных циклов графа

Текст алгоритма из [2] представлен на рисунке 1.1



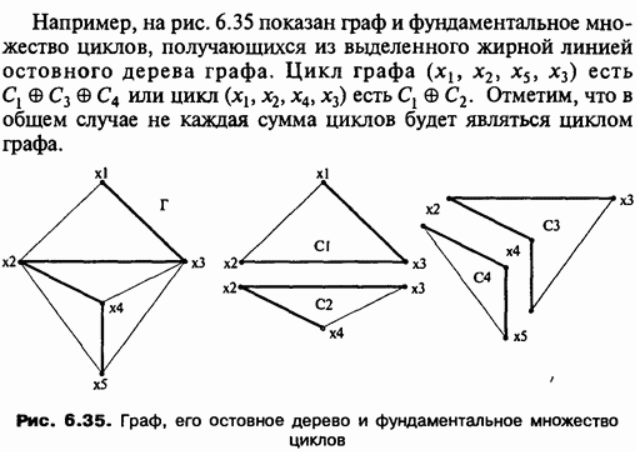
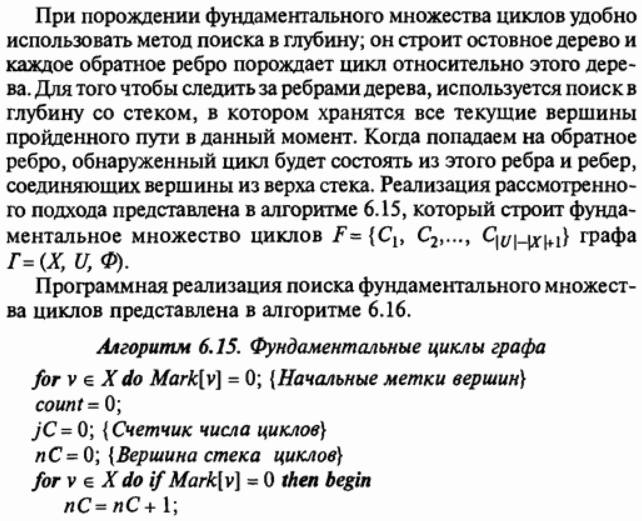


Рисунок 1.1 — Текст алгоритма из книги



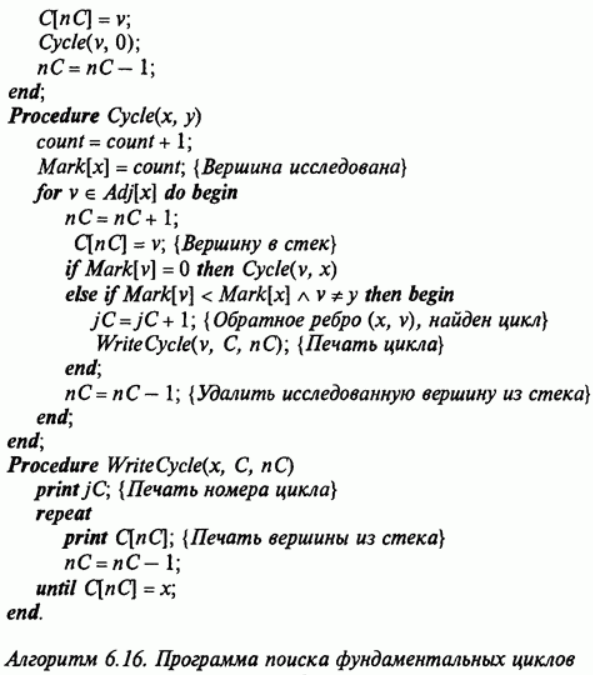


Рисунок 1.1 лист 2

Основными понятиями теории графов являются цикломатическое число графа и остов графа. Понятие цикломатического числа широко используется при проектировании средств вычислительной техники в таких задачах, как выделение схем с обратными связями на платах, обеспечение устойчивой работы цифровых автоматов за счет многофазной синхронизации и т.п.

Отличительной особенностью остова является следующая: добавление к остову графа хотя бы одного ребра, не принадлежащего остову, образует цикл, который называется фундаментальным (базисным) циклом графа. Число фундаментальных циклов графа всегда постоянно, не зависит от выбранного остова и равно числу хорд, не входящих в остов. Это число называется цикломатическим числом графа. Цикломатическое число графа можно определить по формуле:

V(G)=R-N+1, где R - число ребер графа; N - число вершин графа.

Фундаментальные циклы позволяют получить любой цикл, имеющиеся в графе, используя линейную операцию над фундаментальными циклами.

Цикломатическое число графа указывает то наименьшее число рёбер, которое нужно удалить из данного графа, чтобы получить дерево (для связного графа) или лес (для несвязного графа), т.е. добиться отсутствия у графа циклов.

Знание цикломатического числа оказывается полезным при анализе топологии электронных схем, а также для решения целого класса задач конструкторского проектирования РЭС.

## 1.2 Исходные данные

В качестве исходных данных программы были протестированы следующие виды графов (графы задаются матрицами смежности):

1. Клика на 4 вершины (см. рис. 1.2).

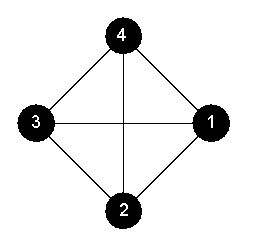


Рисунок 1.2 — Клика

1. Простой цикл на 5 вершин (см. рис. 1.3).

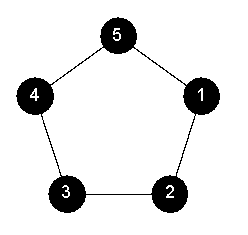


Рисунок 1.3 — Простой цикл

1. Простая цепь на 5 вершин (см. рис. 1.4).

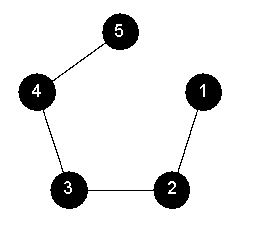


Рисунок 1.4 — Простая цепь

1. Пустой граф на 3 вершины (см. рис. 1.5).

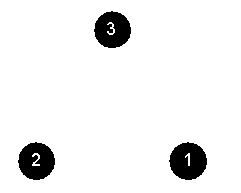


Рисунок 1.5 — Пустой граф

1. Тривиальный граф (см. рис. 1.6).



Рисунок 1.6 — Тривиальный граф

1. Однородный граф на 5 вершин со степенью вершин 2 (см. рис. 1.7).

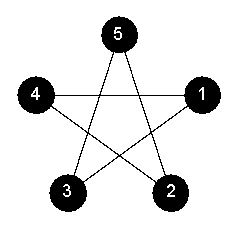


Рисунок 1.7 — Однородный граф

1. Двудольный граф 3,3 (см. рис. 1.8).

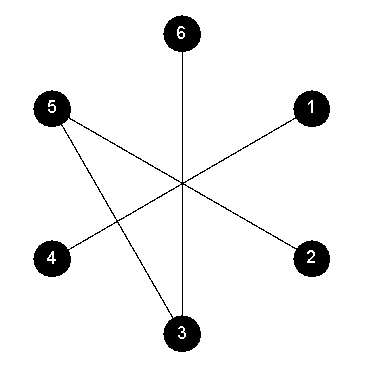


Рисунок 1.8 — Двудольный граф

1. Полный двудольный граф 3,3 (см. рис. 1.9).

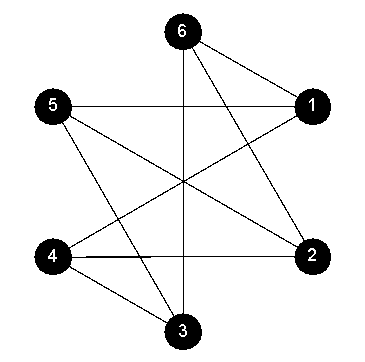


Рисунок 1.9 — Полный двудольный граф

1. Звезда 1,4 (см. рис. 1.10).

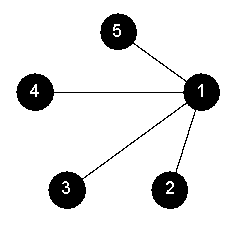


Рисунок 1.10 — Звезда

10) Несвязный граф (см. рис. 1.11).

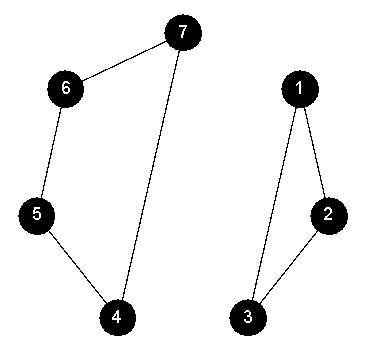


Рисунок 1.11 — Несвязный граф

11) Орграф сильный на 4 вершины (см. рис. 1.12).

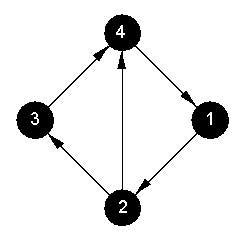


Рисунок 1.12 — Сильный орграф

12) Орграф односторонний на 4 вершины (см. рис. 1.13).

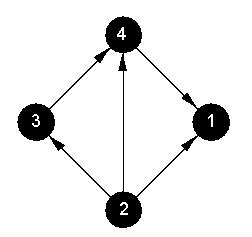


Рисунок 1.13 — Односторонний орграф

13) Орграф слабый на 4 вершины (см. рис. 1.14).

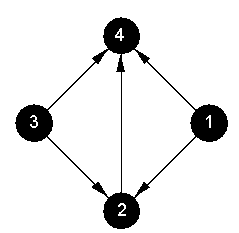


Рисунок 1.14 — Слабый орграф

14) Орпуть на 5 вершин (см. рис. 1.15).

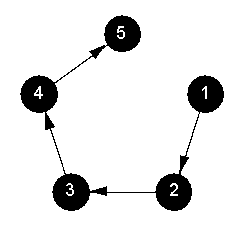


Рисунок 1.15 — Орпуть

15) Контур на 4 вершины (см. рис. 1.16).

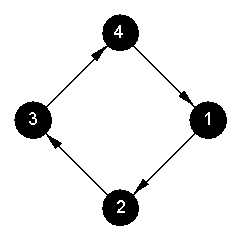


Рисунок 1.16 — Контур

## 1.3 Алгоритм работы программы

На вход алгоритма подается матрица смежности (см. рис. 1.17), в программе она может быть задана вручную либо считана из текстового файла (см. рис. 1.18).

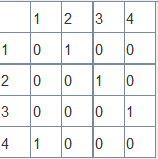


Рисунок 1.17 — Матрица смежности в программе

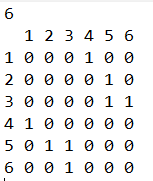


Рисунок 1.18 — Матрица смежности в файле

Если граф неориентированный, тогда запускается алгоритм поиска фундаментальных множеств, иначе пользователю выводится сообщение о некорректных входных данных и алгоритм не запускается, потому что понятие фундаментальных циклов графа относится только к неориентированным графам (см. рис. 1.19).

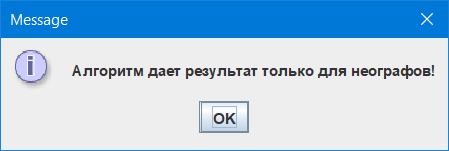


Рисунок 1.19 — Сообщение о некорректных входных данных

Принцип алгоритма поиска фундаментальных циклов графа: проходим по вершинам остовного дерева графа и если находим ребро, которое соединяет две вершины остовного дерева, то это и есть фундаментальный цикл, который состоит из вершин начиная с начальной вершины обратного ребра далее по остовному дереву и заканчивающийся концевой вершиной обратного ребра.

Для этого используется рекурсивный поиск в глубину, во время которого и ищутся обратные ребра путем маркирования вершин, то есть по мере прохода в глубину вершинам даются метки, и, если из одной вершины мы переходим в смежную ей вершину и ее метка меньше чем у первой, тогда мы находим обратное ребро (то есть мы перешли в вершину, которую уже прошли поиском в глубину и которая является вершиной остова).

Ход работы алгоритма

Всем вершинам присваиваются начальные метки 0.

Шаг 1. Выбираем вершину с наименьшей пометкой и меткой 0. Эта и последующие вершины будут находиться в 1-й компоненте связности (2-й, 3-й…) (с каждым повтором этого шага количество компонент связности увеличивается). Если таких вершин нет — конец алгоритма, иначе переходим к шагу 2.

Шаг 2. Даем ей метку 1 (2,3,4…) (с каждым повтором этого шага метки увеличиваются). Переходим к шагу 3.

Шаг 3. Добавляем эту вершину в стек выделения циклов графа. Переходим к шагу 4.

Шаг 4. Идем к смежным вершинам. На каждой вершине переходим к шагу 5. Если проверили все смежные вершины, тогда переходим к шагу 7.

Шаг 5. Если у смежной вершины метка 0, то для нее переходим к шагу 2 и выполняем его, потом возвращаемся, если метка меньше чем у той с которой пришли, то для нее переходим к шагу 6 и выполняем его, потом возвращаемся, если метка больше чем у той с которой пришли переходим к шагу 7 и возвращаемся.

Шаг 6. Выделяем фундаментальный граф из стека путем выписывания вершин начиная с головы стека до второго вхождения вершины из головы стека. Переходим к шагу 7.

Шаг 7. Удаляем из стека исследованную вершину. Если стек стал пустым, переходим к шагу 1.

В конце работы алгоритма программа прорисовывает на графе остовное дерево графа оранжевым цветом, а обратные ребра — красным, тем самым наглядно показывая пользователю результат (см. рис. 1.20), а также описывает эти циклы в текстовом виде (см. рис. 1.21).

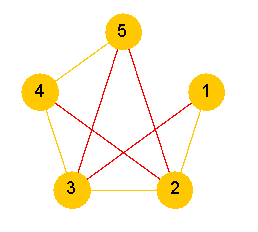


Рисунок 1.20 — Графический результат работы программы

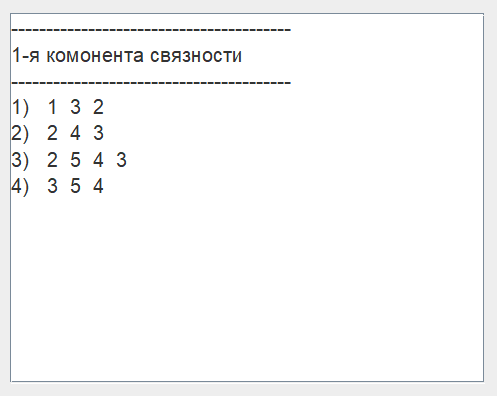


Рисунок 1.21 — Текстовой результат работы программы

## 1.4 Изменения в алгоритме

1) Так как понятие фундаментальных циклов относятся только к неориентированным графам, была добавлена проверка входных данных, которая проверяет является ли входной граф неориентированным, если граф оказывается неориентированным, тогда программа запускает алгоритм.

Проверка осуществляется следующим образом: если матрица смежности симметрична, то граф является неориентированным, в остальных случаях граф — ориентированный.

2) Добавлен вывод компонент связности, для этого добавлен счетчик компонент связности, который инкрементируется, когда срабатывает первый шаг алгоритма, т.е. когда берется вершина с наименьшей пометкой и меткой равной 0.

3) Для оптимизации проверка на возврат в вершину из которой пришли, и проверка на то что данная вершина уже исследована были вынесены перед добавление этих вершин в стек выделения циклов графа, так как трата ресурсов на добавление в стек вершины, которая сразу же будет из него удалена не имеет смысла, тем более это нелогично в сточки зрения алгоритма.

# 2 ПРАКТИЧЕСКАЯ ЧАСТЬ

## 2.1 Ручной расчет контрольного примера

Рассмотрим неориентированный граф на 5 вершин (см. рис. 2.1).

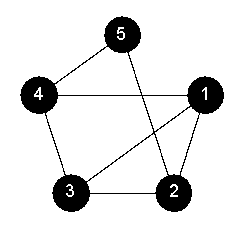


Рисунок 2.1 — Граф контрольного примера

Дадим всем вершинам начальные метки 0.

Берем вершину с минимальной пометкой и начальной меткой 0.

Вершина 1 имеет начальную пометку 0. Значит выбираем ее.

Вершину 1 добавляем в стек выделения циклов графа: {1}

Вершине 1 даем начальную метку 1

Далее переходи к смежным ей вершинам, если они есть

Вершинa 1 смежна вершине 2

Вершину 2 добавляем в стек выделения циклов графа: {1,2}

Вершинa 2 имеет начальную метку 0, поэтому далее выбираем ее

Вершине 2 даем начальную метку 2

Далее переходи к смежным ей вершинам, если они есть

Вершинa 2 смежна вершине 1

Эту вершину пропускаем так как мы из нее пришли

Вершинa 2 смежна вершине 3

Вершину 3 добавляем в стек выделения циклов графа: {1,2,3}

Вершинa 3 имеет начальную метку 0, поэтому далее выбираем ее

Вершине 3 даем начальную метку 3

Далее переходи к смежным ей вершинам, если они есть

Вершинa 3 смежна вершине 1

Вершину 1 добавляем в стек выделения циклов графа: {1,2,3,1}

Вершинa 1 имеет начальную метку >0, из этого следует, что найдено обратное ребро (3,1), а значит найден фундаментальный цикл:

1) 1 3 2

Удаляем вершину 1 из стека, так как она уже исследована: {1,2,3}

Вершинa 3 смежна вершине 2

Эту вершину пропускаем так как мы из нее пришли

Вершинa 3 смежна вершине 4

Вершину 4 добавляем в стек выделения циклов графа: {1,2,3,4}

Вершинa 4 имеет начальную метку 0, поэтому далее выбираем ее

Вершине 4 даем начальную метку 4

Далее переходи к смежным ей вершинам, если они есть

Вершинa 4 смежна вершине 1

Вершину 1 добавляем в стек выделения циклов графа: {1,2,3,4,1}

Вершинa 1 имеет начальную метку >0, из этого следует, что найдено обратное ребро (4,1), а значит найден фундаментальный цикл:

2) 1 4 3 2

Удаляем вершину 1 из стека, так как она уже исследована: {1,2,3,4}

Вершинa 4 смежна вершине 3

Эту вершину пропускаем так как мы из нее пришли

Вершинa 4 смежна вершине 5

Вершину 5 добавляем в стек выделения циклов графа: {1,2,3,4,5}

Вершинa 5 имеет начальную метку 0, поэтому далее выбираем ее

Вершине 5 даем начальную метку 5

Далее переходи к смежным ей вершинам, если они есть

Вершинa 5 смежна вершине 2

Вершину 2 добавляем в стек выделения циклов графа: {1,2,3,4,5,2}

Вершинa 2 имеет начальную метку >0, из этого следует, что найдено обратное ребро (5,2), а значит найден фундаментальный цикл:

3) 2 5 4 3

Удаляем вершину 2 из стека, так как она уже исследована: {1,2,3,4,5}

Вершинa 5 смежна вершине 4

Эту вершину пропускаем так как мы из нее пришли

Удаляем вершину 5 из стека, так как она уже исследована: {1,2,3,4}

Удаляем вершину 4 из стека, так как она уже исследована: {1,2,3}

Удаляем вершину 3 из стека, так как она уже исследована: {1,2}

Вершинa 2 смежна вершине 5

Эту вершину пропускаем так как она имеет большую начальную метку, а значит она уже исследована

Удаляем вершину 2 из стека, так как она уже исследована: {1}

Вершинa 1 смежна вершине 3

Эту вершину пропускаем так как она имеет большую начальную метку, а значит она уже исследована

Вершинa 1 смежна вершине 4

Эту вершину пропускаем так как она имеет большую начальную метку, а значит она уже исследована

Удаляем вершину 1 из стека, так как она уже исследована: {}

Проверяем всем ли вершинам были даны начальные метки, если нет, тогда граф имеет компонент связности >1 и алгоритм продолжит работу в другой компоненте.

Вершина 1 имеет начальную пометку >0

Вершина 2 имеет начальную пометку >0

Вершина 3 имеет начальную пометку >0

Вершина 4 имеет начальную пометку >0

Вершина 5 имеет начальную пометку >0

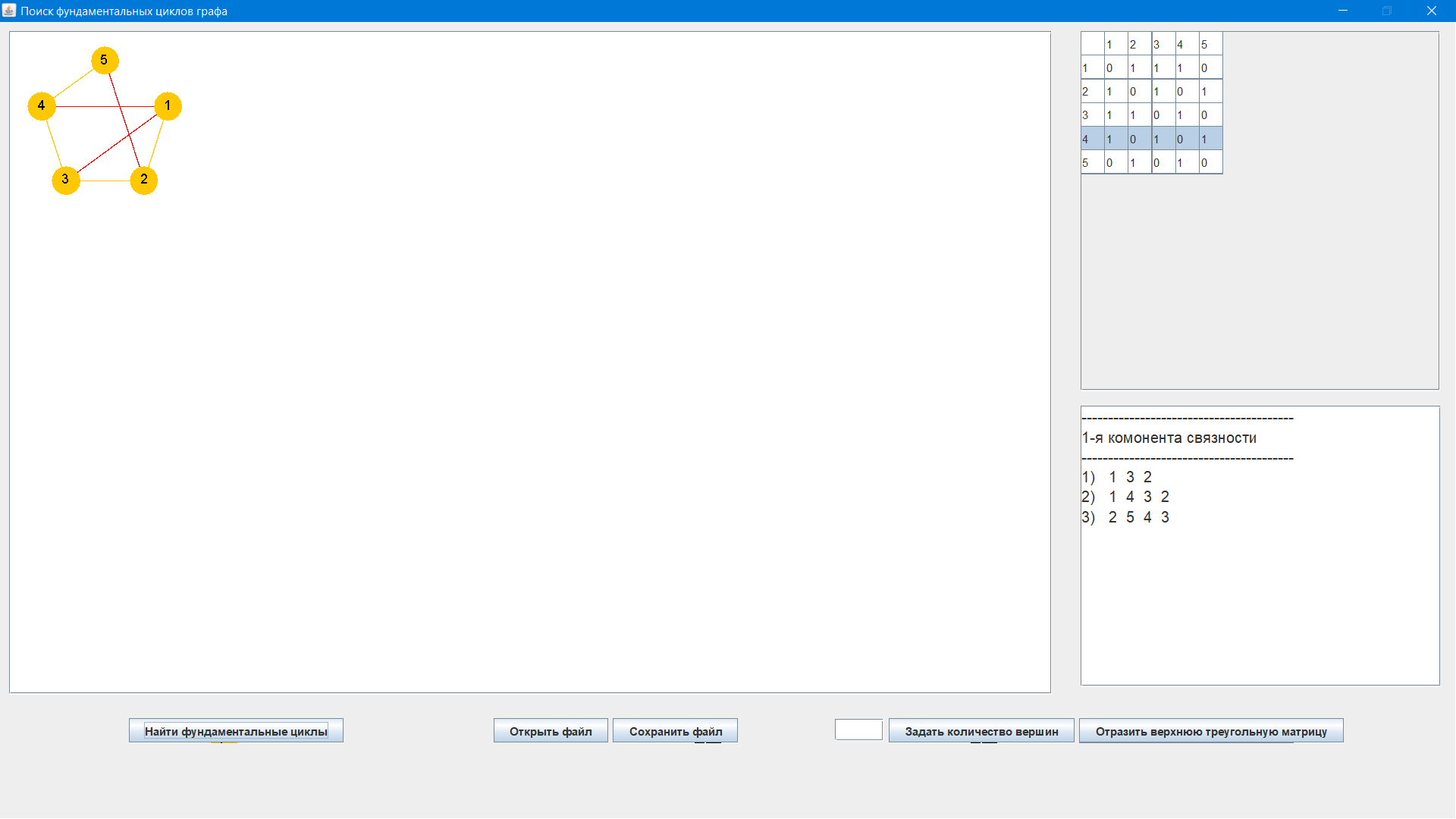
Всем вершинам даны начальные метки, значит конец алгоритма.

## 2.2 Решение контрольного примера в программе

На рисунке 2.2 представлена экранная форма программы во время выполнения контрольного примера.

По завершению алгоритма программа выдает все найденные фундаментальные циклы графа.

На рисунке 2.3 представлен листинг решения контрольного примера программой.



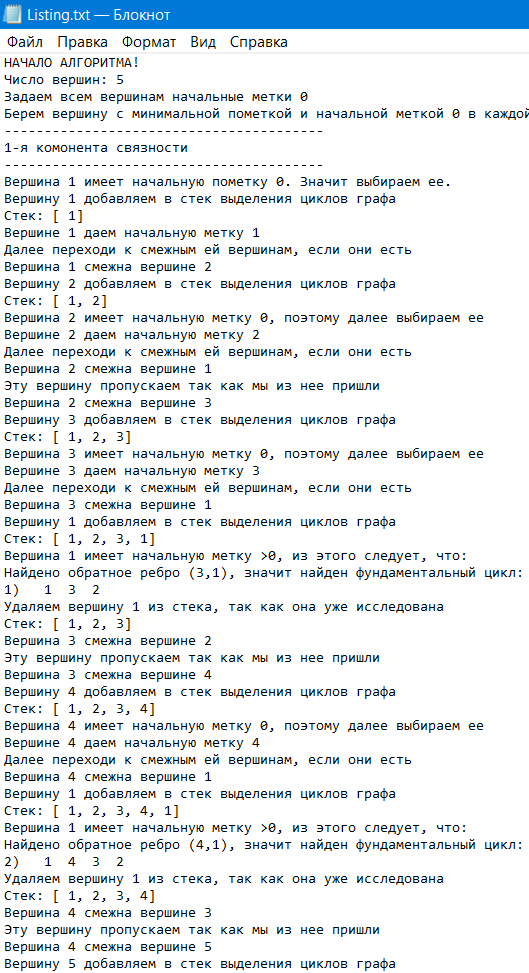
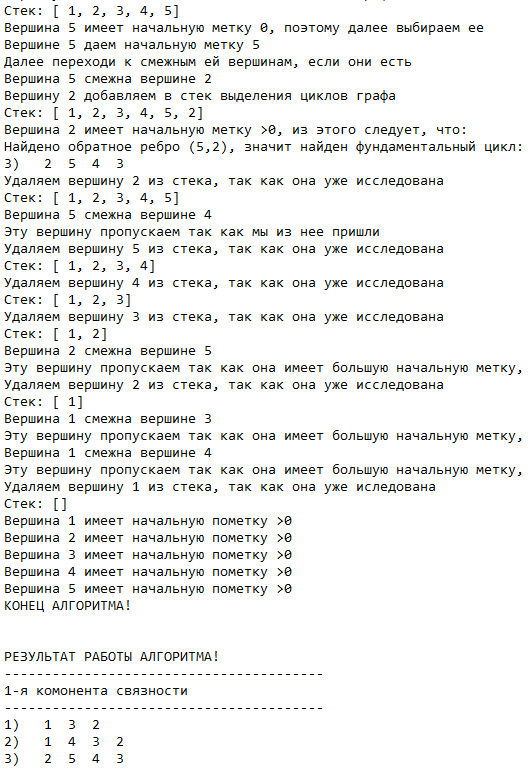
Рисунок 2.2 — Решение контрольного примера в программе

Рисунок 2.3 — Листинг решения контрольного примера

## 2.3 Решение дополнительных контрольных примеров

В качестве дополнительных контрольных примеров, иллюстрирующих поведение алгоритма на различных входных данных, были выбраны: пустой граф (см. рис. 1.5), орграф сильный (см. рис. 1.12), несвязный граф (см. рис. 1.11).

Результаты работы программы представлены на рисунках 2.4 – 2.6.

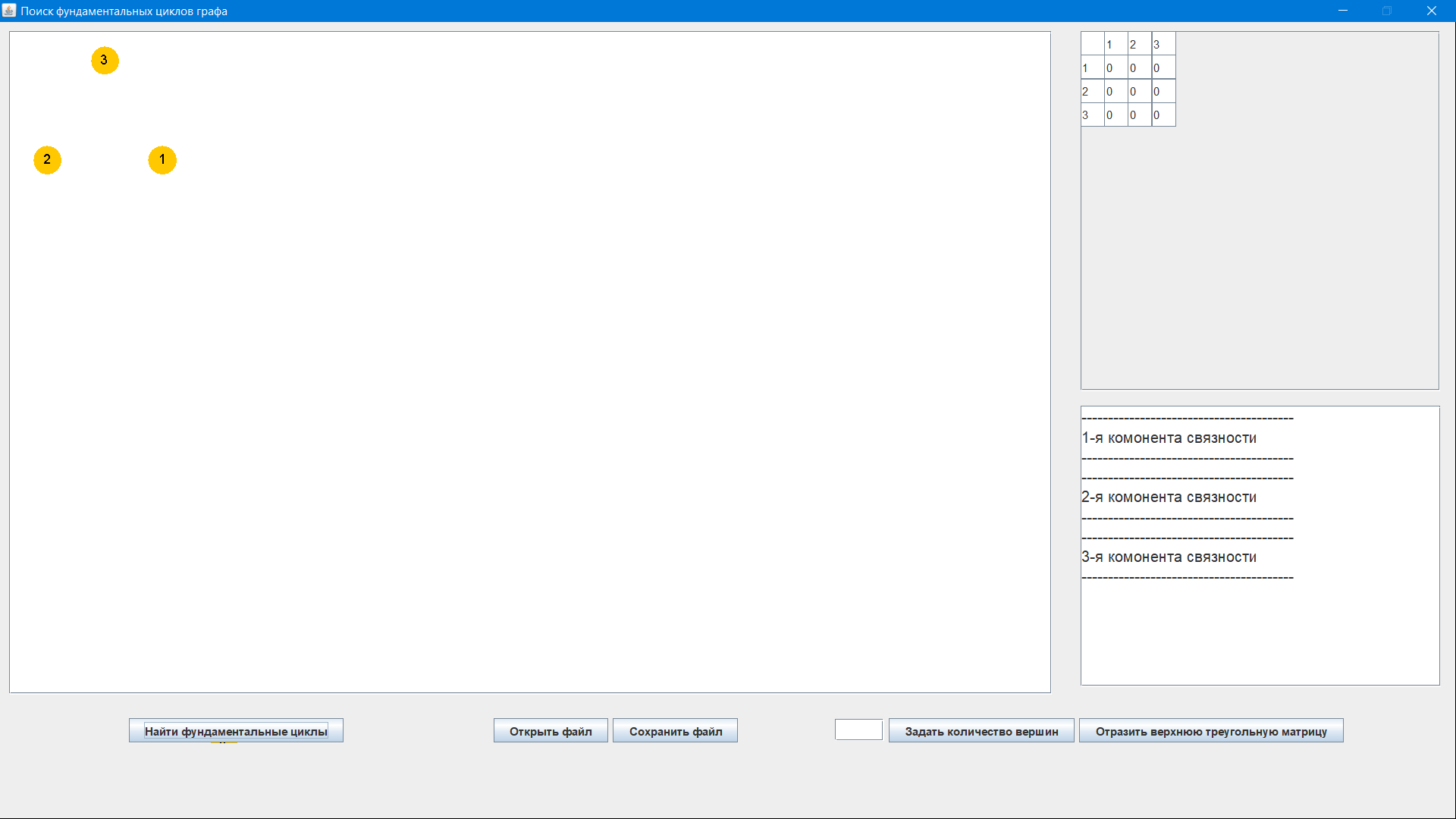


Рисунок 2.4 — Результат работы программы для пустого графа

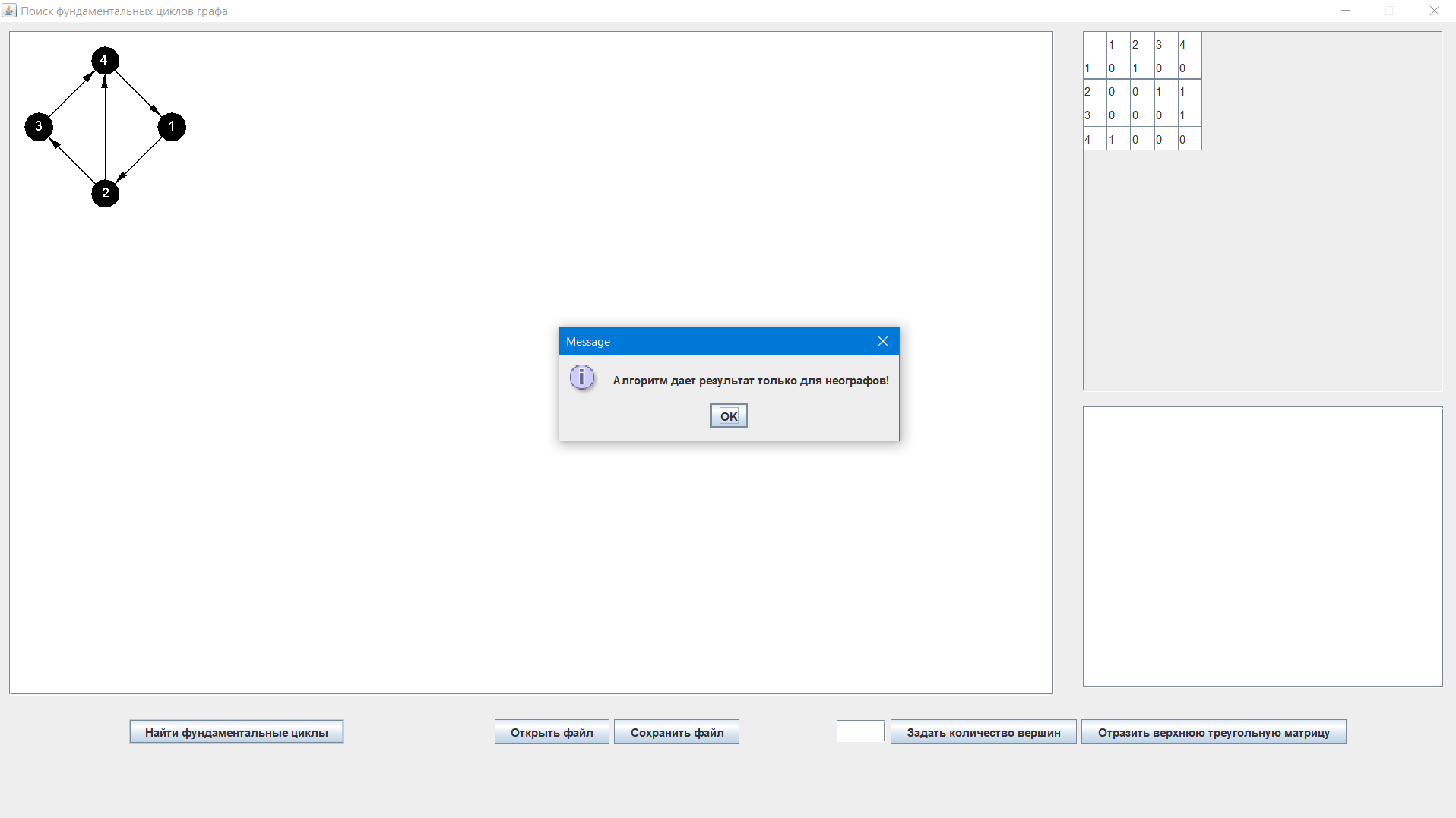


Рисунок 2.5 — Результат работы программы для сильного орграфа

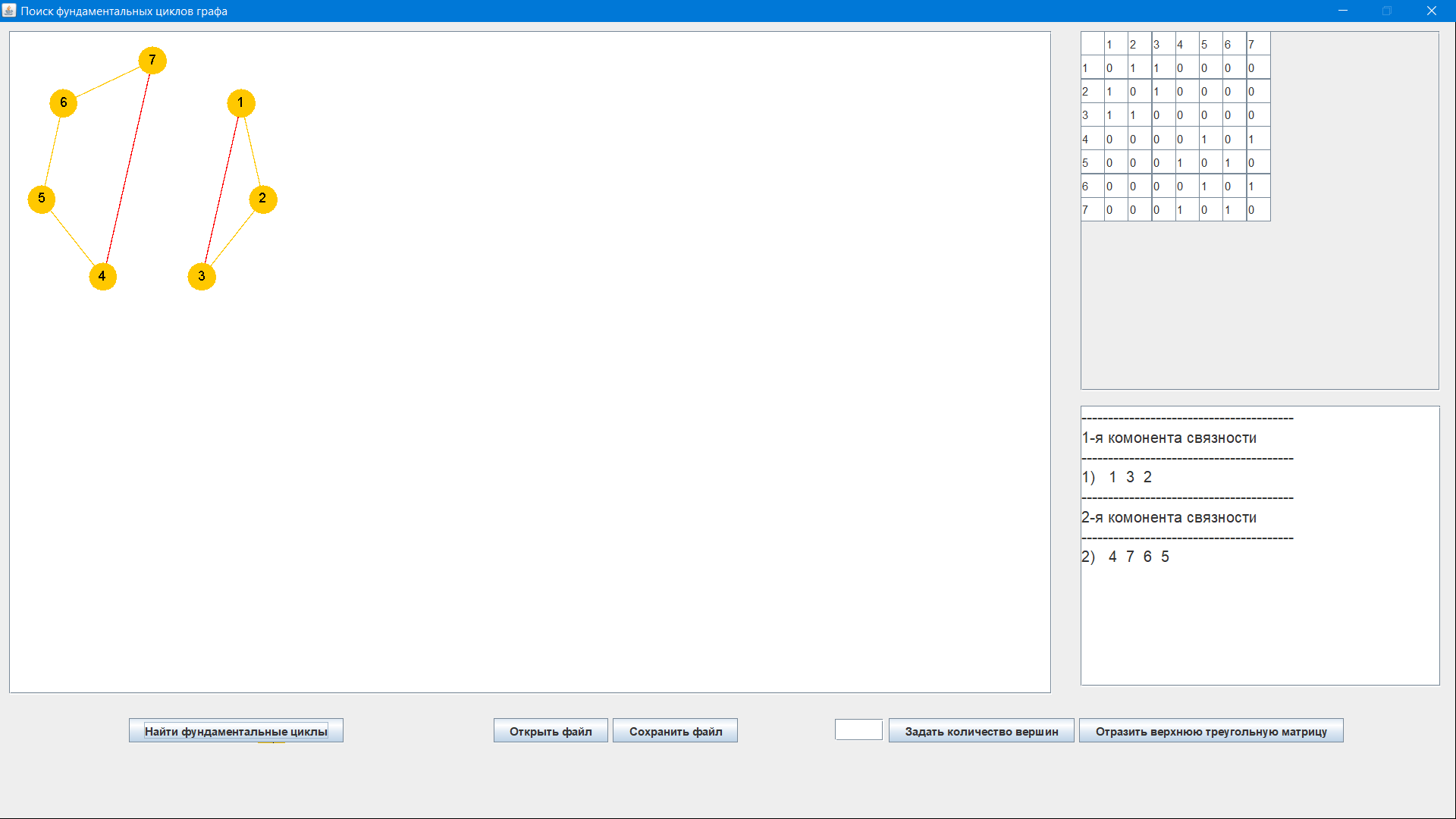


Рисунок 2.6 — Результат работы программы для несвязного графа

# 3 ОПИСАНИЕ ПРОГРАММЫ

Разработанная программа позволяет находить фундаментальные циклы введенного графа, визуально показывая, как исходный граф, так и его фундаментальные циклы. В программе предусмотрено редактирование введенного графа: добавление в него вершин, ребер и дуг. Все изменения отображаются динамически. Графы задаются матрицей смежности, данные в которую могут быть занесены как вручную, так и из файла. В программе предусмотрены все исключительные ситуации, такие как ошибки при чтении файла, запуск алгоритма без входных данных, некорректные входные данные.

Также предусмотрена функция сохранения в файл текущего графа (то есть его матрицу смежности), чтобы в любой момент снова открыть его в программе. Предусмотрена функция, которая делает матрицу смежности симметричной относительно диагонали с 0, то есть функция которая делает граф неориентированным если он был ориентированным.

Также результат работы алгоритма выводится на экран и дублируется в файле листинга решения задачи.

# ВЫВОДЫ

В ходе выполнения индивидуального задания был изучен алгоритм поиска фундаментальных циклов графа и разработана программа, реализующая данный алгоритм на языке программирования Java.

Данная программа может быть использована в высших учебных заведениях с целью демонстрации и лучшего понимания данного алгоритма студентами.

Преимущества разработанной программы: интуитивный интерфейс, динамически изменяемые графы, защита от исключительных ситуаций, возможность сохранения текущего графа в файл, подробный листинг алгоритма записывается в файл.

Недостатки разработанной программы: сложности при работе с большим количеством вершин, сложность в эксплуатации без базовых знаний теории графов, ограниченное максимальное количество вершин.

# ПЕРЕЧЕНЬ ССЫЛОК

1. Яблонский С.В., Введение в дискретную математику, М., 1979;
2. Иванов Б. Н. Дискретная математика. Алгоритмы и программы: Учеб. пособие. — М.: Лаборатория Базовых Знаний, 2001 — 288 с.

# ПРИЛОЖЕНИЕ А

ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЮ

Формат файлов исходных данных — текстовый (\*.txt).

Пример файла исходных данных представлен на рисунке А.1, а соответствующий файлу граф на рисунке А.2.

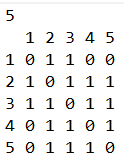


Рисунок А.1 — Пример файла исходных данных

В начале файла записывается количество вершин графа, а далее идет матрица смежности.

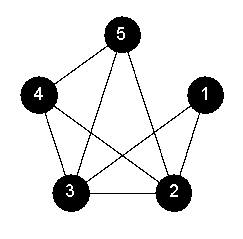


Рисунок А.2 — Соответствующий исходным данным граф

Во время работы программа записывает в текстовый файл (см. рис. А.3), который называется листингом решения задачи, все свои шаги, тем самым пользователь может разобраться и понять алгоритм, а в конце работы алгоритма дублирует на экране и в листинге результат работы программы (см. рис. А.4).

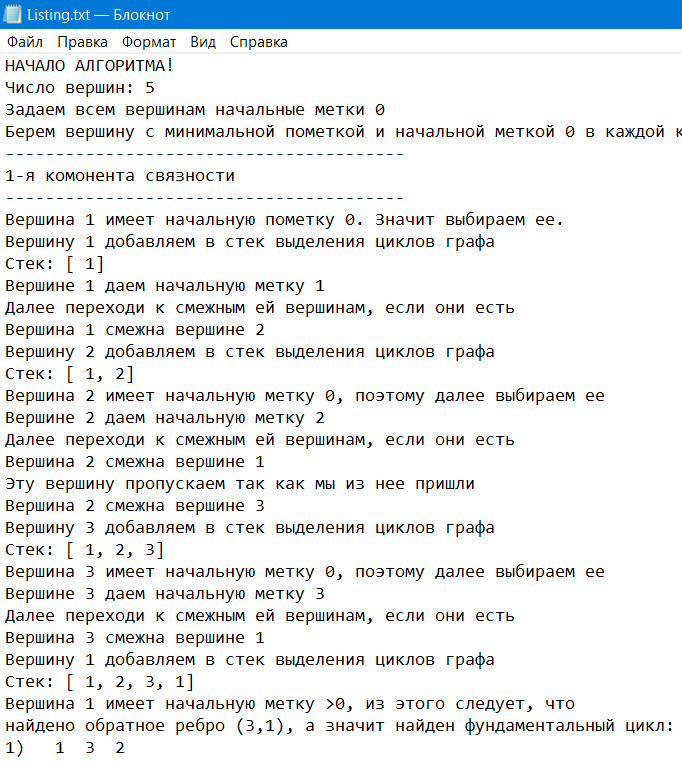


Рисунок А.3 — Пример листинга решения задачи

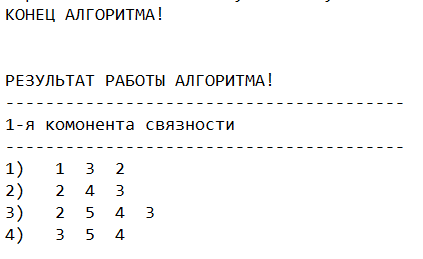


Рисунок А.4 — Дублирование результата работы программы в листинге решения задачи

В результате первая цифра, отделенная скобкой — это номер фундаментального цикла, а цифры после скобки — это вершины цикла.

Для запуска программы достаточно дважды щелкнуть п.к.м. по исполняемому Jar-файлу.

Для загрузки из файла следует нажат кнопку «Открыть файл», далее выбрать файл исходных данных и нажать открыть. Исходный граф сразу же будет нарисован, а таблица смежности заполнена из файла.

Для запуска алгоритма убедитесь, что исходный граф неориентированный, иначе вы получите уведомление от программы о некорректных исходных данных, и нажмите кнопку «Найти фундаментальные циклы». На экране появятся результаты алгоритма: номер цикла и его вершины, а также в какой компоненте связности он находится. Те циклы, которые находятся под надписью «i-я компонента связности», находятся в этой компоненте связности.

# ПРИЛОЖЕНИЕ Б

ЭКРАННЫЕ ФОРМЫ

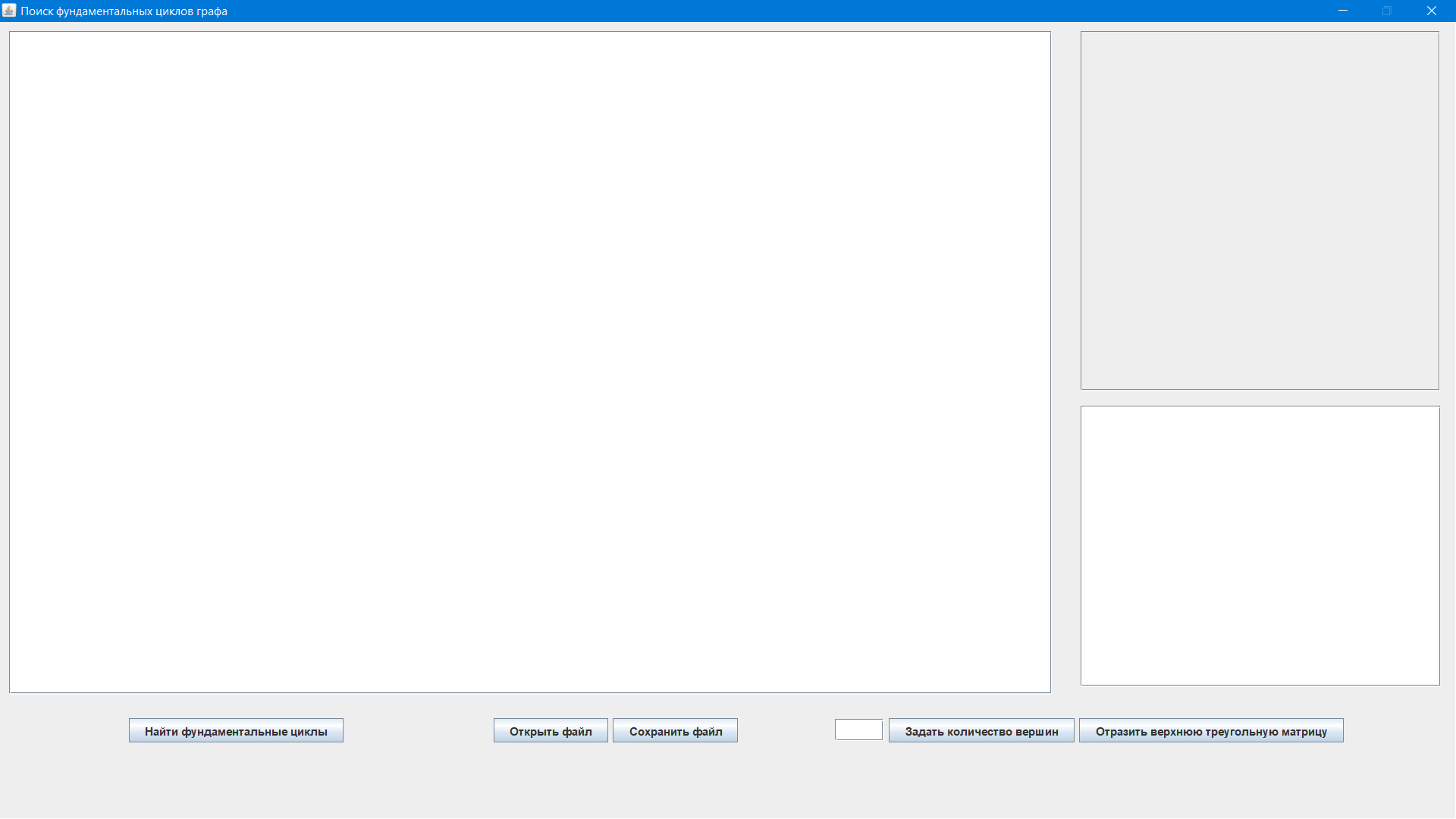


Рисунок Б.1 — Интерфейс программы после открытия

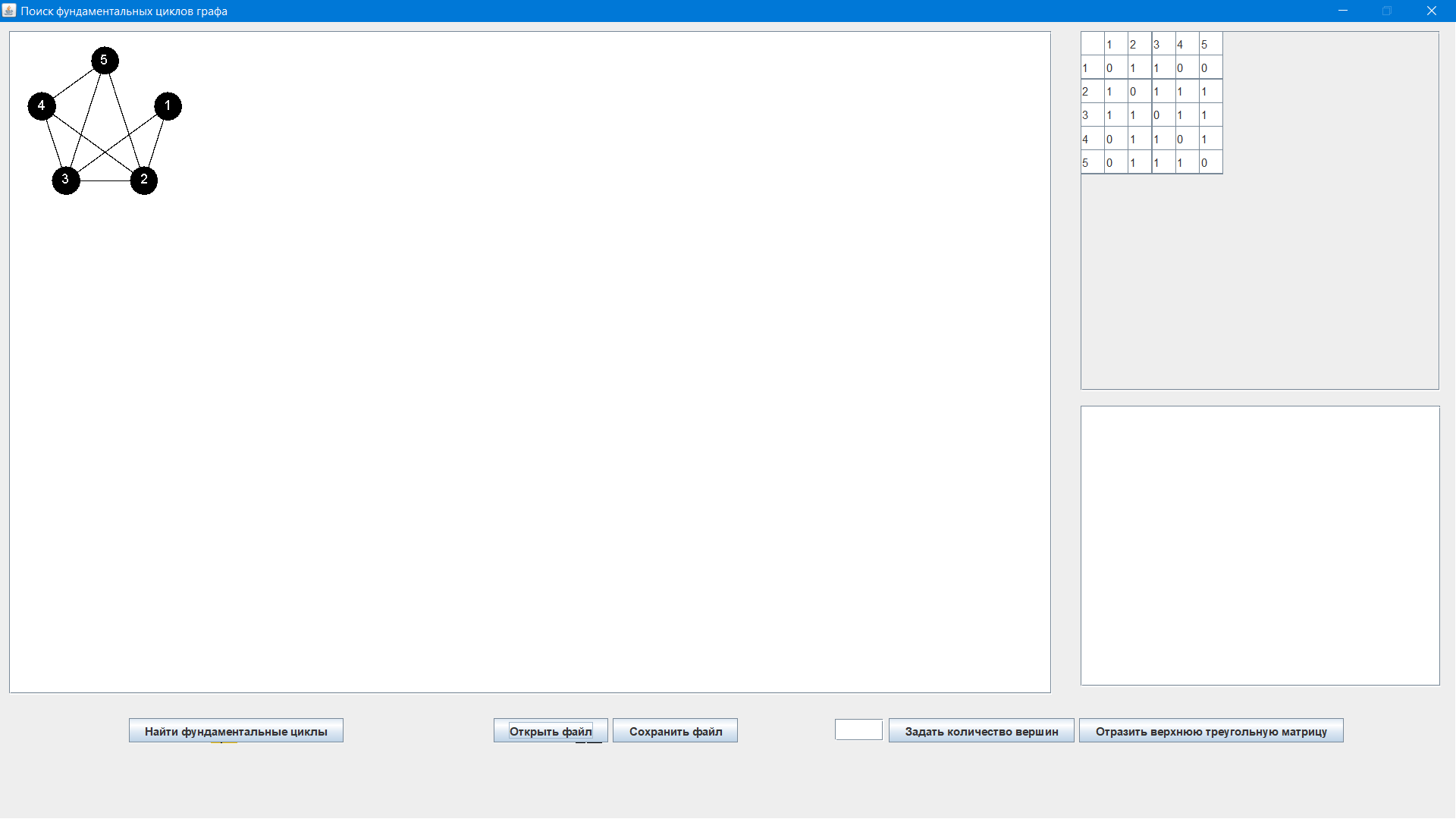


Рисунок Б.2 — Интерфейс программы с исходным графом

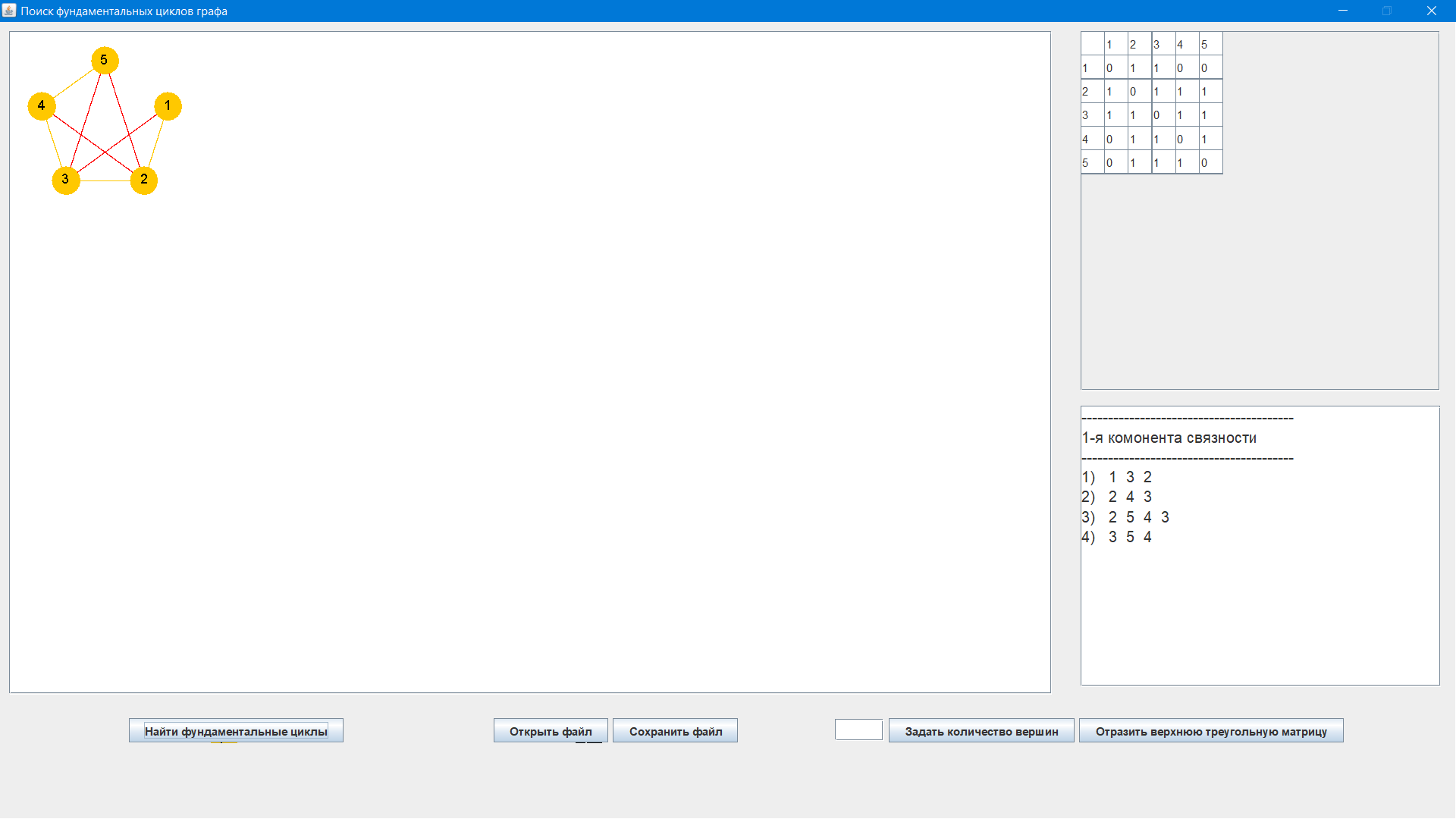


Рисунок Б.3 — Интерфейс программы после выполнения алгоритма

# ПРИЛОЖЕНИЕ В

ЛИСТИНГ ПРОГРАММЫ

**package** programm;

**import** javax.swing.\*;

**public** **class** Launcher {

**public** **static** **void** main(String[] args) {

SwingUtilities.*invokeLater*(() ->

{

BaseFrame frame = **new** BaseFrame();

frame.setTitle("Поиск фундаментальных циклов графа");

frame.setResizable(**false**);

frame.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

frame.setExtendedState(JFrame.***MAXIMIZED\_BOTH***);

frame.setVisible(**true**);

});

}

}

**package** programm;

**import** java.awt.\*;

**import** java.awt.geom.\*;

**import** java.awt.geom.Point2D.Double;

**import** java.io.\*;

**import** java.util.\*;

**import** javax.swing.\*;

**import** javax.swing.filechooser.FileNameExtensionFilter;

**import** javax.swing.table.AbstractTableModel;

**import** javax.swing.table.DefaultTableModel;

**public** **class** BaseFrame **extends** JFrame

{

**static** **boolean** *find* = **false**;//флаг отображение графа или его фунд. циклов

**static** **int** *n* = 0;//количество вершин

**static** **int** *count* = 0;//счетчик меток вершин

**static** **int** *jC* = 0;//счетчик числа циклов

**static** **int** *nC* = -1;//вершина стека циклов

**static** **int**[] *Mark*;//начальные метки вершин

**static** **int**[] *C*;//стек выделения циклов графа

**static** Point2D[] *coords*;//массив координат вершин

**static** FileReader *fr* = **null**;//файл для чтения

**static** FileWriter *fw* = **null**;//файл для записи

**static** Scanner *filescan* = **null**;//считывание с файла

**static** JFileChooser *fileChooser* = **new** JFileChooser("Files");//выбор файла

**static** JTable *AdjTable* = **null**;//матрица смежности

**static** DefaultTableModel *AdjTableModel* = **new** DefaultTableModel()//модель таблицы — матрицы смежности

{

**public** **boolean** isCellEditable(**int** row, **int** col)//запрещает редактирование ячеек с метками вершин и диагональ с 0

{

**if**(row == 0 || col == 0 || row == col)

**return** **false**;

**else**

**return** **true**;

}

**public** **void** setValueAt(Object v, **int** row, **int** col)//разрешает вносить в таблицу только 0 и 1

{

**if**(row == 0 || col == 0)

{

**super**.setValueAt(v, row, col);

}

**else**

{

**if**(v.toString().equals("1") || v.toString().equals("0"))

{

**super**.setValueAt(v, row, col);

}

**else**

JOptionPane.*showMessageDialog*(**null**, "В таблице только 0 и 1!");

}

}

};

**static** JScrollPane *AdjScroll* = **null**;//полосы прокрутки для матрицы смежности

**static** JTextArea *resultText* = **null**;//поле для вывода результата

**static** JPanel *res* = **null**;//панель для поле с результатами

**static**//статический блок инициализации

{

*fileChooser*.setFileFilter(**new** FileNameExtensionFilter("Text only (\*.txt)","txt"));//фильтр для выбора файлов

*AdjTable* = **new** JTable(*AdjTableModel*);

*AdjTable*.getTableHeader().setReorderingAllowed(**false**);

*AdjTable*.setRowHeight(25);

*AdjTable*.setAutoResizeMode(JTable.***AUTO\_RESIZE\_OFF***);

*AdjTable*.setTableHeader(**null**);

*AdjTable*.setIgnoreRepaint(**false**);

*AdjScroll* = **new** JScrollPane(*AdjTable*);

*AdjScroll*.setSize(**new** Dimension(380,380));

*AdjScroll*.setLocation(**new** Point(1140,10));

*resultText* = **new** JTextArea(14,27);

*resultText*.setFont(**new** Font("Sans Serif", Font.***PLAIN***, 16));

*resultText*.setEditable(**false**);

*res* = **new** JPanel();

*res*.setSize(**new** Dimension(380,380));

*res*.setLocation(**new** Point(1140,400));

*res*.add(**new** JScrollPane(*resultText*));

}

**public** BaseFrame()

{

setLocationByPlatform(**true**);

setLayout(**null**);

JPanel panel = **new** JPanel()//панель в которой рисуется граф

{

Point2D center;//центр круга по которому располагаются вершины

**double** radius;//радиус круга по которому располагаются вершины

**double** vertRadius = 15;//радиус вершин

**public** **void** paintComponent(Graphics g)//метод отрисовки объектов

{

**super**.paintComponent(g);

**if**(*n* == 0)//если кол-во вершин 0 ничего не рисуем

**return**;

Graphics2D g2 = (Graphics2D) g;

setSize(getPreferredSize());//меняем размер в зависимости от кол-ва вершин

center = **new** Point2D.Double(getWidth() / 2, getHeight() / 2);

radius = center.getX() - 30D;

Coords();//определяем координаты вершин

**if**(*AdjTable*.getCellEditor() != **null**)//если матрица редактируется отрисовываем новый граф

{

*find* = **false**;//режим отображение графа без фунд. циклов

*resultText*.setText("");

}

**try**

{

**if**(!*find*)//рисуем граф смотря от режима: рисуем с фунд. циклами или без

{

drawGraph(g2);

}

**else**

{

drawCycle(g2);

drawMark(g2,Color.***ORANGE***, Color.***BLACK***);

}

}**catch**(NullPointerException ne)

{

**return**;

}

**this**.repaint();

}

**public** **void** Coords()//вычисление координат вершин графа для отображения

{

*coords* = **new** Point2D[*n*];

**double** angl = 360D / *n*;

**double** a = angl;

**for**(**int** i = 0; i < *coords*.length; i++)

{

**double** x = (center.getX() - center.getX());

**double** y = (center.getY() - radius - center.getY());

*coords*[i] = **new** Point2D.Double( ( (x \* Math.*cos*(Math.*toRadians*(a))) - (y \* Math.*sin*(Math.*toRadians*(a)))) + center.getX(),

( (x \* Math.*sin*(Math.*toRadians*(a))) + (y \* Math.*cos*(Math.*toRadians*(a)))) + center.getY() );

a += angl;

}

}

**public** **void** drawGraph(Graphics2D g2)//отрисовка графа в зависимости от его ориентированности или неориентированности

{

**boolean** isNeo = *isNeoGraph*();

**for**(**int** v = 1; v < *AdjTable*.getRowCount(); v++)

{

**for**(**int** adj = 1; adj < *AdjTable*.getColumnCount(); adj++)

{

**if**(*AdjTable*.getValueAt(v, adj).toString().equals("1"))

{

**if**(isNeo)

{

drawNeoEdge(g2, v , adj, Color.***BLACK***);

}

**else**

drawOrEdge(g2, v, adj, Color.***BLACK***);

}

}

}

drawMark(g2, Color.***BLACK***, Color.***WHITE***);

}

**public** **void** drawOrEdge(Graphics2D g2, **int** v, **int** adj, Color color)//отрисовка ориентированного ребра

{

g2.setColor(color);

**double** k = vertRadius/Math.*sqrt*(Math.*pow*(*coords*[v - 1].getX() - *coords*[adj - 1].getX(),2)

+ Math.*pow*(*coords*[v - 1].getY() - *coords*[adj - 1].getY(),2) );

Point2D c = **new** Point2D.Double((*coords*[adj - 1]).getX() + k\*(*coords*[v - 1].getX() - *coords*[adj - 1].getX()),

(*coords*[adj - 1]).getY() + k\*(*coords*[v - 1].getY() - *coords*[adj - 1].getY()));

**double** a = 165D;

**double** x = (*coords*[adj - 1].getX() - c.getX());

**double** y = (*coords*[adj - 1].getY() - c.getY());

Point2D first = **new** Point2D.Double( ( (x \* Math.*cos*(Math.*toRadians*(a))) - (y \* Math.*sin*(Math.*toRadians*(a)))) + c.getX(),

( (x \* Math.*sin*(Math.*toRadians*(a))) + (y \* Math.*cos*(Math.*toRadians*(a)))) + c.getY() );

Point2D second = **new** Point2D.Double( ( (x \* Math.*cos*(Math.*toRadians*(-a))) - (y \* Math.*sin*(Math.*toRadians*(-a)))) + c.getX(),

( (x \* Math.*sin*(Math.*toRadians*(-a))) + (y \* Math.*cos*(Math.*toRadians*(-a)))) + c.getY() );

**int**[] xPoints = {(**int**) c.getX(), (**int**) first.getX(), (**int**) second.getX()};

**int**[] yPoints = {(**int**) c.getY(), (**int**) first.getY(), (**int**) second.getY()};

g2.fillPolygon(xPoints, yPoints, 3);

drawNeoEdge(g2, v, adj, color);

}

**public** **void** drawNeoEdge(Graphics2D g2, **int** v, **int** adj, Color color)//отрисовка неориентированного ребра

{

g2.setColor(color);

g2.draw(**new** Line2D.Double(*coords*[v - 1],*coords*[adj - 1]));

}

**public** **void** drawCycle(Graphics2D g2)//отрисовка фунд. циклов

{

**for**(**int** k = 0; k < *Mark*.length; k++)

*Mark*[k] = 0;

*count* = 0;

*nC* = -1;

**for**(**int** v = 0; v < *n*; v++)

{

**if**(*Mark*[v] == 0)

{

*nC*++;

*C*[*nC*] = v;

Cycle(v,0,g2);

*nC*--;

}

}

}

**public** **void** drawMark(Graphics2D g2, Color vert, Color mark)//отрисовка меток вершин

{

**for**(**int** i = 0; i < *coords*.length; i++)

{

g2.setColor(vert);

g2.fill(**new** Ellipse2D.Double(*coords*[i].getX() - vertRadius,

*coords*[i].getY() - vertRadius,

vertRadius \* 2,

vertRadius \* 2));

g2.setColor(mark);

g2.setFont(**new** Font("Sans Serif", Font.***PLAIN***, 14));

g2.drawString((i+1) + "", (**int**)(*coords*[i].getX() - 4), (**int**)(*coords*[i].getY() + 4));

}

}

**public** **void** Cycle(**int** x, **int** y,Graphics2D g2)//рекурсивная функция отрисовки циклов

{

*count*++;

*Mark*[x] = *count*; //вершина исследована

**for**(**int** i = 1; i < *AdjTable*.getColumnCount(); i++)

{

**if**(*AdjTable*.getValueAt(x + 1, i).toString().equals("1"))

{

**int** v = i - 1;

*nC*++;

*C*[*nC*] = v; //вершину в стек

**if**(*Mark*[v] == 0)

{

g2.setColor(Color.***ORANGE***);

g2.draw(**new** Line2D.Double(*coords*[v],*coords*[*C*[*nC*-1]]));

Cycle(v,x,g2);

}

**else**

{

**if**((*Mark*[v] < *Mark*[x]) & v!=y)// v!=y исключаем возвращение назад в вершину которую уже прошли

{

g2.setColor(Color.***RED***);

g2.draw(**new** Line2D.Double(*coords*[v],*coords*[x]));

}

}

*nC*--;//удаляем исследованную вершину из стека

}

}////for

}

**public** Dimension getPreferredSize()//предпочтительный размер панели в зависимости от вершин

{

**if**(*n* < 5)

**return** **new** Dimension(200,200);

**else**

**return** **new** Dimension((*n*/2) \* 100, (*n*/2) \* 100);

}

};

panel.setBackground(Color.***WHITE***);

JScrollPane scroll = **new** JScrollPane(panel);//полосы прокрутки для панели графа

scroll.setLocation(**new** Point(10,10));

scroll.setSize(**new** Dimension(1100,700));

JPanel buttons = **new** JPanel();

buttons.setSize(**new** Dimension(300,100));

buttons.setLocation(**new** Point(100,730));

JButton findButton = **new** JButton("Найти фундаментальные циклы");

findButton.addActionListener(event ->

{

**if**(*AdjTable*.getCellEditor() != **null**)//убираем фокус от матрицы смежности

*AdjTable*.getCellEditor().stopCellEditing();

**if**(*n* == 0)//если граф не занан выдаем сообщение

{

JOptionPane.*showMessageDialog*(**null**, "Задайте граф!");

**return**;

}

**if**(!*isNeoGraph*())//если граф ориентированный выдаем сообщение о неправильных входных данных

{

JOptionPane.*showMessageDialog*(**null**, "Алгоритм дает результат только для неографов!");

**return**;

}

**try** {

*fw* = **new** FileWriter("Listing.txt");

} **catch** (Exception e) {

JOptionPane.*showMessageDialog*(**null**, "Нельзя открыть файл для сохранения!");

**return**;

}

*DepthCycle*();//главный алгоритм поиска фунд. циклов

*find* = **true**;//режим отображения фунд. циклов

panel.repaint();

**try** {

*fw*.close();

} **catch** (IOException e) {

JOptionPane.*showMessageDialog*(**null**, "Нельзя закрыть файл для сохранения!");

**return**;

}

}

);

JPanel matrixButtons = **new** JPanel();

matrixButtons.setSize(**new** Dimension(600,380));

matrixButtons.setLocation(**new** Point(850,730));

JPanel file = **new** JPanel();

file.setSize(**new** Dimension(400,400));

file.setLocation(**new** Point(450,730));

JButton openFile = **new** JButton("Открыть файл");//кнопка открытия файла

openFile.addActionListener(event ->

{

**if**(*AdjTable*.getCellEditor() != **null**)

*AdjTable*.getCellEditor().stopCellEditing();

*fileChooser*.setDialogTitle("Выбор файла");

*fileChooser*.setFileSelectionMode(JFileChooser.***FILES\_ONLY***);

**int** result = *fileChooser*.showOpenDialog(BaseFrame.**this**);

**if**(result == JFileChooser.***APPROVE\_OPTION***)

{

JOptionPane.*showMessageDialog*(BaseFrame.**this**, "Выбран файл: " + *fileChooser*.getSelectedFile());

**if**(*FileReading*())//если файл считался удачно, то отрисовываем граф

{

*quadrMatrix*();//ячейки в виде квадратов

*find* = **false**;

}

**else**//если нет — не отрисовываем ничего

{

*matrixOrder*(*n* = 0);

}

panel.repaint();

*resultText*.setText("");

}

}

);

JButton saveFile = **new** JButton("Сохранить файл");//кнопка сохранения текущей матрицы смежности

saveFile.addActionListener(event ->

{

**if**(*AdjTable*.getCellEditor() != **null**)

*AdjTable*.getCellEditor().stopCellEditing();

**if**(*n* == 0)

{

JOptionPane.*showMessageDialog*(**null**, "Задайте граф!");

**return**;

}

*fileChooser*.setDialogTitle("Сохранение файла");

*fileChooser*.setFileSelectionMode(JFileChooser.***FILES\_ONLY***);

**int** result = *fileChooser*.showSaveDialog(BaseFrame.**this**);

**if** (result == JFileChooser.***APPROVE\_OPTION*** )

{

*FileWriting*();//запись в файл

JOptionPane.*showMessageDialog*(BaseFrame.**this**,

"Файл " + *fileChooser*.getSelectedFile() +

".txt сохранен");

}

});

JTextField orderField = **new** JTextField(4);

orderField.setFont(**new** Font("Sans Serif", Font.***PLAIN***, 14));

JButton orderButton = **new** JButton("Задать количество вершин");//определение кол-ва вершин в новой матрице смежности

orderButton.addActionListener(event ->

{

**if**(*AdjTable*.getCellEditor() != **null**)

*AdjTable*.getCellEditor().stopCellEditing();

**int** tmp;

**if**(orderField.getText().equals(""))

{

JOptionPane.*showMessageDialog*(**null**, "Задайте количество вершин!");

**return**;

}

**try**

{

tmp = Integer.*parseInt*(orderField.getText());

}**catch**(Exception e)//перехват некорректного ввода

{

JOptionPane.*showMessageDialog*(**null**, "Некорректный ввод количества вершин! Повторите ввод.");

orderField.setText("");

**return**;

}

**if**(tmp < 1 || tmp > 1000)//ограничения

{

JOptionPane.*showMessageDialog*(**null**, "Количество вершин в пределах от 1 до 1000!");

orderField.setText("");

**return**;

}

*n* = tmp;

*matrixOrder*(*n*+1);

*Mark* = **new** **int**[*n*];//начальные метки вершин

*C* = **new** **int**[*n*+1];//стек выделения циклов графа

**for**(**int** i = 1; i < *AdjTable*.getColumnCount(); i++)//заполнение меток вершин матрицы смежности

{

*AdjTable*.setValueAt(i + "", 0, i);

*AdjTable*.setValueAt(i + "", i, 0);

}

**for**(**int** i = 1; i < *AdjTable*.getColumnCount(); i++)//запонение матрицы нулями

{

**for**(**int** j = 1; j < *AdjTable*.getColumnCount(); j++)

{

*AdjTable*.setValueAt("0", i, j);

}

}

*quadrMatrix*();//квадратные ячейки матрицы

*find* = **false**;//режим отображение фунд. циклов выключен

panel.repaint();

*resultText*.setText("");

orderField.setText("");

}

);

JButton symmetric = **new** JButton("Отразить верхнюю треугольную матрицу");//отразить верхнюю тругольную матрицу в матрице смежности

//чтобы сделать граф неориентированным

symmetric.addActionListener(event ->

{

**if**(*n* == 0)

{

JOptionPane.*showMessageDialog*(**null**, "Задайте граф!");

**return**;

}

*find* = **false**;

**if**(*AdjTable*.getCellEditor() != **null**)

*AdjTable*.getCellEditor().stopCellEditing();

**for**(**int** v = 1; v < *AdjTable*.getRowCount(); v++)

{

**for**(**int** adj = 1 + v; adj < *AdjTable*.getColumnCount(); adj++)

{

*AdjTable*.setValueAt(*AdjTable*.getValueAt(v, adj), adj, v);

}

}

*resultText*.setText("");

panel.repaint();

}

);

file.add(openFile);

file.add(saveFile);

matrixButtons.add(orderField);

matrixButtons.add(orderButton);

matrixButtons.add(symmetric);

buttons.add(findButton);

add(scroll);

add(buttons);

add(*AdjScroll*);

add(matrixButtons);

add(*res*);

add(file);

}

**static** **boolean** FileReading()//чтение файла

{

**try** {

*fr* = **new** FileReader(*fileChooser*.getSelectedFile());

} **catch** (Exception e) {

JOptionPane.*showMessageDialog*(**null**, "Нет доступа к файлу!");

**return** **false**;

}

*filescan* = **new** Scanner(*fr*);

**try** {

*n* = *filescan*.nextInt();//кол-во вершин

*filescan*.nextLine();

} **catch** (Exception e) {

JOptionPane.*showMessageDialog*(**null**, "Файл пуст или испорчен!");

*filescan*.close();

**try** {

*fr*.close();

} **catch** (IOException e1) {

JOptionPane.*showMessageDialog*(**null**, "Файл нельзя закрыть!");

**return** **false**;

}

**return** **false**;

}

**if**(*n* > 0 & *n* < 1001)//ограничения

{

*Mark* = **new** **int**[*n*];//начальные метки вершин

*C* = **new** **int**[*n*+1];//стек выделения циклов графа

}**else**

{

JOptionPane.*showMessageDialog*(**null**, "Количество вершин от 1 до 1000!");

*filescan*.close();

**try** {

*fr*.close();

} **catch** (IOException e1) {

JOptionPane.*showMessageDialog*(**null**, "Файл нельзя закрыть!");

**return** **false**;

}

**return** **false**;

}

*matrixOrder*(*n*+1);//задаем порядок матрицы

**for**(**int** i = 0; i < *n*; i++)//проверка меток в столбцах матрицы смежности

{

**try**

{

**if**(*filescan*.nextInt() != i+1)

{

JOptionPane.*showMessageDialog*(**null**, "Файл испорчен: Метки вершин в таблице смежности! Столбец: " + (i+1));

*filescan*.close();

**try** {

*fr*.close();

} **catch** (IOException e1) {

JOptionPane.*showMessageDialog*(**null**, "Файл нельзя закрыть!");

**return** **false**;

}

**return** **false**;

}

**else**

*AdjTable*.setValueAt((i+1) + "", 0, i+1);

}**catch**(Exception e)

{

JOptionPane.*showMessageDialog*(**null**, "Файл испорчен: Нет меток вершин");

*filescan*.close();

**try** {

*fr*.close();

} **catch** (IOException e1) {

JOptionPane.*showMessageDialog*(**null**, "Файл нельзя закрыть!");

**return** **false**;

}

**return** **false**;

}

}

*filescan*.nextLine();

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

{

**try** {

**int** mark = *filescan*.nextInt(); //метка вершины

**if**(mark != i+1)//проверка меток в строках матрицы смежности

{

JOptionPane.*showMessageDialog*(**null**, "Файл испорчен: Метки вершин в таблице смежности! Строка: " + (i+1));

*filescan*.close();

**try** {

*fr*.close();

} **catch** (IOException e1) {

JOptionPane.*showMessageDialog*(**null**, "Файл нельзя закрыть!");

**return** **false**;

}

**return** **false**;

}

**else**

*AdjTable*.setValueAt((i+1) + "", i+1, 0);

**for**(**int** j = 0; j < *n*;j++)//проверка матрицы смежности

{

**try**

{

**int** tmp = *filescan*.nextInt();

**if**(j == i & tmp != 0)

{

JOptionPane.*showMessageDialog*(**null**, "Файл испорчен: В диагонале только 0 ! Строка: " + (i+1));

*filescan*.close();

**try** {

*fr*.close();

} **catch** (IOException e1) {

JOptionPane.*showMessageDialog*(**null**, "Файл нельзя закрыть!");

**return** **false**;

}

**return** **false**;

}

**if**(tmp == 1)

{

*AdjTable*.setValueAt(1 + "", i + 1, j + 1);

}**else**

{

**if**(tmp != 0)

{

JOptionPane.*showMessageDialog*(**null**, "Файл испорчен: В таблице смежности числа только 0 и 1! Строка: " + (i+1) + " Столбец: " + (j+1));

*filescan*.close();

**try** {

*fr*.close();

} **catch** (IOException e1) {

JOptionPane.*showMessageDialog*(**null**, "Файл нельзя закрыть!");

**return** **false**;

}

**return** **false**;

}

**else**

*AdjTable*.setValueAt(0 + "", i + 1, j + 1);

}

}**catch**(Exception e)

{

JOptionPane.*showMessageDialog*(**null**, "Файл испорчен!");

*filescan*.close();

**try** {

*fr*.close();

} **catch** (IOException e1) {

JOptionPane.*showMessageDialog*(**null**, "Файл нельзя закрыть!");

**return** **false**;

}

**return** **false**;

}

}

} **catch** (Exception e) {

JOptionPane.*showMessageDialog*(**null**, "Файл испорчен!");

*filescan*.close();

**try** {

*fr*.close();

} **catch** (IOException e1) {

JOptionPane.*showMessageDialog*(**null**, "Файл нельзя закрыть!");

**return** **false**;

}

**return** **false**;

}

**if**(*filescan*.hasNextLine())

*filescan*.nextLine();

}/////for вершины

**try** {

*fr*.close();

*filescan*.close();

} **catch** (IOException e) {

JOptionPane.*showMessageDialog*(**null**, "Файл нельзя закрыть!");

**return** **false**;

}

**return** **true**;

}

**static** **void** FileWriting()//Сохранение в файл текущей матрицы смежности

{

**try** {

*fw* = **new** FileWriter(*fileChooser*.getSelectedFile() + ".txt");

} **catch** (Exception e) {

JOptionPane.*showMessageDialog*(**null**, "Нельзя открыть файл для сохранения!");

**return**;

}

**try** {

*fw*.write(*n* + "\n");

**for**(**int** i = 0; i < *AdjTable*.getColumnCount(); i++)

{

**for**(**int** j = 0; j < *AdjTable*.getColumnCount(); j++)

{

**if**(*AdjTable*.getValueAt(i, j) == **null**)

*fw*.write(" ");

**else**

*fw*.write(*AdjTable*.getValueAt(i, j) + " ");

}

*fw*.write("\n");

}

} **catch** (IOException e1) {

JOptionPane.*showMessageDialog*(**null**, "В файл сохранения нельзя записать!");

**try** {

*fw*.close();

} **catch** (IOException e) {

JOptionPane.*showMessageDialog*(**null**, "Файл сохранения нельзя закрыть!");

**return**;

}

**return**;

}

**try** {

*fw*.close();

} **catch** (IOException e) {

JOptionPane.*showMessageDialog*(**null**, "Нельзя закрыть файл для сохранения!");

**return**;

}

}

**static** **void** DepthCycle()//основной цикл поиска фунд. циклов

{

**try** {

*fw*.write("НАЧАЛО АЛГОРИТМА!\n");

*resultText*.setText("");

*Mark* = **new** **int**[*n*];//задаем всем вершинам начальные метки 0

*count* = 0;//обнуляем счетчик меток вершин

*jC* = 0;//обнуляем счетчик циклов

*fw*.write("Число вершин: " + *n* + "\n");

*fw*.write("Задаем всем вершинам начальные метки 0\n");

**int** countKomp = 0; //Добавка//счетчик компонент связности

*fw*.write("Берем вершину с минимальной пометкой и начальной меткой 0 в каждой компоненте связности и начинаем поиск в глубину\n");

**for**(**int** v = 0; v < *n*; v++)

{

**if**(*Mark*[v] == 0)

{

/\* Добавка

\* Добавляем вывод номеров компонент связности, если граф несвязный

\*/

countKomp++;//Добавка//

*fw*.write("----------------------------------------\n" + countKomp + "-я комонента связности\n----------------------------------------\n");

*fw*.write("Вершина " + (v + 1) + " имеет начальную пометку 0. Значит выбираем ее.\n");

*resultText*.append("----------------------------------------\n" + countKomp + "-я комонента связности\n----------------------------------------\n");

*nC*++;

*fw*.write("Вершину " + (v+1) + " добавляем в стек выделения циклов графа\n");

*C*[*nC*] = v;//добавляем вершину в стек выделение циклов графа

*fw*.write("Стек: " + *stackString*() + "\n");

*Cycle*(v,0);

*fw*.write("Удаляем вершину " + (v+1) + " из стека, так как она уже иследована\n");

*nC*--;

*fw*.write("Стек: " + *stackString*() + "\n");

}

*fw*.write("Вершина " + (v+1) + " имеет начальную пометку >0\n");

}

*fw*.write("КОНЕЦ АЛГОРИТМА!\n\n\n");

*fw*.write("РЕЗУЛЬТАТ РАБОТЫ АЛГОРИТМА!\n");

*fw*.write(*resultText*.getText());

} **catch** (IOException e) {

JOptionPane.*showMessageDialog*(**null**, "В файл сохранения нельзя записать!");

**try** {

*fw*.close();

} **catch** (IOException e1) {

JOptionPane.*showMessageDialog*(**null**, "Файл сохранения нельзя закрыть!");

**return**;

}

**return**;

}/////

}

**static** **void** Cycle(**int** x, **int** y)//рекурсивная функция поиска фунд. циклов

{

**try**

{

*count*++;

*Mark*[x] = *count*; //вершина исследована

*fw*.write("Вершине " + (x+1) + " даем начальную метку " + *count* + "\n");

*fw*.write("Далее переходи к смежным ей вершинам, если они есть\n");

**for**(**int** i = 1; i < *AdjTable*.getColumnCount(); i++)

{

**if**(*AdjTable*.getValueAt(x + 1, i).toString().equals("1"))

{

**int** v = i - 1;

*fw*.write("Вершинa " + (x+1) + " смежна вершине " + (v+1) + "\n");

**if**(v == y)////new// v == y исключаем возвращение назад в вершину которую уже прошли

{

*fw*.write("Эту вершину пропускаем так как мы из нее пришли\n");

**continue**;

}//new

**if**((*Mark*[v] > *Mark*[x]))///new

{

*fw*.write("Эту вершину пропускаем так как она имеет большую начальную метку, а значит она уже исследована\n");

**continue**;

}///new

*fw*.write("Вершину " + (v+1) + " добавляем в стек выделения циклов графа\n");

*nC*++;

*C*[*nC*] = v; //вершину в стек

*fw*.write("Стек: " + *stackString*() + "\n");

**if**(*Mark*[v] == 0)

{

*fw*.write("Вершинa " + (v+1) + " имеет начальную метку 0, поэтому далее выбираем ее\n");

*Cycle*(v,x);

}

**else**

{

*fw*.write("Вершинa " + (v+1) + " имеет начальную метку >0, из этого следует, что \n");

*fw*.write("найдено обратное ребро (" + (x+1) + "," + (v+1) + "), а значит найден фундаментальный цикл:\n");

*jC*++; //обратное ребро (x,v), найден цикл

*WriteCycle*(v, *C*, *nC*);//запись цикла в файл

}

*fw*.write("Удаляем вершину " + (v+1) + " из стека, так как она уже исследована\n");

*nC*--;//удаляем исследованную вершину из стека

*fw*.write("Стек: " + *stackString*() + "\n");

}

}

} **catch** (IOException e) {

JOptionPane.*showMessageDialog*(**null**, "В файл сохранения нельзя записать!");

**try** {

*fw*.close();

} **catch** (IOException e1) {

JOptionPane.*showMessageDialog*(**null**, "Файл сохранения нельзя закрыть!");

**return**;

}

**return**;

}/////

}

**static** **void** WriteCycle(**int** x, **int**[] C, **int** nC)

{

**try** {

*fw*.write(*jC* + ") ");//печать номера цикла

*resultText*.append(*jC* + ") ");

**do**

{

*fw*.write(" " + (C[nC] + 1));//печать вершины из стека

*resultText*.append(" " + (C[nC] + 1));

nC--;

}**while**(!(C[nC] == x));

*fw*.write("\n");

*resultText*.append("\n");

} **catch** (IOException e) {

JOptionPane.*showMessageDialog*(**null**, "В файл сохранения нельзя записать!");

**try** {

*fw*.close();

} **catch** (IOException e1) {

JOptionPane.*showMessageDialog*(**null**, "Файл сохранения нельзя закрыть!");

**return**;

}

**return**;

}

}

**static** **void** matrixOrder(**int** order)//задание порядка матрицы

{

*AdjTableModel*.setRowCount(order);

*AdjTableModel*.setColumnCount(order);

}

**static** **void** quadrMatrix()//функция которая делает ячейки таблицы в виде квадратов

{

**for**(**int** p = 0; p < *AdjTable*.getColumnCount(); p++)

{

*AdjTable*.getColumnModel().getColumn(p).setPreferredWidth(25);

}

}

**static** **public** **boolean** isNeoGraph()//функция проверки является ли граф неориентированным

{

**for**(**int** v = 1; v < *AdjTable*.getRowCount(); v++)

{

**for**(**int** adj = 1 + v; adj < *AdjTable*.getColumnCount(); adj++)

{

**if**(!(*AdjTable*.getValueAt(v, adj).toString().equals(*AdjTable*.getValueAt(adj, v).toString())))

{

**return** **false**;

}

}

}

**return** **true**;

}

**static** String stackString()//функция печати стека выделения циклов графа

{

String str = **new** String();

str+="[";

**for**(**int** i = 0; i <= *nC*; i++)

{

**if**(i == *nC*)

str+= " " + (*C*[i] + 1);

**else**

str+= " " + (*C*[i] + 1) + ",";

}

str+="]";

**return** str;

}

}