МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ ДНР
ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

ОТЧЁТ ПО ИНДИВИДУАЛЬНОМУ ЗАДАНИЮ КУРСА «КОМПЬЮТЕРНАЯ ДИСКРЕТНАЯ МАТЕМАТИКА»

Выполнила:
стгр. ПИ-16Б
Мамутова В. А.

Проверила:
асс. каф. ПИ
Незамова Л.В.

Донецк – 2017

РЕФЕРАТ

Отчет по курсовой работе содержит: 34 страниц, 26 рисунков, 3 приложений, 3 источников.

Объект исследования — правильная раскраска неориентированных и ориентированных графов.

Цель — реализовать алгоритм нахождения правильной раскраски графа в виде программы, обеспечить визуализацию и вывод результата работы алгоритма.

Результат — программная реализация нахождения правильной раскраски графа, вывод результата, запись результата в файл.

ПРАВИЛЬНАЯ РАСКРАСКА, ТЕОРИЯ ГРАФОВ, НЕОРИЕНТИРОВАННЫЙ ГРАФ, ОРИЕНТИРОВАННЫЙ ГРАФ, АЛГОРИТМ.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Дискретная математика – это цикл математических наук, изучающих свойства конечных множеств. Это часть математики, изучающая дискретные математические структуры, такие, как графы и утверждения в логике. В контексте математики, в целом, дискретная математика часто отождествляется с конечной математикой — направлением, изучающим конечные структуры — конечные графы, конечные группы, конечные автоматы.

В настоящее время эти науки бурно развиваются, что определяется тремя очень важными факторами:

1) развитием компьютерной техники и компьютерных наук, которые базируются, а по существу являются продолжением дискретной математики;
2) запросами различных прикладных наук – теории управления, экономики, оптимизации и многих, многих других;
3) логикой внутреннего развития этих наук. Появлением новых разделов, глубоких интересных проблем, развитием мощных методов их решения.

Все это и предопределило тот факт, что различные разделы дискретной математики все настойчивее внедряются не только в университеты, но и в технические и экономические вузы и даже в гуманитарные.

1 ОПИСАНИЕ АЛГОРИТМА ДЛЯ НАХОЖДЕНИЯ ПРАВИЛЬНОЙ РАСКРАСКИ

Раскраска графов – это присвоение маркировки (цветов, чисел, букв) либо вершинам, либо рёбрам графа. В зависимости от объекта раскрашивания различают вершинную и рёберную раскраску. В работе будем рассматривать вершинную раскраску графа.

Пусть G = (V, E) − простой неориентированный граф, k − натуральное число. Вершинной k-раскраской, или просто k-раскраской графа G, называется произвольная функция f, отображающая множество вершин графа G в некоторое k-элементное множество: f: VG → $\{a_1, a_2, \ldots, a_k\}$ = A. Если для некоторой вершины v графа G: f(v)=i, то говорят что вершина v раскрашена в i-тый цвет. Раскраска называется правильной, если f(u) ≠ f(v) для любых смежных вершин u и v графа G (или концевые вершины любого ребра окрашены в разные цвета). Граф, для которого существует правильная k-раскраска, называется k-раскрашиваемым.

Хроматическое число графа G − это минимальное число красок, при котором граф имеет правильную раскраску. Если хроматическое число равно k, то граф называется k-хроматическим (обозначают $\chi(G) = k$). Правильную k-раскраску графа G можно рассматривать как разбиение множества вершин графа G на не более чем k непустых множеств, которые называются цветными классами.

Алгоритм последовательной раскраски:

1. Произвольной вершине графа G приписываем цвет 1.

2. Пусть раскрашены i вершин графа G в цвета от 1 до k, где $k \leq i$. Произвольной неокрашенной вершине v_{i+1} приписываем минимальный цвет, неиспользованный при раскраске смежных с ней вершин. Алгоритм последовательной раскраски зависит от способа выбора вершин на обслуживание.

Текст алгоритма нахождения правильной вершинной раскраски графа представлен на рис. 1.1.

```
Const Nmax=100;  {*Максимальное количество вершин
                   графа.*}
Type  V=0..Nmax;
      Ss=Set Of V;
      MyArray=Array[1..Nmax] Of V;
      Var Gr:MyArray; {*Gr - каждой вершине графа
                 определяется номер цвета. *}

<формирование описания графа>;
For i:=l To N Do Gr[i] : = [Color (i ,0) ] ;
<вывод решения>;
```

Поиск цвета раскраски для одной вершины можно реализовать с помощью следующей функции:

```
Function   Color(i,t:V):Integer;{*i номер
     окрашиваемой вершины, t - номер цвета,
     с которого следует искать раскраску данной
     вершины, A - матрица смежности, Gr -
     результирующий массив. *}
 Var  Ws:Ss;
     j:Byte;
Begin
 Ws:=[ ];
 For j:=l To i-1 Do If A[j,i]=1Then Ws:=Ws+[Gr[j]] ;
        {*Формируем множество цветов, в которые
        окрашены смежные вершины с меньшими
        номерами. *}
 j:=t;
 Repeat {*Поиск минимального номера цвета,
       в который можно окрасить данную вершину.*}
   Inc(j)  ;
 Until Not(j In Ws) ;
 Color:=j;
End;
```

Рисунок 1.1 – Исходный алгоритм

В ходе работы над программой были внесены некоторые изменения, которые приведены на рис. 1.2.

```
    private int[] vertexesColor; // Массив цветов для окрашивания вершин.

    ссылка: 1
    private void GetGraphColoring_Click(object sender, EventArgs e)
    {
        vertexesColor = new int[AdjacencyMatrix.RowCount]; // Массив цветов для вершин.

        // Инициализируем массив цветов для вершин.
        for (int i = 0; i < vertexesColor.Length; i++)
            vertexesColor[i] = -1;

        // Для каждой вершины находим цвет.
        for (int i = 0; i < AdjacencyMatrix.RowCount; i++)
            vertexesColor[i] = GetColor(i, vertexesColor);
    }

    // Получение цвета вершины.
    ссылка: 1
    int GetColor(int i, int[] vertexesColor)
    {
        // Список цветов вершин, смежных с рассматриваемой.
        List<int> colorList = new List<int>();
        int j; // Временная переменная для прохождения по вершинам графа.

        // Добавляем в список порядковые номера цветов,
        // в которые окрашены смежные вершины.
        for (j = 0; j < i; j++)
        {
            if (AdjacencyMatrix.Rows[j].Cells[i].Value.ToString() != "0")
                colorList.Add(vertexesColor[j]);

            // Для орграфов проверяем дополнительно симметричную позицию.
            else if (GraphType.SelectedIndex == 1 &&
                AdjacencyMatrix.Rows[i].Cells[j].Value.ToString() != "0")
    }

    j = -1;

    // Находимый минимальный допустимый цвет.
    do
        j++;
    while (colorList.FindIndex(x => x.Equals(j)) != -1);
    return j;
}
```

Рисунок 1.2 – Измененный алгоритм

Основное изменение в алгоритме связано с необходимостью применения данного алгоритма на примерах ориентированных графов. Так как матрица смежности орграфа не симметрична, то поиск смежных вершин должен производиться не только по ячейкам, находящимся ниже главной диагонали матрицы, но и по симметричным ячейкам.

Более того, для хранения номера цветов смежных вершин была использована другая структура данных – объект класса List<T>, представляющий собой строго типизированный список объектов, доступных по индексу. Списки - очень удобный инструмент для программирования особенно там, где количество элементов в коллекции наперёд не известно. Они выгодно отличаются от массивов тем, что по ходу выполнения программы их размер можно изменять в любую сторону. Данная особенность как раз и является основным требованием списка цветов смежных вершин для заданной, так количество таких вершин перед просмотром необходимой вершины заранее неизвестно. Динамическое добавление элементов в список является решением, экономящим объём оперативной памяти, необходимой для нормального функционирования разработанного приложения.

Листинг программы с подробными комментариями представлен в Приложении Б.

2 РУЧНОЙ РАСЧЁТ КОНТРОЛЬНОГО ПРИМЕРА

Рассмотрим неориентированный граф на 6 вершин (рис. 2.1).

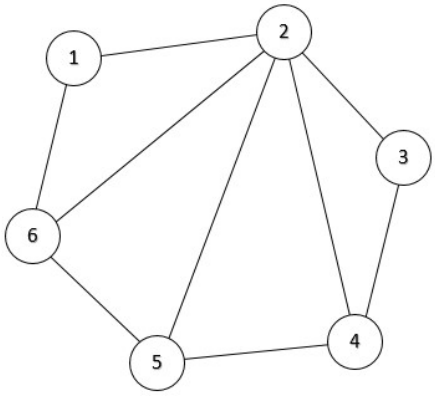

Рисунок 2.1 – Исходный граф

Начинаем присваивать цвета вершинам алгоритмом последовательной раскраски.

1. Первую вершину окрашиваем в цвет 1.

2. Переходим к v_2. Определяем список смежных вершин, уже окрашенных в цвет. Смежная вершина – v_1 с цветом 1. Минимально допустимый цвет – цвет 2.

3. Список вершин, смежных с v_3 и уже окрашенных:

- v_2 – цвет 2.

Минимально допустимый цвет – 1.

4. Список вершин, смежных с v_4 и уже окрашенных:

- v_2 – цвет 2;
- v_3 – цвет 1.

Минимально допустимый цвет – 3.

5. Список окрашенных, смежных с v_5 вершин:

- v_4 – цвет 3.

Минимально допустимый цвет – 1.

6. Список окрашенных, смежных с v_6 вершин:

- v_1 – цвет 1.

- v_2 – цвет 2.

- v_4 – цвет 1.

Минимально допустимый цвет – 3.

7. Итак, все вершины рассмотрены. Получаем правильную раскраску: v_1, v_3, v_5 – цвет 1; v_2 – цвет 2; v_4, v_6 – цвет 3. Всего использовано три цвета для раскраски.

Найдём верхнюю и нижнюю оценку хроматического числа.

Нижняя оценка: $\chi(G) \geq \frac{p^2}{p^2-2p}$. $\chi(G) \geq \frac{6^2}{6^2-2*6}$. $\chi(G) \geq 2$.

Верхняя оценка: $\chi(G) \leq \Delta(G) + 1$, где $\Delta(G)$ – максимум из степеней вершин. $\chi(G) \leq 5 + 1$. $\chi(G) \leq 6$.

Получаем следующий диапазон допустимого хроматического числа:

$2 \leq \chi(G) \leq 6$.

Полученное значение количества вершин для правильной раскраски попадает в диапазон верхней и нижней оценки хроматического числа.

Для данного графа хроматическое число равно 3 ($\chi(G) = 3$).

На рис. 2.2 представлена правильная раскраска для исходного неориентированного графа на 6 вершин.

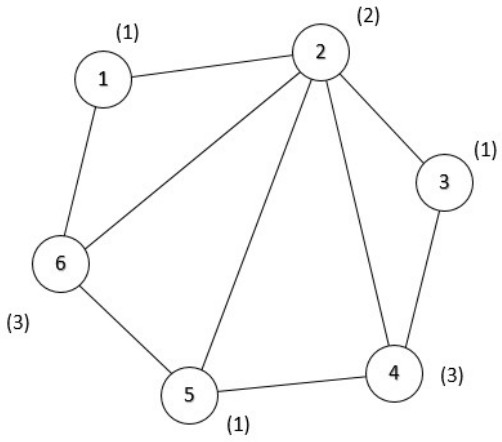

Рисунок 2.2 – Правильная раскраска для исходного графа

3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ КОНТРОЛЬНОГО ПРИМЕРА

На рис. 3.1 представлена экранная форма во время выполнения задания по контрольному примеру.

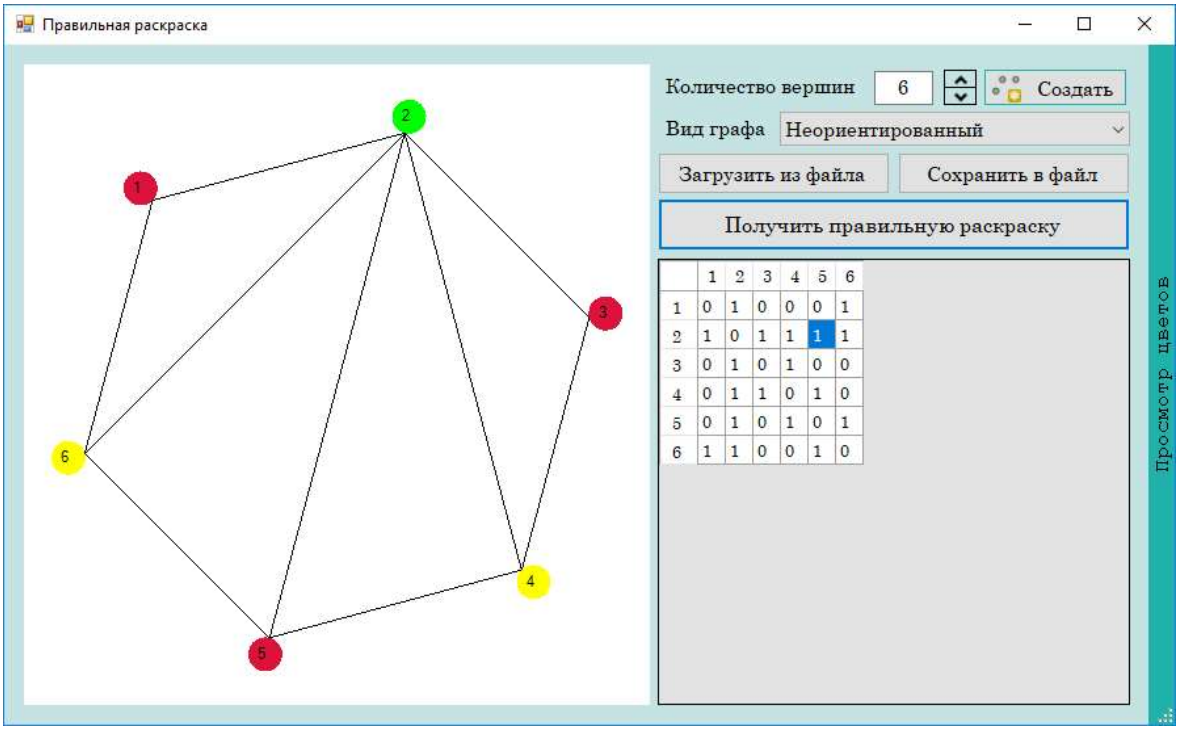

Рисунок 3.1 – Решение контрольного примера в программе

В данном примере темно-розовый цвет имеет первый порядковый номер, зелёный – второй, жёлтый – третий.

Список допустимых цветов, в которые окрашиваются вершины на рисунке, формируется при запуске программы. Названия цветов, определяемых в программе по умолчанию, содержатся в файле «Colors.txt». По завершению алгоритма, как было рассмотрено ранее, создаётся список порядковых номеров цветов. Далее, при добавлении вершины графа на холст каждая из них окрашивается в оттенок, стоящий под определённым по алгоритму.

В правой части окна приложения находится полоса «Просмотр цветов», после наведения на которую можно просмотреть список цветов с их нумерацией. В программе также предусмотрена возможность изменения цветов пользователем. Подробная инструкция представлена в приложении А, панель со списком цветов – в приложении В.

На рис. 3.2 показан сохранённый в файл «Правильная раскраска неографа на 6 вершин.txt» результат работы алгоритма.

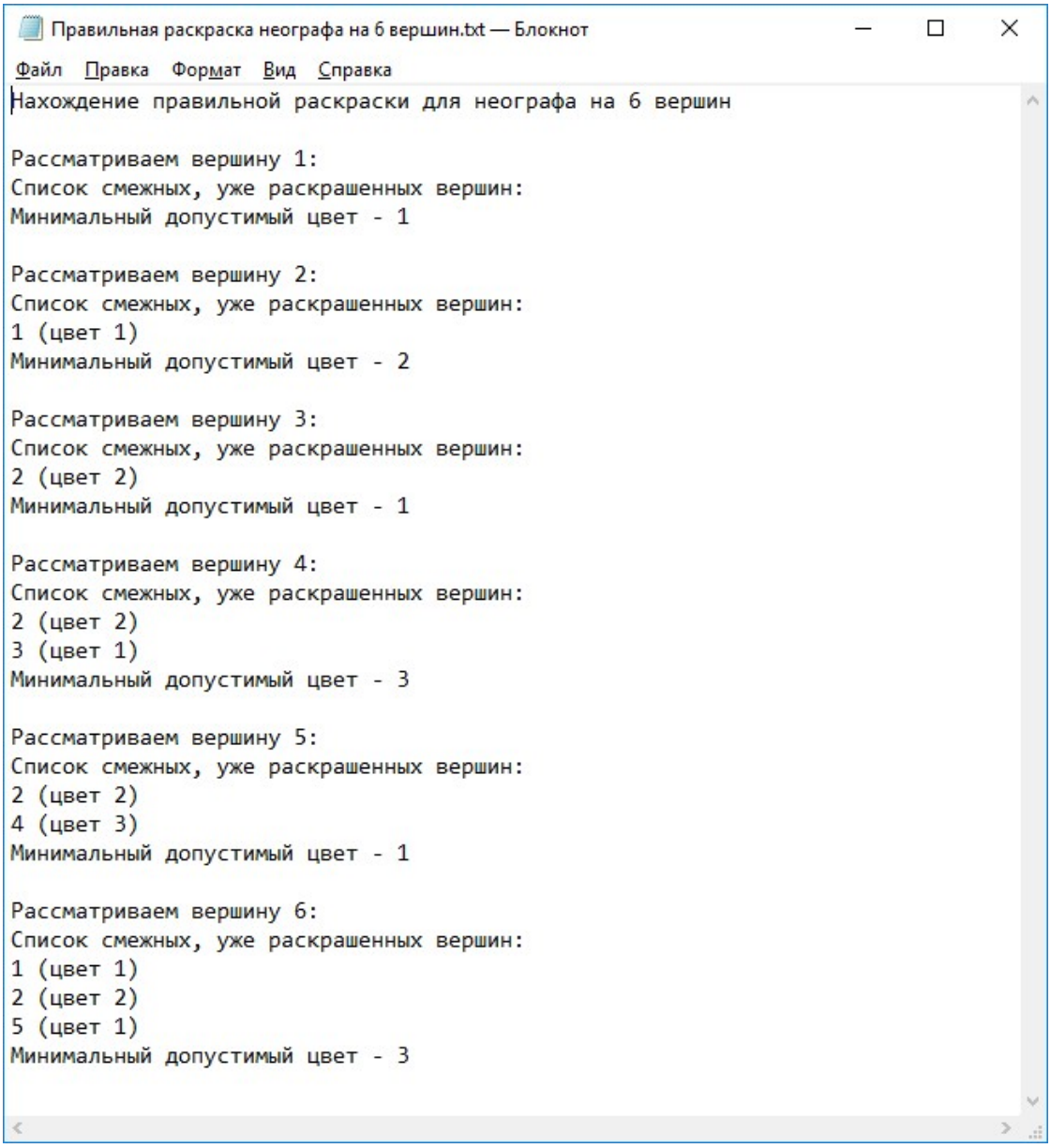

Рисунок 3.2 – Файловое представление результата работы программы

4 РЕАЛИЗАЦИЯ ИСКЛЮЧИТЕЛЬНЫХ СЛУЧАЕВ

Для лучшего ознакомления пользователя с работой данной программы в приложении В представлены все экранные формы, сделанные во время выполнения программы.

На рис. 4.1 представлен результат алгоритма на пустом графе. Так как все вершины не являются смежными, то хроматическое число равно единице. Это также утверждает лемма о 2-х раскрашиваемых графах.

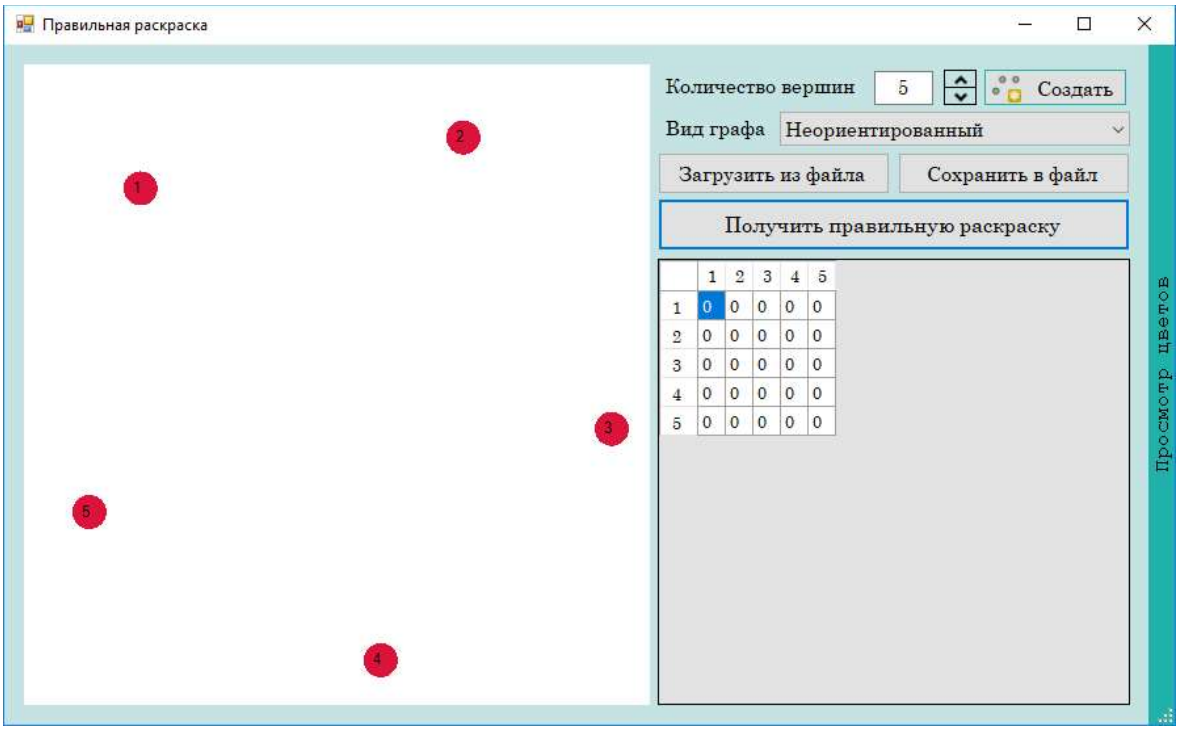

Рисунок 4.1 – Пустой граф (O_5)

На рис. 4.2 представлен результат алгоритма на тривиальном графе. Аналогично пустому графу, $\chi(G) = 1$.

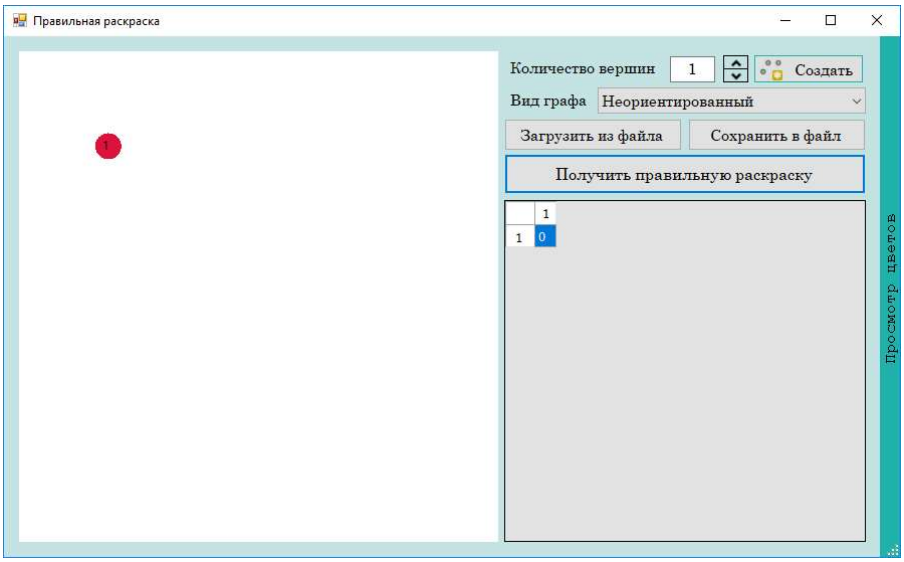

Рисунок 4.2 – Тривиальный граф (K_1)

На рис. 4.3 представлен результат алгоритма на графе, содержащем простую цепь. $\chi(G) = 2$, что соответствует лемме о 2-х раскрашиваемых графах.

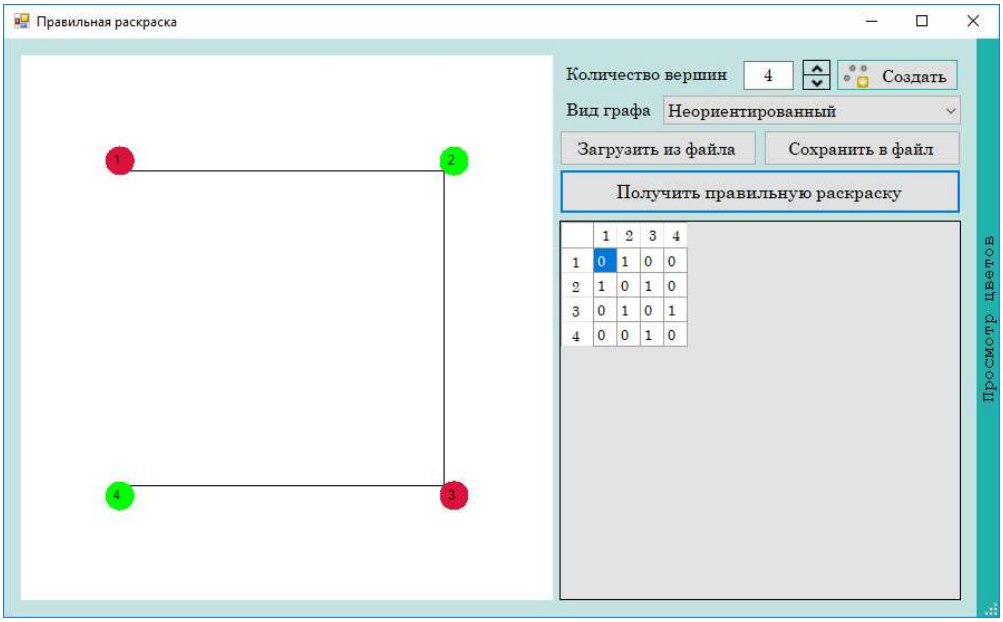

Рисунок 4.3 – Граф, содержащий простую цепь (P_4)

На рис. 4.4 представлен результат алгоритма на простом цикле. По лемме о раскраске циклов: «Хроматическое число всякого цикла, содержащего p вершин, равно 2, если p − четно, и 3, если p – нечетно». В данном графе 5 вершин, $\chi(G) = 3$ – верно.

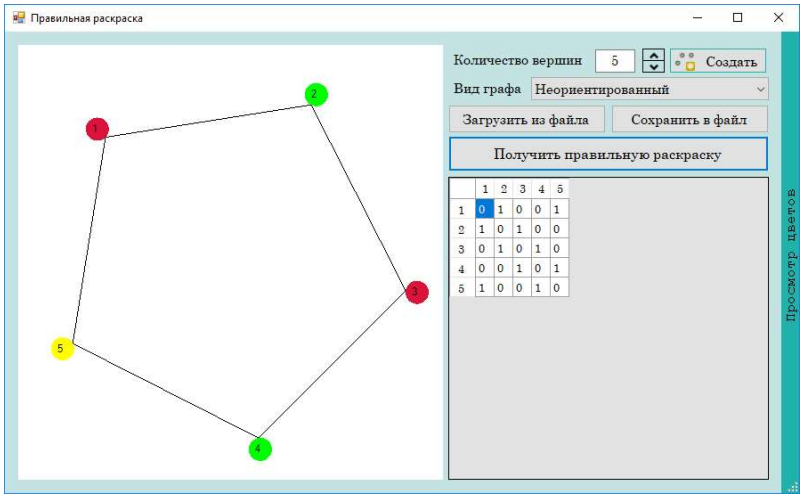

Рисунок 4.4 – Простой цикл (C_5)

На рис. 4.5 представлен результат алгоритма на графе, содержащем 3 компоненты. Для произвольного графа G справедливо неравенство $\chi(G) \geq \boldsymbol{\varphi}(\mathbf{G})$, где $\boldsymbol{\varphi}(\mathbf{G})$ − плотность графа или кликовое число. В данном примере кликовое число равно трём, оно и является хроматическим. $\chi(G) = 3$.

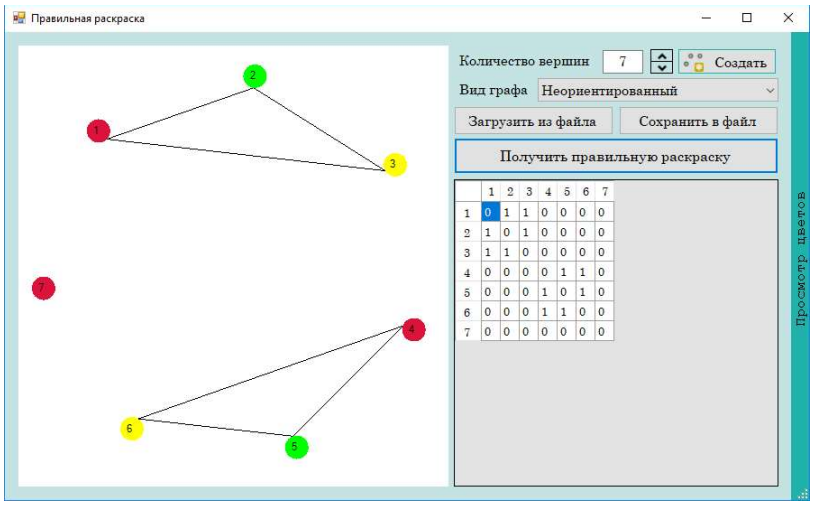

Рисунок 4.5 – Несвязный граф (3 компоненты)

На рис. 4.6 представлен результат алгоритма на полном графе. По лемме о раскраске полного графа: «Хроматическое число полного графа K_p равно p. Если граф G содержит подграф изоморфный графу K_p, то $\chi(G) \geq p$». $\chi(G) = p = 5$.

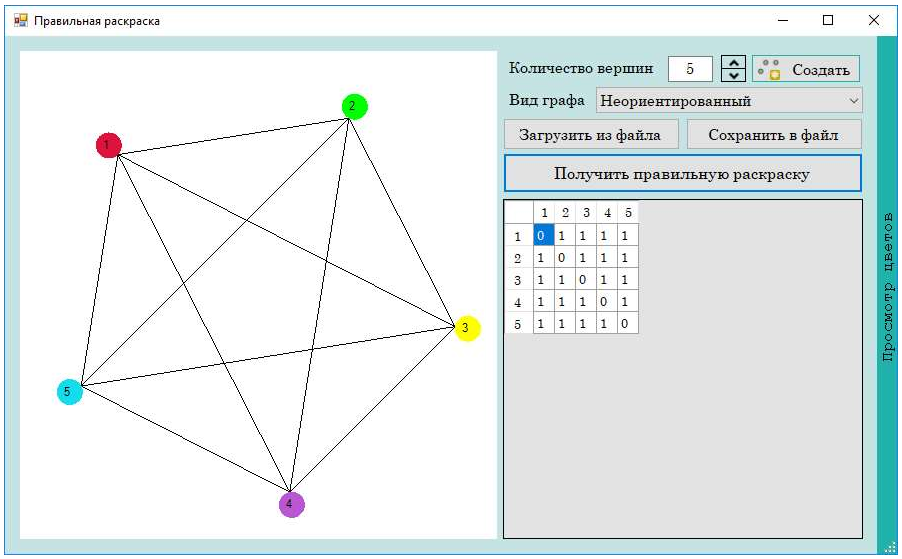

Рисунок 4.6 – Полный граф (K_5)

На рис. 4.7 представлен результат алгоритма на графе-звезде. Звезда является двудольным графом, значит, из следствия теоремы Кёнинга, данный граф является бихроматическим, то есть графом, у которого $\chi(G) = 2$.

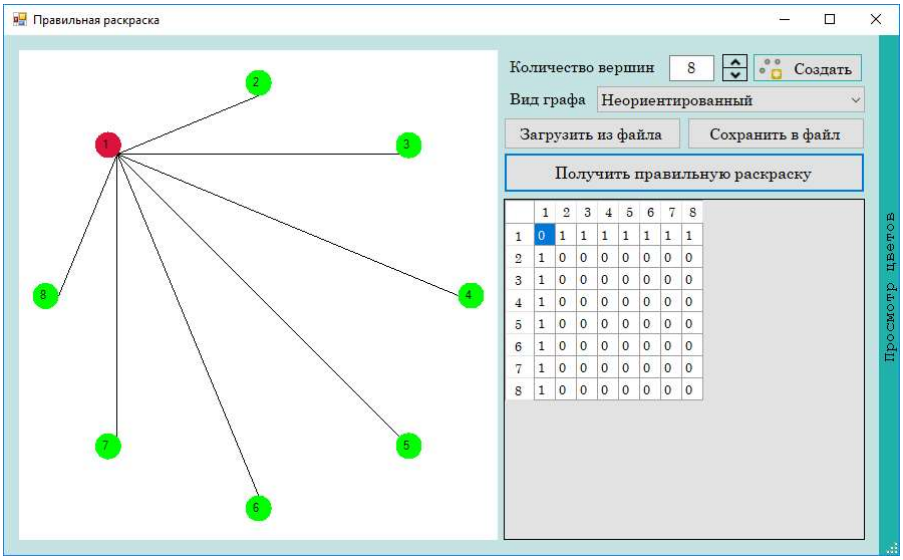

Рисунок 4.7 – Звезда ($K_{1,7}$)

На рис. 4.8 представлен результат алгоритма на двудольном графе. Граф является бихроматическим, то есть $\chi(G) = 2$.

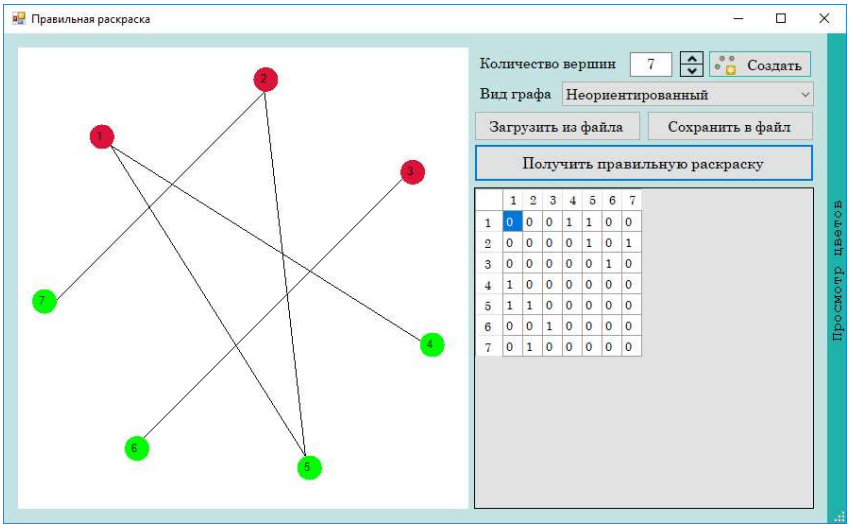

Рисунок 4.8 – Двудольный граф ($K_{3,4}$)

На рис. 4.9 представлен результат алгоритма на полном двудольном графе. Аналогично, из следствия теоремы Кёнинга, граф является бихроматическим, то есть $\chi(G) = 2$.

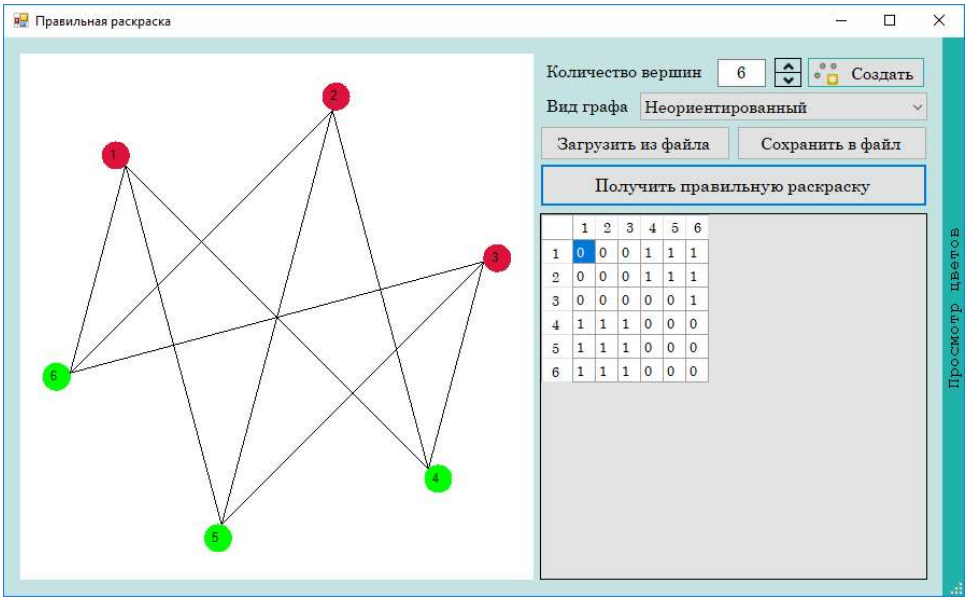

Рисунок 4.9 – Полный двудольный граф ($K_{3,3}$)

На рис. 4.10 представлен результат алгоритма на ориентированном графе. $\chi(G) = 4$..

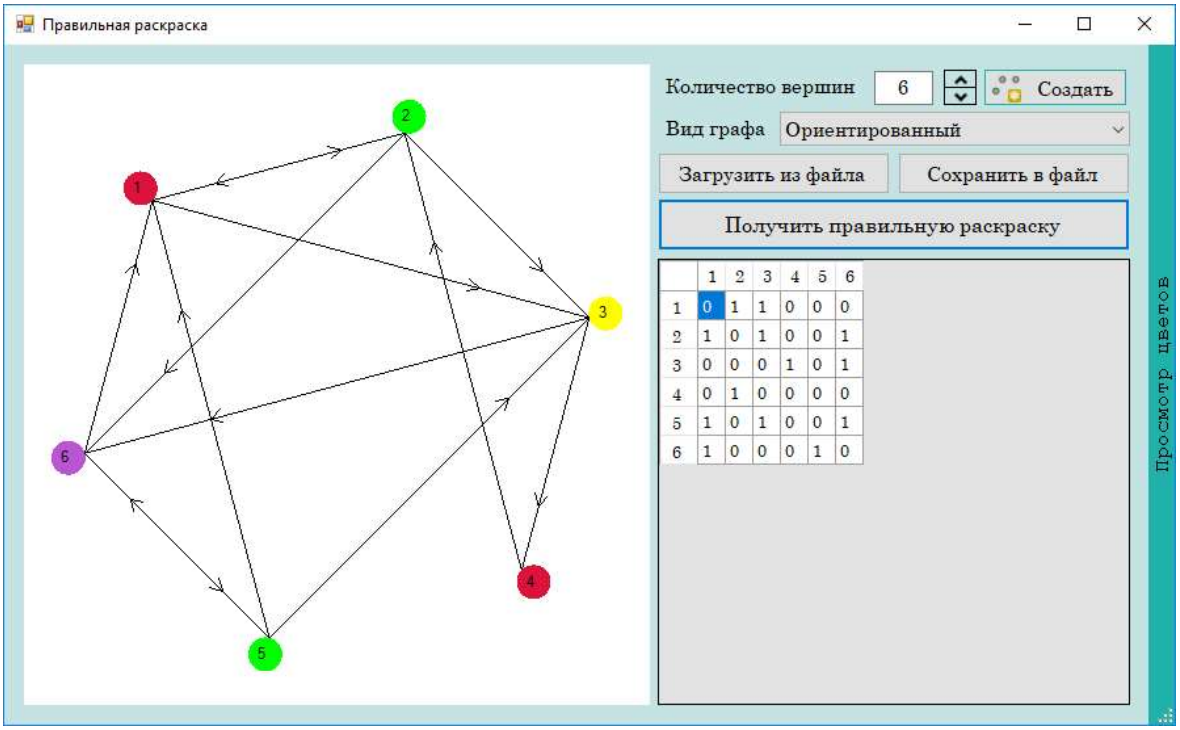

Рисунок 4.10 – Ориентированный граф

5 ОПИСАНИЕ ПРОГРАММЫ

Программа Индивидуальное задание по КДМ (Мамутова В.).exe предназначена для моделирования базовых операций при работе с графом, а также для визуального и файлового представления работы алгоритма нахождения правильной раскраски.

Программа может быть использована для использования в высших учебных заведениях с целью демонстрации алгоритма нахождения правильной раскраски графа.

Для запуска программы требуется персональный компьютер с частотой процессора не менее 1.2 ГГц и объемом оперативной памяти не менее 50 Мб.

Вызов и загрузка программы осуществляется путем запуска исполняемого файла Индивидуальное задание по КДМ (Мамутова В.).exe.

В качестве исходных данных программа использует данные, считываемые из внешних файлов, а также вводимые пользователем с клавиатуры.

Выходные данные выводятся на экран и в файл, путь к которому и его имя может быть задано как программно, так и вследствие указания пользователем в специальном диалоговом окне.

ВЫВОДЫ

В ходе разработки программы был изучен алгоритм нахождения правильной раскраски графа, выполнена запись шагов алгоритма в файл и представлен вывод результата на экране. Реализация алгоритма была осуществлена в среде разработки Visual Studio 2015.

Разработанная программа Индивидуальное задание по КДМ (Мамутова В.).exe является системой для моделирования алгоритма нахождения правильной раскраски.

Данное приложение может быть использовано в высших учебных заведениях с целью демонстрации и лучшего понимания данного алгоритма студентами.

Графы достаточно широко применяются в математике, технике, экономике, управлении. Знание основ теории графов необходимо в различных областях, связанных с управлением производством, бизнесом (например, сетевой график строительства, графики доставки почты).

Раскрашивание вершин, в свою очередь, решает многие задачи планирования (распределение самолетов по рейсам), используется в распределении регистров для ускорения времени выполнения результирующего кода компилятором. Также раскраска применяется в технологии цифровых водяных знаков. С помощью данного алгоритма можно закодировать сообщение в числе и в способе распределения регистров, извлечь это сообщение путём сравнения распределения регистров с исходной раскраской. Помимо этого, решение головоломки Судоку можно рассматривать как завершение раскраски 9 цветами заданного графа из 81 вершины.

В дальнейшем можно расширить функционал данной программы и усовершенствовать её. Например, разработать более удобный и понятный графический пользовательский интерфейс: добавить возможность

визуального создания графа, перемещения вершин по холсту, удаления рёбер. Более того, в целях облегчения работы пользователя и уменьшения количества вводимой информации, создать вкладку с шаблонами специальных графов. С помощью таких шаблонов можно будет достаточно быстро генерировать графы в зависимости от требований пользователя.

ПЕРЕЧЕНЬ ССЫЛОК

1. Донец Г.А. Алгебраический подход к проблеме раскраски плоских графов / Г. А. Донец, Н.З. Шор. - Киев: Наукова думка, 1982. – 144 с.
2. Окулов. С. М. Программирование в алгоритмах \ С. М. Окулов. - М.: БИНОМ. Лаборатория знаний, 2002. – 341 с: ил.
3. Раскраски [Электронный ресурс]. – Режим доступа: http://rain.ifmo.ru/cat/view.php/theory/graph-coloring-layout/coloring.

ПРИЛОЖЕНИЕ А
ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЮ

Для начала работы программы нужно совершить её запуск, открыв файл Индивидуальное задание по КДМ (Мамутова В.).exe. Рекомендуется прочитать справку перед началом работы программы. Пользуясь информацией из справки, можно начинать работу.

Правая часть формы приложения предназначена для работы с графом. Здесь пользователь может создать граф на определённое количество вершин и модифицировать его. Для того, чтобы создать граф, необходимо ввести количество вершин в специальное поле, выбрать вид графа и нажать кнопку «Создать» или «Enter». Также можно загрузить матрицу смежности из файла, при этом определится количество вершин и вид графа в зависимости от симметричности матрицы и заполнятся соответствующие поля.

Для изменения количества вершин можно нажать кнопки со стрелочками вверх/вниз либо стрелки вверх/вниз на клавиатуре для добавления/удаления соответственно вершины с наибольшим порядковым номером. При этом граф автоматически перерисовывается, а все рёбра, инцидентные оставшимся вершинам, остаются. Созданный граф можно сохранить в файл, нажав соответствующую кнопку.

В этой же области расположена кнопка для нахождения результата алгоритма («Получить правильную раскраску»). После нажатия данной кнопки происходит расчёт алгоритма, раскрашиваются вершины графа и записываются шаги алгоритма в файл.

В левой части окна расположен холст для отображения графа.

Справа также находится полоса «Просмотр цветов», при наведении на которую отображается скрытая панель со списком цветов и их нумерацией. Нажав на любой оттенок, открывается диалоговое окно, в котором пользователь может изменит текущий цвет.

ПРИЛОЖЕНИЕ Б

ЛИСТИНГ ПРОГРАММЫ С ПОДРОБНЫМИ КОММЕНТАРИЯМИ

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Threading;

namespace
Индивидуальное_задание_по_КДМ__Мамутова_В._
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            ClearImage();
            InitColors();

            IsColorDialogOpen = false;
            GraphHasColoring = false;
            IsLoadingFromFile = true;

            GraphType.SelectedIndex = 0;
            IsLoadingFromFile = false;
        }
        List<Color> colorList; // Список
цветов.
        private int[] vertexesColor; // Массив
цветов для окрашивания вершин.

        Thread drawVertexesThread;
        Thread fillTableThread;

        private bool IsColorDialogOpen; //
Открыт диалог для выбора цвета.
        private bool GraphHasColoring; // Для
текущего графа уже найдена правильная
раскраска.
        private bool IsLoadingFromFile; //
Происходит загрузка из файла.

        // Создание и инициализация списка
цветов по умолчанию.
        void InitColors()
        {
            colorList = new List<Color>();

            // Чтение наименований цветов из
файла.
            string colorName;
            try
            {
                FileStream file = new
FileStream(Application.StartupPath +
"\\Colors.txt", FileMode.Open);
                StreamReader streamReader =
new StreamReader(file, Encoding.UTF8);
                while
(streamReader.EndOfStream != true)
                {
                    colorName =
streamReader.ReadLine();

colorList.Add(Color.FromName(colorName));
                }
                streamReader.Close();
            }
            catch (Exception ex) {
MessageBox.Show(ex.Message); }

            // Выводим цвета в таблицу.
            ColorGrid.RowCount =
colorList.Count;
            for (int i = 0; i <
ColorGrid.RowCount; i++)
            {
                // Ставим порядковый номер
цвета.

ColorGrid.Rows[i].Cells[0].Value = (i +
1).ToString();
                ColorGrid.Rows[i].Height = 40;

                // Окрашиваем ячейки в цвета
списка.

ColorGrid.Rows[i].Cells[1].Style.BackColor =
colorList[i];

ColorGrid.Rows[i].Cells[1].Style.SelectionBack
Color = colorList[i];

ColorGrid.Rows[i].Cells[1].Style.SelectionFore
Color = Color.Black;
            }
        }

        // Смена цвета пользователем.
        private void
ColorGrid_CellContentClick(object sender,
DataGridViewCellEventArgs e)
        {
            if (e.ColumnIndex == 1)
            {
                // Открываем диалоговое окно
для выбора цвета.
                IsColorDialogOpen = true;
                colorDialog.FullOpen = true;
                colorDialog.ShowDialog();

                colorList[e.RowIndex] =
colorDialog.Color; // Меняем цвет в списке.

                // Окрашиваем ячейку в
выбранный цвет.

ColorGrid.Rows[e.RowIndex].Cells[1].Style.Back
Color = colorDialog.Color;

ColorGrid.Rows[e.RowIndex].Cells[1].Style.Sele
ctionBackColor = colorDialog.Color;
                IsColorDialogOpen = false;
            }
        }

        // Очистка холста для графа, заливка
его белым цветом.
        void ClearImage()
        {
            Bitmap graphBitmap = new
Bitmap(GraphImage.Width, GraphImage.Height);
            Graphics g =
Graphics.FromImage(graphBitmap);
            g.Clear(Color.White);
            GraphImage.Image = graphBitmap;
        }

        // Добавление на холст вершин графа.
        void DrawVertexes(Image image, bool
slowDown = false) {
            if(slowDown) Thread.Sleep(500);
            Bitmap graphBitmap = new
Bitmap(image);
```

```
            Graphics g =
Graphics.FromImage(graphBitmap);

            int count =
Convert.ToInt32(VertexCount.Text);
            float radius = GraphImage.Width /
2 - 26; // Радиус большого круга, на
окружности которого будут располагаться
вершины.
            float angleDelta = 360F / count;
// Угол, на который будет изменяться положение
на окружности каждой новой рисуемой вершины.
            float x = 0, y = 0; // Координаты
центра вершины.
            float angle = 135; // Угол, под
которым расположена первая вершина.

            for (int i = 0; i < count; i++) {
                if (i != 0)
                    angle -= angleDelta;

                x = radius *
(float)Math.Cos(angle * 3.14 / 180) +
GraphImage.Width / 2;
                y = -radius *
(float)Math.Sin(angle * 3.14 / 180) +
GraphImage.Height / 2;

                // Опреляем цвет кисти в
зависимости от того, была ли получена раскаска
для текущего графа.
                Brush brush = new
SolidBrush(GraphHasColoring ?
colorList[vertexesColor[i]] :
Color.LightPink);

                g.FillEllipse(brush, x - 13, y
- 13, 26, 26);
                g.DrawString((i +
1).ToString(), DefaultFont, Brushes.Black, x -
7, y - 7);

                Invoke((MethodInvoker)
delegate () { GraphImage.Image = graphBitmap;
});

                if(slowDown)
                    Thread.Sleep(100);
            }
        }

        // Добавление ребра на холст.
        void DrawEdge(int vertex1, int
vertex2, Graphics g)
        {
            float angleDelta = 360F /
AdjacencyMatrix.RowCount;
            float x1 = 0, x2 = 0, y1 = 0, y2 =
0; // Координаты начала и конца отрезка ребра.
            float angle = 135; // Угол, под
которым расположена первая вершина.
            float radius = GraphImage.Width /
2 - 26; // Радиус большого круга, на
окружности которого будут располагаться
вершины.

            // Находим угол, под котором
расположена первая вершина.
            for (int k = 0; k < vertex1; k++)
                angle -= angleDelta;

            float cos = (float)Math.Cos(angle
* 3.14 / 180);
            float sin = (float)Math.Sin(angle
* 3.14 / 180);

            // Вычисляем координаты первой
вершины.
            x1 = radius * cos +
GraphImage.Width / 2 - 13 * cos;
            y1 = -radius * sin +
GraphImage.Height / 2 + 13 * sin;

            // Находим угол, под которым
расположена вторая вершина.
            if (vertex1 < vertex2)
                for (int k = vertex1; k <
vertex2; k++)
                    angle -= angleDelta;
            else
                for (int k = vertex1; k >
vertex2; k--)
                    angle += angleDelta;

            cos = (float)Math.Cos(angle * 3.14
/ 180);
            sin = (float)Math.Sin(angle * 3.14
/ 180);

            // Вычисляем координитаты второй
вершины.
            x2 = radius * cos +
GraphImage.Width / 2 - 13 * cos;
            y2 = -radius * sin +
GraphImage.Height / 2 + 13 * sin;

            g.DrawLine(Pens.Black, x1, y1, x2,
y2);

            // Рисуем стрелочку для оргграфов.
            if (GraphType.SelectedIndex == 1)
            {
                x2 = (((x1 + x2) / 2) + x2) /
2;
                y2 = (((y1 + y2) / 2) + y2) /
2;

                float exAngle = (float)(180 *
Math.Atan2(y2 - y1, x2 - x1) / 3.14);

                Point p1 = new Point((int)(x2
+ (10 * Math.Cos(3.14 * (exAngle + 150) /
180))), (int)(y2 + (10 * Math.Sin(3.14 *
(exAngle + 150) / 180))));
                Point p2 = new Point((int)(x2
+ (10 * Math.Cos(3.14 * (exAngle - 150) /
180))), (int)(y2 + (10 * Math.Sin(3.14 *
(exAngle - 150) / 180))));
                g.DrawLine(Pens.Black, x2, y2,
p1.X, p1.Y);
                g.DrawLine(Pens.Black, x2, y2,
p2.X, p2.Y);
            }
        }

        // Добавление рёбер для графа.
        void DrawEdges(bool slowDown = false)
        {
            Bitmap graphBitmap = new
Bitmap(GraphImage.Image);
            Graphics g =
Graphics.FromImage(graphBitmap);

            bool edgeWasPainted = false;
            for (int i = 0; i <
AdjacencyMatrix.RowCount; i++)
                for (int j = 0; j <
AdjacencyMatrix.ColumnCount; j++)
                {
                    // Для неорграфов
достаточно пройти только по одной части
матрицы
                    // для проведения ребёр
(выше главной диагонали),
                    // для оргграфов проверяем
таблицу полноситью, так как она не
симметрична.

Invoke((MethodInvoker)delegate ()
                    {
                        if
((GraphType.SelectedIndex == 0 && i < j) ||
(GraphType.SelectedIndex == 1))

                            if
(AdjacencyMatrix.Rows[i].Cells[j].Value.ToStri
ng() != "0")
                            {
                                DrawEdge(i, j,
g);
```

```
GraphImage.Image = graphBitmap;
                                edgeWasPainted = true;
                            }
                            else edgeWasPainted = false;
                        else edgeWasPainted = false;
                    });

                    if(slowDown && edgeWasPainted) Thread.Sleep(50);
                }

        }

        void AddRowsAndColumns(int count, int startIndex)
        {
            Invoke((MethodInvoker) delegate ()
            {
                // Добавляем необходимое количество колонок и строк таблицы с начального индекса.
                for (int i = startIndex; i < count; i++)
                {
                    AdjacencyMatrix.Columns.Add("Vertex" + i.ToString(), (i + 1).ToString());
                    AdjacencyMatrix.Columns[i].SortMode = DataGridViewColumnSortMode.NotSortable;
                }

                for (int i = startIndex; i < count; i++)
                {
                    AdjacencyMatrix.Rows.Add();
                    AdjacencyMatrix.Rows[i].HeaderCell.Value = (i + 1).ToString();

                    // Значения ячеек по главной диагонали матрицы смежности всегда равны нулю, их нельзя изменять.
                    AdjacencyMatrix.Rows[i].Cells[i].ReadOnly = true;
                }
            } );
        }

        // Создание таблицы со значениями по умолчанию и построение пустого графа.
        void CreateGraph()
        {
            ClearImage();
            GraphHasColoring = false;
            Cursor = Cursors.WaitCursor;

            drawVertexesThread = new Thread(() => { DrawVertexes(GraphImage.Image, true); });
            drawVertexesThread.Start();

            // Очищаем таблицу.
            AdjacencyMatrix.Rows.Clear();
            AdjacencyMatrix.Columns.Clear();

            int size = Convert.ToInt32(VertexCount.Text);
            if (size != 0)
            {
                // Добаляем строки и столбцы в пустую таблицу.
                AddRowsAndColumns(size, 0);

                fillTableThread = new Thread(() =>
                {
                    // Заполняем ячейки добавленных строк нулями.
                    for (int i = 0; i < size; i++)
                        for (int j = 0; j < size; j++)
                            AdjacencyMatrix.Rows[i].Cells[j].Value = 0.ToString();

                    Invoke((MethodInvoker)delegate { Cursor = Cursors.Arrow; });
                });
                fillTableThread.Start();
            }
        }

        // Изменение таблицы смежности с сохранением всех ранее введённых данных и обновлением рисунка графа.
        void RefreshGraph(int delta)
        {
            ClearImage();

            int prevSize = AdjacencyMatrix.RowCount; // Предыдущий размер таблицы.

            if (delta == -1)
            {
                AdjacencyMatrix.Rows.RemoveAt(prevSize - 1);
                AdjacencyMatrix.Columns.RemoveAt(prevSize - 1);
            }
            else
            {
                // Добавляем одну строку и один стобец в конец таблицы.
                AddRowsAndColumns(1 + prevSize, prevSize);

                // Заполняем ячейки добавленных строк нулями.
                for (int i = 0; i < AdjacencyMatrix.RowCount; i++)
                    AdjacencyMatrix.Rows[i].Cells[prevSize].Value =
                    AdjacencyMatrix.Rows[prevSize].Cells[i].Value = 0.ToString();
            }
            GraphHasColoring = false;
            DrawVertexes(GraphImage.Image);
            DrawEdges();
        }

        int GetColor(int i, int[] vertexesColor)
        {
            // Получение цвета вершины.
            List<int> colorList = new List<int>(); // Список цветов вершин, смежных с рассматриваемой.
            int j;

            FileStream file = new FileStream(Application.StartupPath + "\\Алгориты правильной раскраки\\" + "Правильная раскраска " + (GraphType.SelectedIndex == 0 ? "нео" : "ор") + "графа на " + AdjacencyMatrix.RowCount + " вершин.txt", FileMode.Append, FileAccess.Write);
            StreamWriter streamWriter = new StreamWriter(file);
            try
            {
                // Добавляем в список порядковые номера цветов, в которые окрашены смежные вершины.
                for (j = 0; j < i; j++)
                {
```

```
                    if
(AdjacencyMatrix.Rows[j].Cells[i].Value.ToStri
ng() != "0")
                    {

colorList.Add(vertexesColor[j]);

streamWriter.WriteLine((j + 1).ToString() + "
(цвет " + (vertexesColor[j] + 1) + ") ");
                    }
                    // Для орграфов проверяем
дополнительно симметричную позицию.
                    else if
(GraphType.SelectedIndex == 1 &&
AdjacencyMatrix.Rows[i].Cells[j].Value.ToStrin
g() != "0")
                    {

colorList.Add(vertexesColor[j]);

streamWriter.WriteLine((j + 1).ToString() + "
(цвет " + (vertexesColor[j] + 1) + ") ");
                    }
                }
                streamWriter.Close();
            }
            catch (Exception ex)
            {
                MessageBox.Show("Обнаружена
ошибка при записи шагов алгоритма в файл!\n" +
ex.StackTrace + "\n\nПричина: " + ex.Message,
"",
                        MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
            j = -1;

            // Находимый минимальный
допустимый цвет.
            do
                j++;
            while (colorList.FindIndex(x =>
x.Equals(j)) != -1);
            return j;
        }

        private void
GetGraphColoring_Click(object sender,
EventArgs e)
        {
            vertexesColor = new
int[AdjacencyMatrix.RowCount]; // Массив
цветов для вершин.

            // Инициализируем массив цветов
для вершин.
            for (int i = 0; i <
vertexesColor.Length; i++)
                vertexesColor[i] = -1;

            try
            {
                FileStream file = new
FileStream(Application.StartupPath +
"\\Алгориты правильной раскраки\\" +
"Правильная раскраска " +
(GraphType.SelectedIndex == 0 ? "нео" : "ор")
+ "графа на " + AdjacencyMatrix.RowCount + "
вершин.txt", FileMode.Create,
FileAccess.Write);
                StreamWriter streamWriter =
new StreamWriter(file);

streamWriter.WriteLine("Нахождение правильной
раскраски для " + (GraphType.SelectedIndex ==
0 ? "нео" : "ор") + "графа на " +
AdjacencyMatrix.RowCount + " вершин");
                streamWriter.Close();
                for (int i = 0; i <
AdjacencyMatrix.RowCount; i++)
                {
                    file = new
FileStream(Application.StartupPath +
"\\Алгориты правильной раскраки\\" +
"Правильная раскраска " +
(GraphType.SelectedIndex == 0 ? "нео" : "ор")
+ "графа на " + AdjacencyMatrix.RowCount + "
вершин.txt", FileMode.Append,
FileAccess.Write);
                    streamWriter = new
StreamWriter(file);
                    streamWriter.WriteLine();

streamWriter.WriteLine("Рассматриваем вершину
" + (i + 1) + ":");

streamWriter.WriteLine("Список смежных, уже
раскрашенных вершин: ");
                    streamWriter.Close();

                    // Для каждой вершины
находим цвет.
                    vertexesColor[i] =
GetColor(i, vertexesColor);

                    file = new
FileStream(Application.StartupPath + "\\" +
"Правильная раскраска " +
(GraphType.SelectedIndex == 0 ? "нео" : "ор")
+ "графа на " + AdjacencyMatrix.RowCount + "
вершин.txt", FileMode.Append,
FileAccess.Write);
                    streamWriter = new
StreamWriter(file);

streamWriter.WriteLine("Минимальный допустимый
цвет - " + (vertexesColor[i] + 1));
                    streamWriter.Close();
                }

                GraphHasColoring = true;

DrawVertexes(GraphImage.Image);

                MessageBox.Show("Шаги
алгоритма успешно записаны в файл по
следующему пути:\n" + file.Name, "",
MessageBoxButtons.OK,
MessageBoxIcon.Information);
            }
            catch (Exception ex)
            {
                MessageBox.Show("Обнаружена
ошибка при записи шагов алгоритма в файл!\n" +
ex.StackTrace + "\n\nПричина: " + ex.Message,
"",
                    MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }

        private void cellText_KeyPress(object
sender, KeyPressEventArgs e)
        {
            if
(AdjacencyMatrix.CurrentCell.RowIndex !=
AdjacencyMatrix.CurrentCell.ColumnIndex)
            {
                int rowIndex =
AdjacencyMatrix.CurrentCell.RowIndex;
                int columnIndex =
AdjacencyMatrix.CurrentCell.ColumnIndex;

                // Если нажаты не ноль, не
единица, значение не меняем.
                if (e.KeyChar == '1' ||
e.KeyChar == '0')
                {
                    (sender as
TextBox).SelectionStart = 0;
                    (sender as
TextBox).SelectionLength = 1;

AdjacencyMatrix.Rows[rowIndex].Cells[columnInd
ex].Value = e.KeyChar;

                    if
(GraphType.SelectedIndex == 0) // Для
неорграфов матрица смежности симметрична.
```

```
AdjacencyMatrix.Rows[columnIndex].Cells[rowInd
ex].Value = e.KeyChar;

                    GraphHasColoring = false;

                    if (e.KeyChar == '1') //
Если нажата единица, рисуем ребро (ребро).
                    {

DrawVertexes(GraphImage.Image);
                        Bitmap graphBitmap =
new Bitmap(GraphImage.Image);
                        Graphics g =
Graphics.FromImage(graphBitmap);
                        DrawEdge(rowIndex,
columnIndex, g);
                        GraphImage.Image =
graphBitmap;
                        GraphImage.Size =
graphBitmap.Size;
                    }
                    else
                    {
                        ClearImage();

DrawVertexes(GraphImage.Image);
                        DrawEdges();
                    }
                }
                else e.Handled = true;
            }
        }

        private void
AdjacencyMatrix_EditingControlShowing(object
sender,
DataGridViewEditingControlShowingEventArgs e)
        {
            TextBox cellText = e.Control as
TextBox;
            cellText.MaxLength = 1;
            cellText.KeyPress -=
cellText_KeyPress;
            cellText.KeyPress += new
KeyPressEventHandler(cellText_KeyPress);
        }

        private void
statusStrip_MouseHover(object sender,
EventArgs e)
        {
            ColorGrid.Show();
        }

        private void
ColorGrid_MouseLeave(object sender, EventArgs
e)
        {
            if(!IsColorDialogOpen)
                ColorGrid.Hide();
        }

        private void LoadFromFile_Click(object
sender, EventArgs e)
        {
            Stream myStream = null;
            openFileDialog.InitialDirectory =
Application.StartupPath + "\\Графы";
            openFileDialog.Filter = "txt files
(*.txt)|*.txt|All files (*.*)|*.*";
            openFileDialog.FilterIndex = 1;

            if (openFileDialog.ShowDialog() ==
DialogResult.OK)
            {
                if ((myStream =
openFileDialog.OpenFile()) != null)
                {
                    try
                    {
                        using (myStream) //
Блок using гарантирует освобождение потока,
связанный с чтением файла.
                        {
                            StreamReader
streamReader = new StreamReader(myStream);
                            IsLoadingFromFile
= true;

                            string Line; //
Строка текстового файла.
                            char separator = '
'; // Разедлитель значений матрицы смежности.
                            int matrixSize =
0; // Размер матрицы (количество вершин в
графе).
                            string[]
cellValues; // Массив ячеек строки таблицы.
                            int lineCount = 0;
// Порядковый номер считываемой строки.

                            if
(streamReader.EndOfStream != true)
                            {
                                matrixSize =
Convert.ToInt16(streamReader.ReadLine());


VertexCount.Text = matrixSize.ToString();

                                // Очищаем
таблицу.

AdjacencyMatrix.Rows.Clear();

AdjacencyMatrix.Columns.Clear();

                                // Создаём
новую таблицу.

AddRowsAndColumns(matrixSize, 0);
                            }

                            int[][] matrix =
new int[matrixSize][]; // Создаём временную
матрицу для занесения значений из файла.


                            while
(streamReader.EndOfStream != true)
                            {
                                Line =
streamReader.ReadLine(); // Считываем строку
матрицы.
                                cellValues =
Line.Split(separator); // Разбиваем на ячейки.

                                if (matrixSize
== cellValues.Length && lineCount <
matrixSize)
                                {

matrix[lineCount] = new int[matrixSize];
                                    for (int i
= 0; i < cellValues.Length; i++)
                                    {
                                        if
(cellValues[i] != "0" && cellValues[i] != "1")

throw new Exception("Элементы матрицы
смежности должны быть равны либо 0, либо 1 (и
-1 для орграфов)!");

                                        if (i
== lineCount && cellValues[i] != "0")

throw new Exception("Элементы матрицы
смежности по главной диагонали должны быть
равны нулю!");


matrix[lineCount][i] =
Convert.ToInt16(cellValues[i]);
                                    }
                                }
                                else throw new
Exception("Неверно указан размер матрицы!");

                                lineCount++;
                            }
```

```
streamReader.Close();

                            // Определяем вид
графа в зависимости от симметричности матрицы
смежности.
                            int selectedType =
0;
                            for (int i = 0; i
< matrixSize && selectedType == 0; i++)
                                for (int j =
0; j < matrixSize && selectedType == 0; j++)
                                    if
(matrix[i][j] != matrix[j][i])

selectedType = 1;


GraphType.SelectedIndex = selectedType;

                            GraphHasColoring =
false;

                            Cursor =
Cursors.WaitCursor;
                            ClearImage();

                            drawVertexesThread
= new Thread(() => {
DrawVertexes(GraphImage.Image, true); });

drawVertexesThread.Start();

                            fillTableThread =
new Thread(() =>
                                {
                                    //
Заполняем ячейки добавленных строк нулями.
                                    for (int i
= 0; i < matrixSize; i++)
                                        for
(int j = 0; j < matrixSize; j++)

AdjacencyMatrix.Rows[i].Cells[j].Value =
matrix[i][j].ToString();
                                    while
(drawVertexesThread.IsAlive) { }

DrawEdges(true);

Invoke((MethodInvoker)delegate () { Cursor =
Cursors.Arrow; });
                                });

fillTableThread.Start();

                            IsLoadingFromFile
= false;
                        }
                    }
                    catch (Exception ex)
                    {
                        if
(drawVertexesThread.IsAlive)
drawVertexesThread.Abort();
                        if
(fillTableThread.IsAlive)
fillTableThread.Abort();
                        ClearImage();

AdjacencyMatrix.Rows.Clear();

AdjacencyMatrix.Columns.Clear();

MessageBox.Show("Невозможно считать данные
матрицы смежности из файла!\nПричина: " +
                            ex.Message, "",
MessageBoxButtons.OK, MessageBoxIcon.Error);

                        VertexCount.Text =
0.ToString();
                    }
                }
            }
        }

        private void SaveToFile_Click(object
sender, EventArgs e)
        {
            Stream myStream;
            saveFileDialog.InitialDirectory =
Application.StartupPath + "\\Графы";
            saveFileDialog.Filter = "txt files
(*.txt)|*.txt|All files (*.*)|*.*";
            saveFileDialog.FilterIndex = 1;
            saveFileDialog.FileName =
(GraphType.SelectedIndex == 0 ? "Нео" : "Ор")
+ "граф на " + AdjacencyMatrix.RowCount + "
вершин.txt";

            if (saveFileDialog.ShowDialog() ==
DialogResult.OK)
            {
                if ((myStream =
saveFileDialog.OpenFile()) != null)
                {
                    try
                    {
                        using (myStream) //
Блок using гарантирует освобождение потока,
связанный с чтением файла.
                        {
                            StreamWriter
streamWriter = new StreamWriter(myStream);


streamWriter.WriteLine(AdjacencyMatrix.RowCoun
t.ToString());
                            for (int i = 0; i
< AdjacencyMatrix.RowCount; i++)
                            {
                                for (int j =
0; j < AdjacencyMatrix.ColumnCount; j++)
                                {

streamWriter.Write(AdjacencyMatrix.Rows[i].Cel
ls[j].Value.ToString());

streamWriter.Write(j ==
AdjacencyMatrix.ColumnCount - 1 ? "" : " ");
                                }

streamWriter.WriteLine();
                            }

streamWriter.Close();
                        }
                    }
                    catch (Exception ex)
                    {

MessageBox.Show("Ошибка при записи данных в
файл!\nПричина: " +
                            ex.Message, "",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                    }
                }
            }
        }

        private void
layoutPanel_SizeChanged(object sender,
EventArgs e)
        {
            ClearImage();
            DrawVertexes(GraphImage.Image);
            DrawEdges();
            GraphImage.Refresh();
        }

        void TransformToUndirectedGraph()
        {
            for (int i = 0; i <
AdjacencyMatrix.RowCount; i++)
                for (int j = i + 1; j <
AdjacencyMatrix.ColumnCount; j++)
                    if
(AdjacencyMatrix.Rows[i].Cells[j].Value.ToStri
ng() !=
```

```
AdjacencyMatrix.Rows[j].Cells[i].Value.ToStrin
g())

AdjacencyMatrix.Rows[i].Cells[j].Value =
AdjacencyMatrix.Rows[j].Cells[i].Value = "1";
        }
        private void
GraphType_SelectedIndexChanged(object sender,
EventArgs e)
        {
            if (!IsLoadingFromFile)
            {
                if (GraphType.SelectedIndex ==
0)

TransformToUndirectedGraph();

                ClearImage();
                GraphHasColoring = false;

DrawVertexes(GraphImage.Image);
                DrawEdges();
            }
        }

        private void
VertexCount_KeyDown(object sender,
KeyEventArgs e)
        {
            if (e.KeyData == Keys.Up &&
VertexCount.Text != "75")
                AddVertex_Click(sender, e);
            else if (e.KeyData == Keys.Down &&
VertexCount.Text != "0")
                RemoveVertex_Click(sender, e);
        }

        private void
VertexCount_KeyPress(object sender,
KeyPressEventArgs e)
        {
            if (e.KeyChar == 13 &&
VertexCount.Text != "")
            {
                CreateGraph();
            }
            else if ((e.KeyChar < 46 ||
e.KeyChar > 57) && e.KeyChar != 8)
                e.Handled = true;
            else if (e.KeyChar != 8 &&
Convert.ToInt16(VertexCount.Text + e.KeyChar)
> 75)
                e.Handled = true;
            else
            {
                if (GetGraphColoring.Enabled
== true) GetGraphColoring.Enabled = false;
                if (SaveToFile.Enabled ==
true) SaveToFile.Enabled = false;
            }
        }

        private void AddVertex_Click(object
sender, EventArgs e)
        {
            if
(Convert.ToInt16(VertexCount.Text) + 1 <= 75)
            {
                VertexCount.Text =
(Convert.ToInt16(VertexCount.Text) +
1).ToString();
                if (GetGraphColoring.Enabled
== false) GetGraphColoring.Enabled = true;
                if (SaveToFile.Enabled ==
false) SaveToFile.Enabled = true;
                RefreshGraph(1);
            }
        }

        private void RemoveVertex_Click(object
sender, EventArgs e)
        {
            if
(Convert.ToInt16(VertexCount.Text) - 1 >= 0)
            {
                VertexCount.Text =
(Convert.ToInt16(VertexCount.Text) -
1).ToString();
                if (VertexCount.Text == "0")
                {
                    GetGraphColoring.Enabled =
false;
                    SaveToFile.Enabled =
false;
                }
                RefreshGraph(-1);
            }
        }

        private void Create_Click(object
sender, EventArgs e)
        {
            CreateGraph();
            if (VertexCount.Text == "" ||
VertexCount.Text == "0")
            {
                GetGraphColoring.Enabled =
false;
                SaveToFile.Enabled = false;
            }
            else
            {
                GetGraphColoring.Enabled =
true;
                SaveToFile.Enabled = true;
            }
        }

        private void Form1_FormClosing(object
sender, FormClosingEventArgs e)
        {
            if (drawVertexesThread != null)
drawVertexesThread.Abort();
            if (fillTableThread != null)
fillTableThread.Abort();
        }

        private void ChangeCursors(Cursor
cursor, bool enabled)
        {
            MaximizeBox = enabled;
            MinimizeBox = enabled;
            VertexCount.Enabled = enabled;
            AddVertex.Enabled = enabled;
            RemoveVertex.Enabled = enabled;
            Create.Enabled = enabled;
            GraphType.Enabled = enabled;
            SaveToFile.Enabled = enabled;
            LoadFromFile.Enabled = enabled;
            GetGraphColoring.Enabled =
enabled;
        }
        private void
Form1_CursorChanged(object sender, EventArgs
e)
        {
            // При изменении курсора окна
формы изменяем курсора
            // каждого элемента формы и
изменяем доступность кнопок.

            if (Cursor == Cursors.WaitCursor)

ChangeCursors(Cursors.WaitCursor, false);
            else ChangeCursors(Cursors.Hand,
true);

        }
    }
}
```

ПРИЛОЖЕНИЕ В
ЭКРАННЫЕ ФОРМЫ

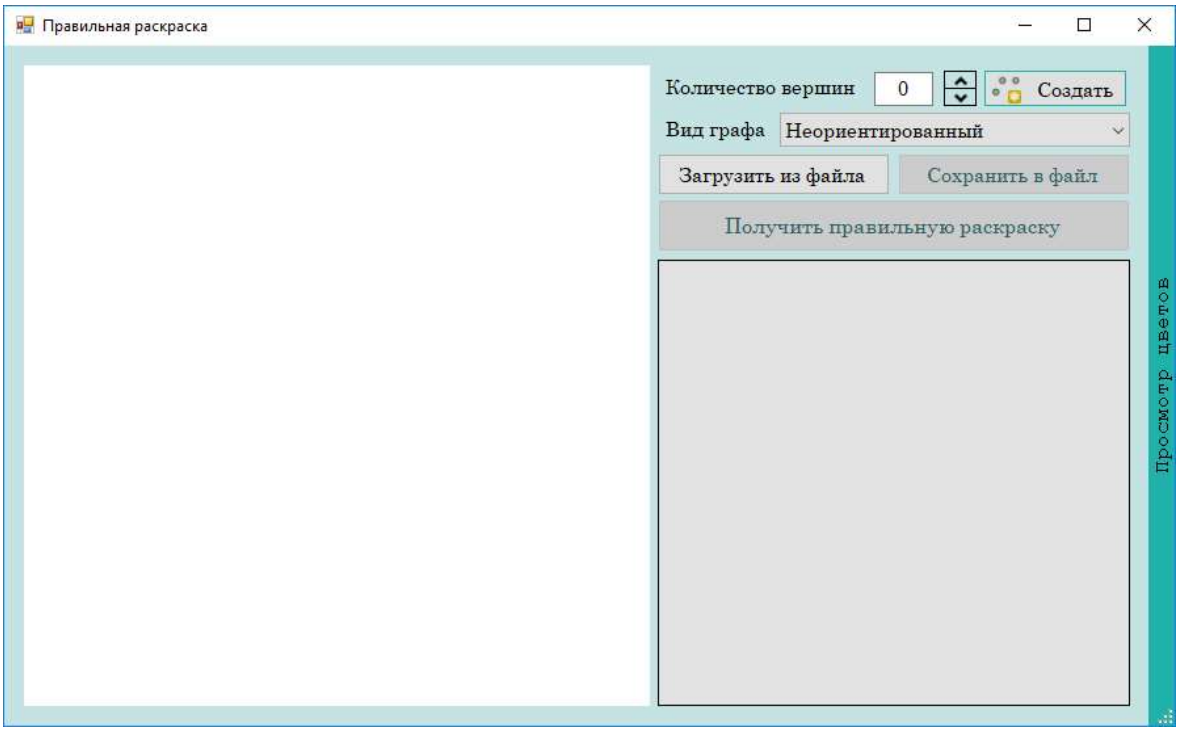

Рисунок В.1 – Главное окна программы

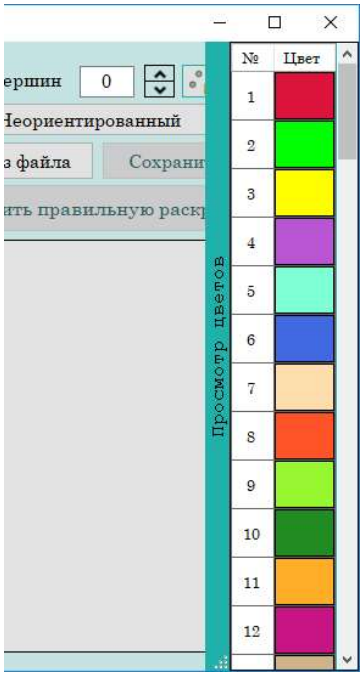

Рисунок В.2 – Пример списка цветов по умолчанию

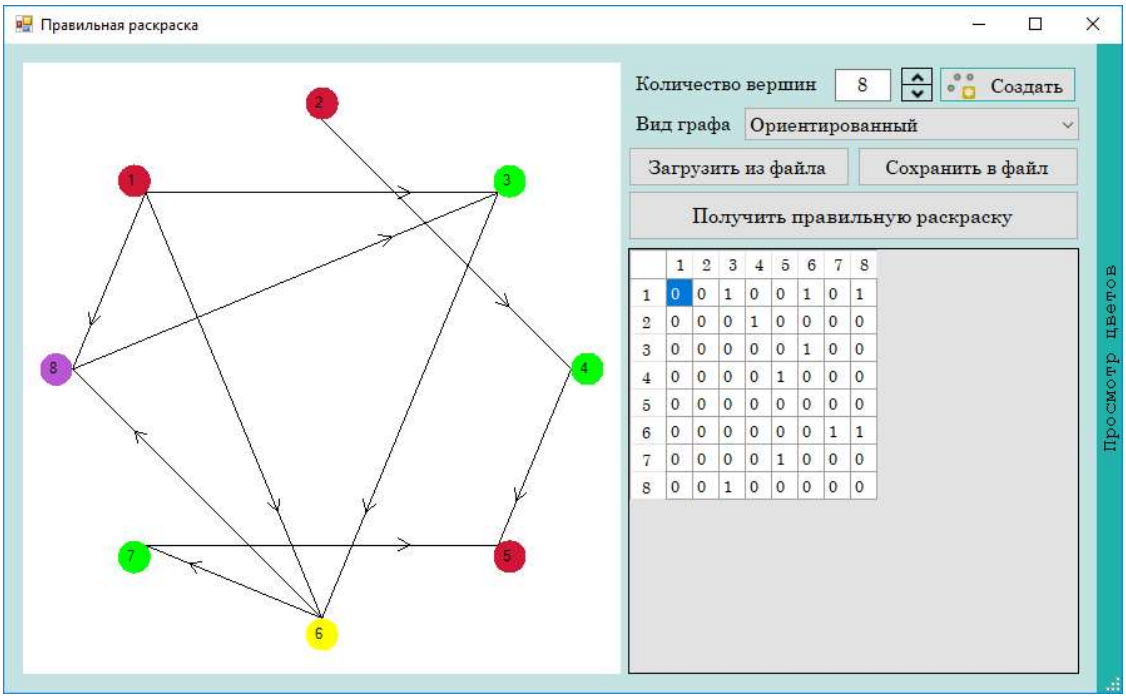

Рисунок В.3 – Результат работы алгоритма для графа

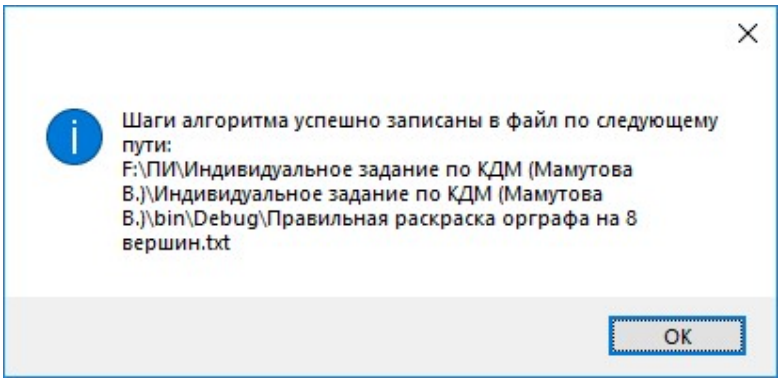

Рисунок В.3 – Сообщение об успешной записи шагов алгоритма в файл, отображающееся после нажатия кнопки «Получить правильную раскраску»

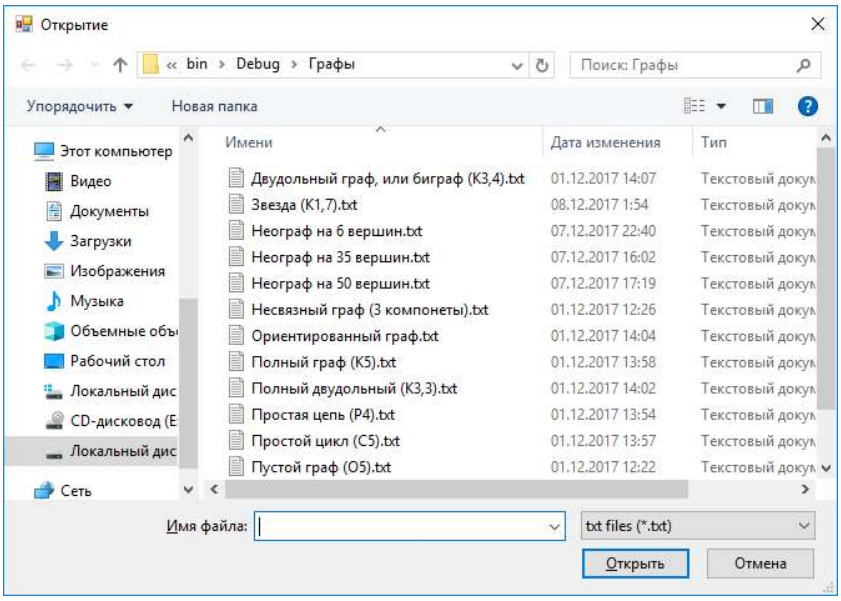

Рисунок В.4 – Диалоговое окно для загрузки матрицы из файла

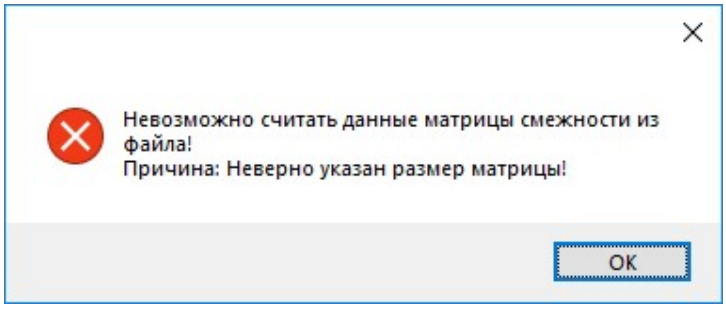

Рисунок В.5 – Сообщение 1 об ошибке в исходном файле матрицы

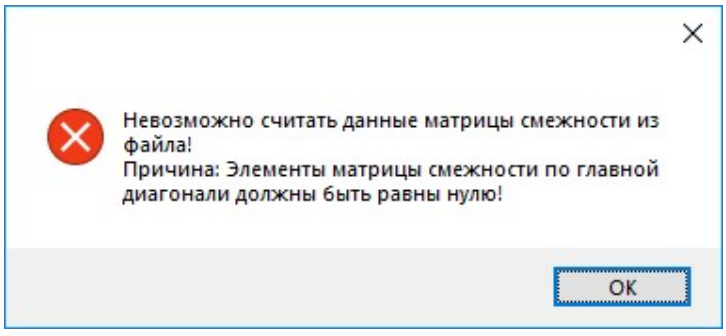

Рисунок В.6 – Сообщение 2 об ошибке в исходном файле матрицы

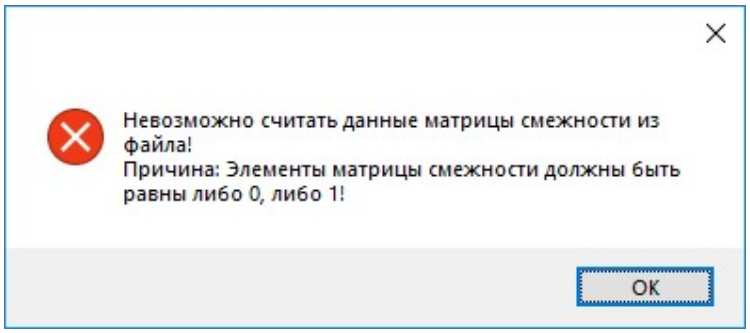

Рисунок В.7 – Сообщение 3 об ошибке в исходном файле матрицы

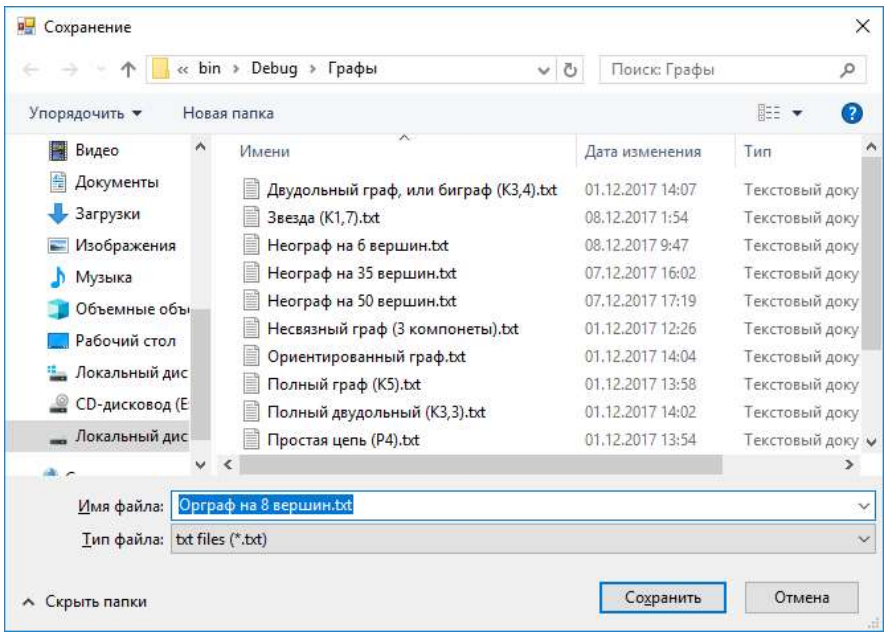

Рисунок В.8 – Диалоговое окно для сохранения матрицы в файл с автоматически предлагаемым именем файла