Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

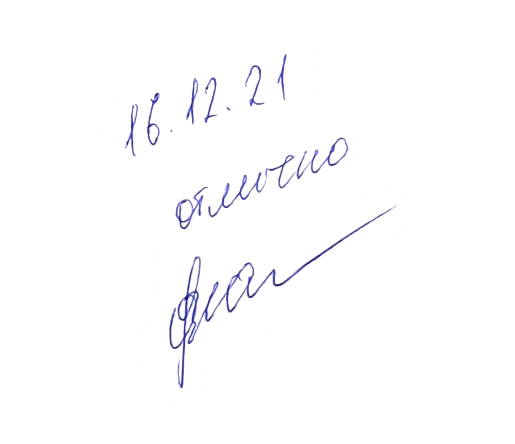
Пенза 2021

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проектированию

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Реализация алгоритма раскрашивания графов»

Выполнила:

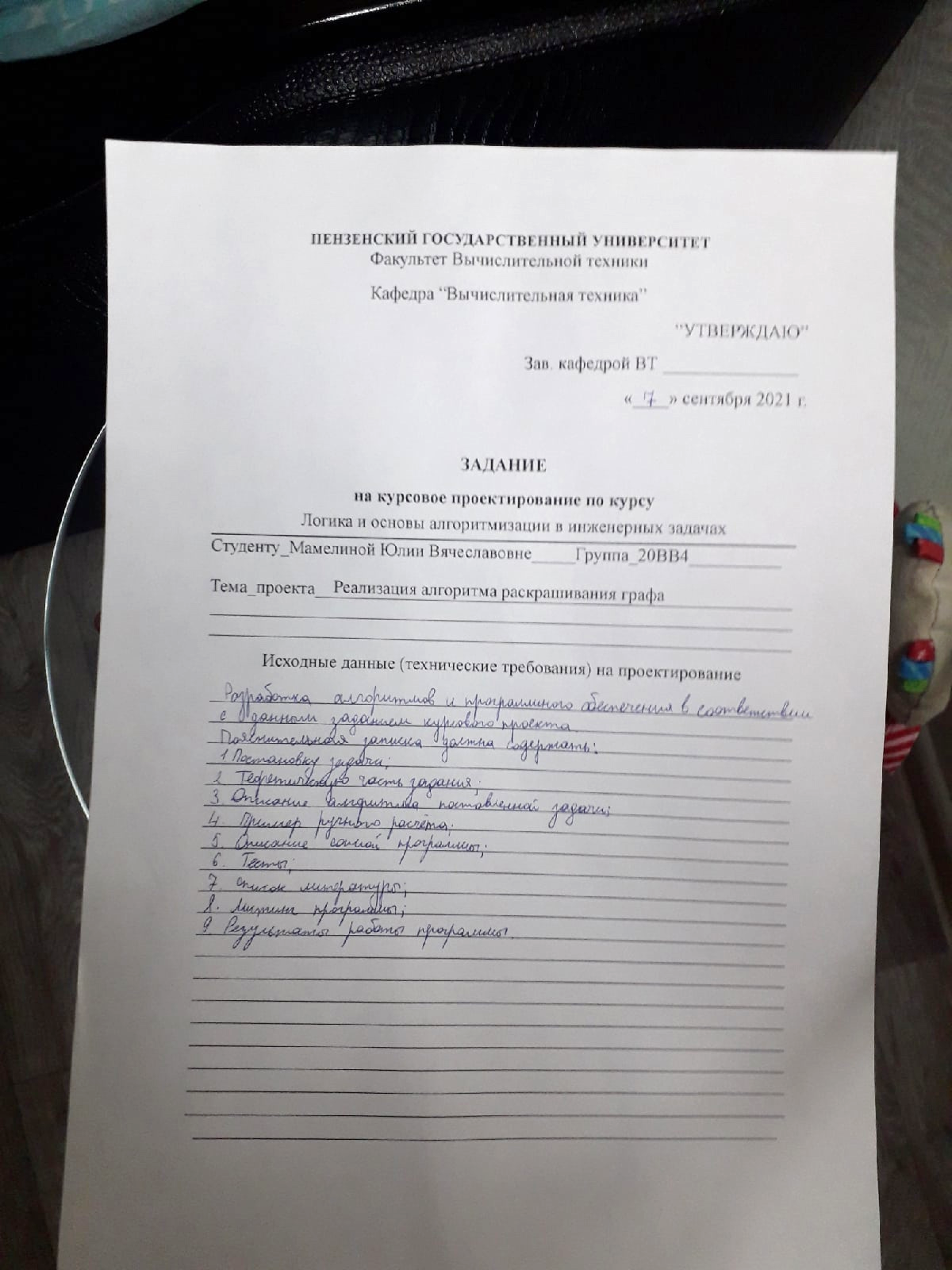
студентка группы 20ВВ4

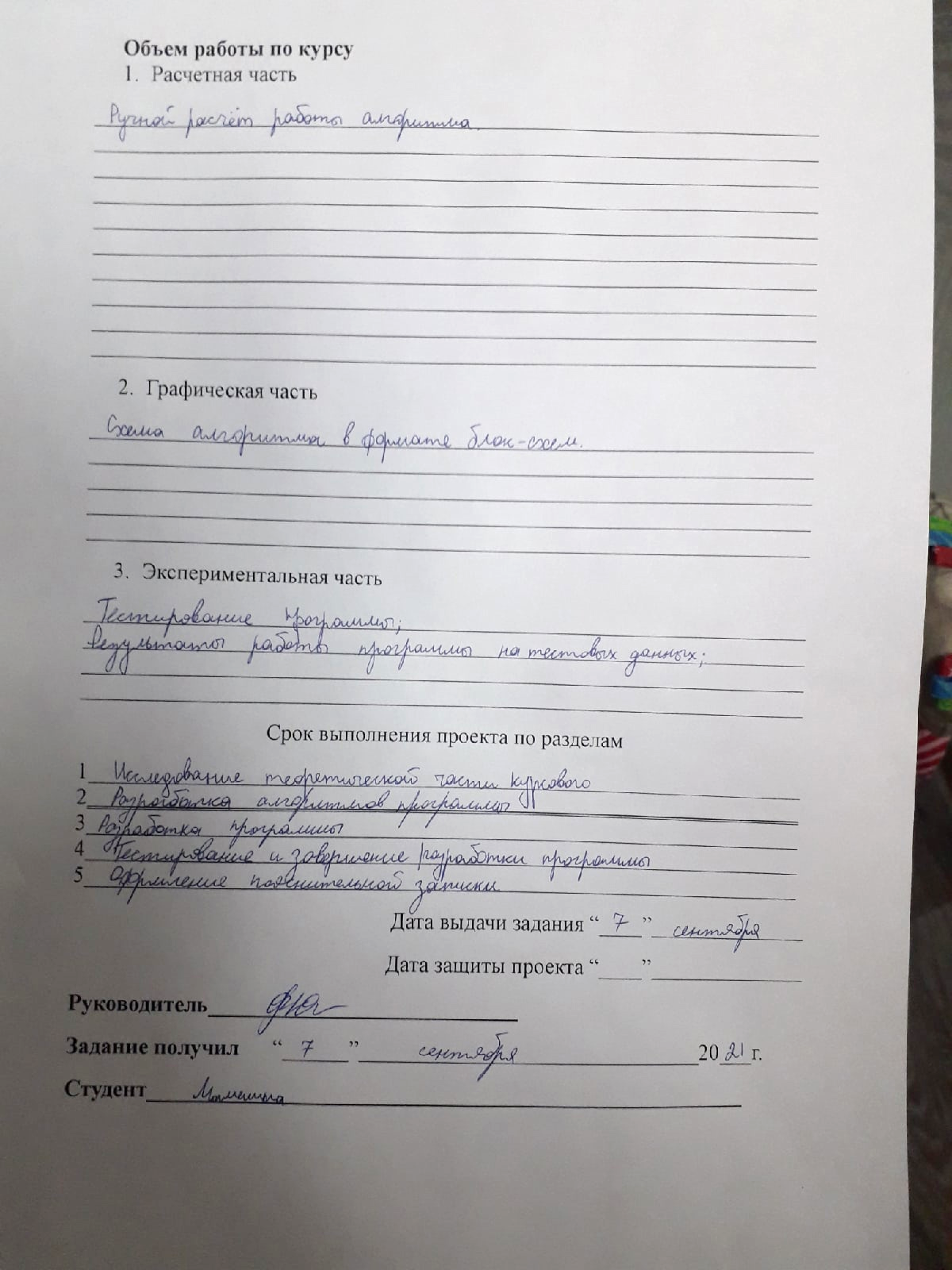
Мамелина Ю. В.

Приняла:

Юрова О. В.

Пенза 2021





**Содержание**

Реферат 5

Введение 6

Постановка задачи 7

1. Теоретическая часть 8

1.1. Некоторые теоремы, относящиеся к хроматическим числам 8

1.2. Точные алгоритмы раскраски 8

2. Описание алгоритма программы 11

3. Описание программы 13

4. Тестирование 16

5. Ручной расчёт задачи 21

6. Заключение 24

Список литературы 25

Приложение А. Листинг программы. 26

Приложение Б. Блок – схема алгоритма 33

# Реферат

ОТЧЕТ 35 СТР., 17 РИСУНКОВ.

ГРАФ, ТЕОРИЯ ГРАФОВ, РАСКРАСКА, ХРОМАТИЧЕСКОЕ ЧИСЛО, ГИПОТЕЗА ЧЕТЫРЕХ КРАСОК, ТЕОРЕМА О ПЯТИ КРАСКАХ, ТЕОРЕМА О ЧЕТЫРЕХ КРАСКАХ

Цель исследования: создание программы для решения одной из задач теории графов – построение правильной раскраски.

В работе рассмотрены алгоритмы нахождения хроматического числа графа (построения раскраски) графа. Приведено описание точных и приближенных алгоритмов. Установлено, что, когда размеры графа слишком велики, получение оптимальной раскраски точными методами, затруднительно. Поэтому в работе использован последовательный алгоритм, основанный на упорядочивании множества вершин.

# Введение

Некоторые задачи, возникающие при планировании производства, составлении графиков осмотра и транспортировки товаров, могут быть представлены как задачи теории графов, тесно связанные с так называемой «задачей раскраски», которая формулируется следующим образом: для данного графа определить минимальное количество цветов, необходимых для раскраски вершин графа так, чтобы никакие две смежные вершины не были окрашены в один цвет.

Граф  называют *r*-хроматическим, если его вершины могут быть раскрашены с использованием *r* цветов (красок) так, что не найдется двух смежных вершин одного цвета. Наименьшее число *r*, такое, что граф *G* является *r*-хроматическим, называется хроматическим числом графа *G* и обозначается . Задача нахождения хроматического числа графа называется задачей о раскраске (или задачей раскраски) графа. Соответствующая этому числу раскраска вершин разбивает множество вершин графа на *r* подмножеств, каждое из которых содержит вершины одного цвета. Эти множества являются независимыми, поскольку в пределах одного множества нет двух смежных вершин.

Существует много алгоритмов раскрашивания графов, позволяющих находить хорошие приближения для хроматического числа графа в тех случаях, когда размеры графа слишком велики и получение оптимальной раскраски точными методами, затруднительно. В работе рассмотрен последовательный алгоритм, основанный на упорядочивании множества вершин.

В качестве среды разработки мною была выбрана среда Microsoft Visual Studio 2019, язык программирования С++.

Целью данной курсовой работы по дисциплине «Логика и основы алгоритмизации в инженерных задачах» является создание программы для решения одной из задач теории графов – построение правильной раскраски.

# ПОСТАНОВКА ЗАДАЧИ

Задан граф *G*(*V*, *E*), где *V* - множество вершин; *E* - множество ребер. Раскрасить вершины графа красками так, чтобы ни одно его ребро не имело соцветных концов. Предусмотреть графическое представление исходного графа и цветовое выделение его вершин на экране.

Исходный граф в программе должен задаваться списком ребер. Пользователь задает количество вершин и ребер и список ребер (ребро задается двумя вершинами). После обработки этих данных на экран должен выводиться список вершин с указанием цвета каждой. Устройство ввода - клавиатура или файл.

# Теоретическая часть

Разнообразные задачи, возникающие при планировании производства, составлении графиков осмотра, хранении и транспортировке товаров и т. д., могут быть представлены часто как задачи теории графов, тесно связанные с так называемой «задачей раскраски». Графы, рассматриваемые далее, являются неориентированными и не имеют петель; если специально не оговаривается иное, то под словом «граф» понимается именно такой граф.

Задача нахождения хроматического числа произвольного графа явилась предметом многих исследований в конце XIX и в текущем столетии. Сейчас по этому вопросу известно большое количество интересных результатов. Мы вводим только такие понятия, которые нужны для построения алгоритмов решения задачи о раскраске графа. Здесь мы рассматриваем в основном алгоритмы, позволяющие находить значение хроматического числа произвольного графа и соответствующую этому значению раскраску вершин.

## Некоторые теоремы, относящиеся к хроматическим числам

*Гипотеза четырех красок*

Граф, который можно так изобразить на плоскости, что никакие два его ребра не пересекаются между собой[[1]](#footnote-1), называется планарным. Планарные графы важны как с теоретической, так и с практической точек зрения и обладают рядом таких свойств, связанных с раскраской, о которых следует упомянуть.

*Теорема о пяти красках*

Каждый планарный граф можно так раскрасить, используя пять цветов, что любые две смежные вершины будут окрашены в разные цвета, т.е. если G – планарный граф, то γ(G) ≤ 5.

*Теорема о четырех красках (недоказанная).*

Каждый планарный граф можно так раскрасить, используя 4 цвета, что любые две смежные вершины будут окрашены в разные цвета, т. е. γ(G) ≤ 4, если G – планарный граф.

## Точные алгоритмы раскраски

В этом разделе рассматриваются некоторые алгоритмы, которые являются точными в том смысле, что они гарантируют нахождение оптимальной раскраски и истинного значения хроматического числа для любого графа.

*Метод динамического программирования*

Максимальные *r*–подграфы. Порожденный подграф графа *G = (V, E)* называется *r* – подграфом, если он *r*–хроматический. Если не существует такого множества, что оно принадлежит порождённому подграфу и подграф является *r*–хроматическим, то подграф называется максимальным *r*–подграфом графа G. Очевидно, что для фиксированного значения *r*, обще говоря, существует много различных максимальных *r*–подграфов данного графа G. Подграф, у которого множество вершин совпадает с некоторым максимальным независимым множеством в G и который не имеет ребер (т. е. является вполне несвязным), есть максимальный 1–подграф графа G, поскольку в G нет 1–хроматического подграфа, имеющего большее множество вершин, чем у рассматриваемого подграфа.

Хроматическое число графа G можно определить как такое наименьшее число *r*, что подграф по крайней мере для одного из максимальных *r*–подграфов графа G.

*Теорема 1.* Если граф G является *r*–хроматическим, то он может быть раскрашен с использованием *r* (или меньшего числа) красок с помощью следующей процедуры: сначала в один цвет окрашивается некоторое максимальное независимое множество, затем окрашивается в следующий цвет множество имеющее с ним смежность и т. д. до тех пор, пока не будут раскрашены все вершины.

Раскраску указанного в теореме вида будем называть оптимальной независимой раскраской.

Рекуррентные соотношения. Вышеприведенную теорему можно использовать для получения рекуррентного соотношения, связывающего максимальные *r*–подграфы графа с его максимальными (*r* – 1)–подграфами.

Пусть есть некое семейство максимальных (*r* – 1)–подграфов графа G, а его множество вершин *n*–го подграфа принадлежат данному семейству. Множество подграфа является тогда множеством вершин *k*–го максимального 1–подграфа (независимого множества) графа, образованного вершинами графа G, не попавшими в (*r* – 1)–подграф.

Алгоритм, основанный на рассмотрении максимальных r–подграфов. В предыдущем разделе было указано, что хроматическое число графа G является таким наименьшим значением r, при котором множество вершин некоторого максимального *r*–подграфа – совпадает с множеством вершин графа G. Поэтому рекуррентные соотношения и могут быть использованы для последовательного построения максимальных 1–подграфов, 2–подграфов и т. д. и нахождения хроматического числа графа G, нужно просто на каждом шаге указанной процедуры проверять, не содержится ли множество вершин графа G в каком–нибудь из построенных подграфов.

Ниже приводится описание алгоритма, позволяющего находить хроматическое число графа G и раскраску, соответствующую этому числу.

Шаг 1. Упорядочить вершины по не возрастанию степени;

Шаг 2. Окрасить первую вершину в цвет *r*=1;

Шаг 3. Выбрать цвет окраски *r*=1;

Шаг 4. Пока не окрашены все вершины, повторять 4.1. – 4.2.:

4.1. Окрасить в выбранный цвет всякую вершину, которая не смежна

с другой, уже окрашенной в этот цвет;

4.2. Выбрать цвет окраски *r*+1.

Следует отметить, однако, что описанный алгоритм не дает полного перечисления всех возможных раскрасок в *r* + 1 цветов, а только порождает оптимальные независимые раскраски. Такие раскраски могут оказаться только небольшой частью всех возможных раскрасок в *r* + 1 цветов. [[2]](#footnote-2)

Формулировка задачи о раскраске на языке 0–1–программирования

Пусть [Gij] – матрица раскраски графа (задающая некоторую конкретную раскраску вершин графа), так что

# Описание алгоритма программы

Алгоритм раскраски графа

Раскрасить граф можно используя следующий алгоритм:

1. Найти число связей всех вершин графа.

2. Рассматривать вершины в порядке не возрастания числа связей.

3. Начинаем раскрашивание в цвет №1. Рассматриваем вершины последовательно и если рассматриваемая вершина не раскрашена, а также не имеет связей с вершинами, раскрашенными в этот цвет, то раскрашиваем её в этот цвет.

4. Если все вершины просмотрены, но не раскрашены, то повторяем пункт 3, с цветом на единицу больше, пока все вершины не раскрашены. Число цветов это и есть хроматическое число.

Для программной реализации алгоритма необходимо описать следующие переменные:

A - матрица смежности графа, двумерный массив (int);

n - количество вершин графа (int);

m - количество ребер (int);

colors - массив названий используемых системных цветов (string);

Colors - массив номеров цветов вершин (int);

FIFO - структура данных очередь (int);

Set - массив цветов, в которые уже окрашены вершины (int);

cr - количество цветов для раскраски (int);

Start - номер начальной вершины (int);

k - указатель на конец очереди (int);

p - указатель на начало очереди (int);

cur - номер очередной вершины (int);

pos - счетчик цветов, в которые окрашены вершины (int).

В начале выполняется чтение размерности графа и помещение числа вершин и ребер в переменные n и m соответственно. Массивам A, Colors, FIFO и Set выделяется память при помощи функции new, массивы обнуляются. Далее из таблицы ребер считываются значения и формируется матрица смежности А.

Выбираем начальную вершину Start = 0 и раскрашиваем ее в цвет 1 Colors[Start] = 1. Инициализируем указатели на начало и конец очереди p = 0, k = 1. Помещаем в очередь начальную вершину FIFO[p] = Start.

Далее в цикле с предусловием: пока p ≠ k (пока очередь не станет пустой) выполняем раскраску.

Извлекаем из очереди текущую вершину cur = FIFO[p] и увеличиваем указатель на начало очереди p на 1.

В цикле просматриваем строку cur матрицы смежности А. Если ребро A[cur][i] существует и i-тая вершина Colors[i] не окрашена, выполняем:

помещаем i-тую вершину в очередь FIFO[k] = i и увеличиваем указатель на конец очереди k на 1.

обнуляем счетчик цветов pos.

извлекаем из очереди FIFO[k - 1]-й элемент и в цикле по j просматриваем соответствующую ему строку матрицы смежности. Если A[FIFO[k - 1]][j] существует и вершина j окрашена Colors[j] ≠ 0, формируем массив цветов, в которые уже окрашены вершины Set[pos] = Colors[j], увеличиваем счетчик pos на 1.

выполняем поиск минимального номера цвета, в который можно окрасить вершину Colors[i] = Find(pos, Set).

После завершения цикла с предусловием, все вершины будут окрашены.

В массиве Colors находятся номера цветов соответствующих вершин. Максимальный из них cr = max(Colors) и будет хроматическим числом графа.

Выводим решение в виде:

номер вершины - название цвета

из массива colors в соответствии с номером массива Colors[i - 1].

Рассмотрим функцию

int Find(int pos, int \*Set), которая выполняет поиск минимального номера цвета для раскраски вершины.

В качестве параметров в функция принимает pos - количество окрашенных вершин и Set - массив номеров цветов, в которые уже окрашены вершины.

Минимальному цвету присваиваем начальное значение Min\_color = 0.

Далее в цикле среди элементов массива Set находим цвет с максимальным номером.

Передаем функции значение Min\_color.

Полный код программы приведен в Приложении А.

Полный код программы приведен в Приложении Б.

# Описание программы

Для написания данной программы использован язык программирования С++ и среда Visual Studio.

С++ – это язык программирования общего назначения, хорошо известный своей эффективностью, экономичностью, и переносимостью. Во многих случаях программы, написанные на С++, сравнимы по скорости с программами, написанными на языке ассемблера. При этом они имеют лучшую наглядность, и их более просто сопровождать.

Проект был создан в виде приложения Windows Forms.

Рассмотрим состав функций программы:

String^ sToS(string t) - преобразование строки типа string в System::String.

string STos(System::String^ t) - преобразование строки System::String в string.

void calcCoordinats(double \*xv, double \*yv, int n) - формирует массив координат вершин, расположенных по двум концентрическим окружностям.

На входе: n - количество вершин.

Возвращает:

xv, yv - массивы координат вершин.

void createCharts(void) - создает семейство графиков для отображения ребер и вершин графа.

int Find(int pos, int \*Set) - поиск минимального номера цвета в который можно окрасить вершину.

На входе:

pos - количество окрашенных вершин.

Set - массив цветов в которые уже окрашены вершины.

Возвращает: номер минимального цвета, в который можно окрасить вершину.

Далее приведены событийные функции, обрабатывающие элементы управления программы.

edgeChange\_ValueChanged - изменение числа ребер графа.

vertexChange\_ValueChanged - изменение количества вершин.

menuExit\_Click - выход из программы.

randomGraph\_Click - генератор псевдослучайного взвешенного графа.

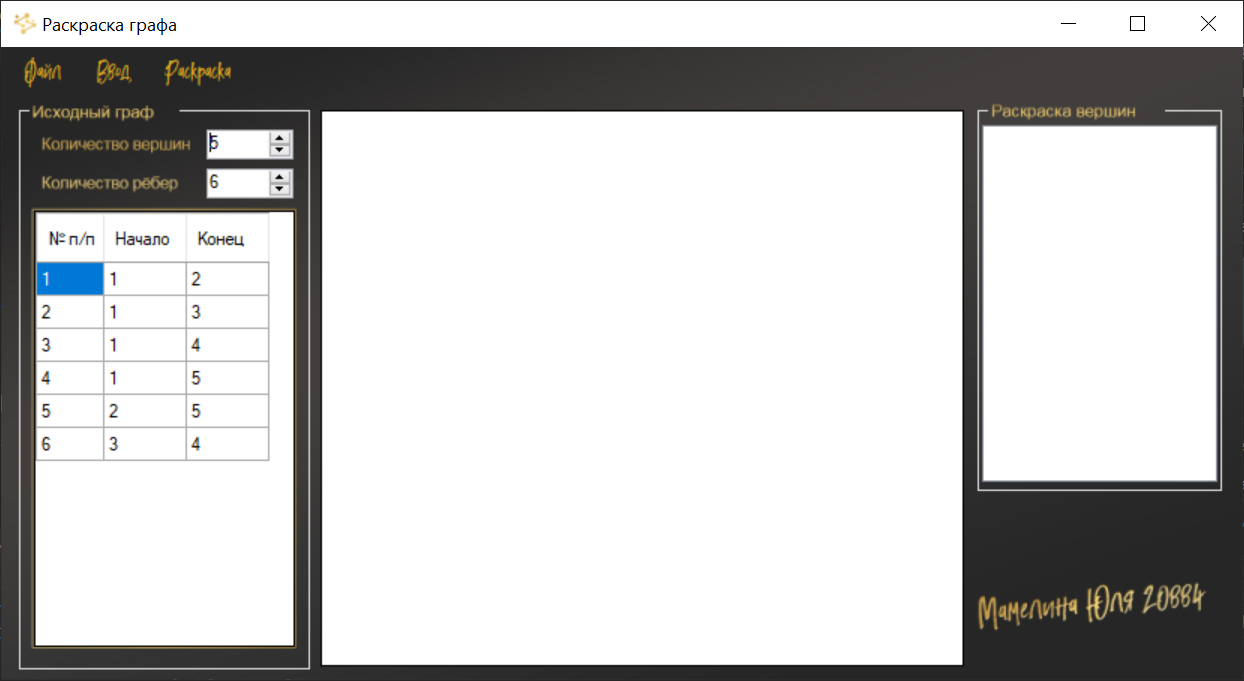
menuColored\_Click - выполняет раскраску графа и вывод результата в графическом и текстовом виде.

saveGraph\_Click - сохранить граф в файле.

loadGraph\_Click - загрузить граф из файла.

menuSaveResult\_Click - сохранить результат раскраски в файле.

После запуска программы на экране монитора компьютера появляется главное окно программы (рис. 1).

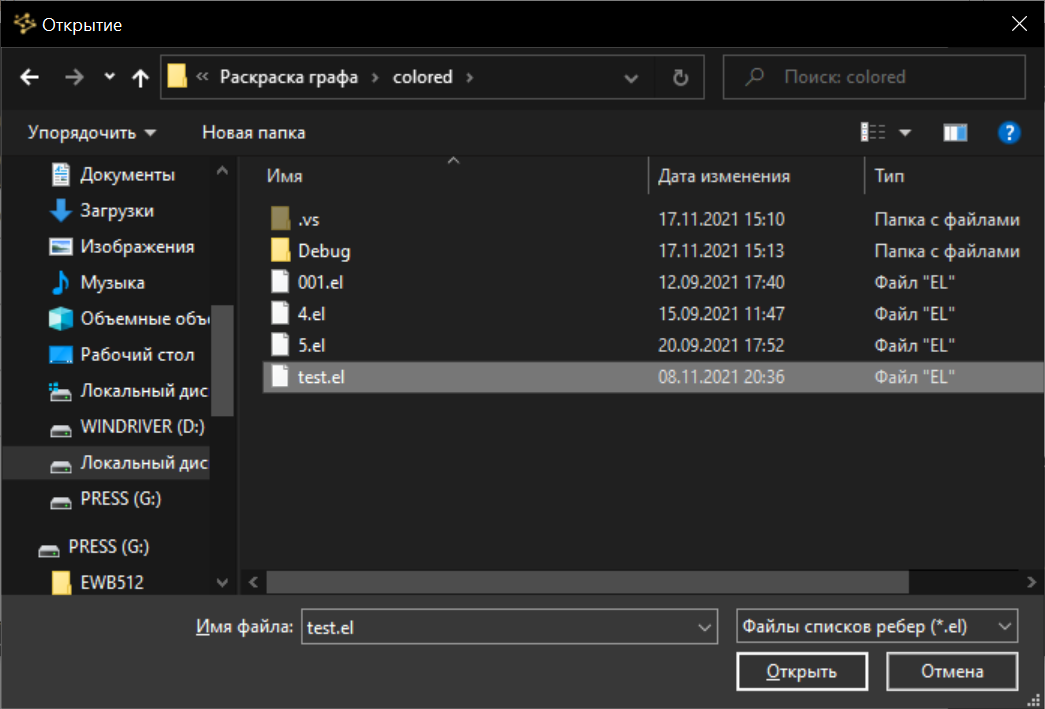


1. Главное окно программы

Работа с программой производится в следующей последовательности:

1) ввод графа.

* Ввод с клавиатуры. Задается количество вершин (максимально 15) и количество ребер - в соответствующих полях задаются нужные значения. Граф задается списком ребер. Для этого в таблицу вводятся пары чисел – указываются номер начальной вершины и номер конечной вершины ребра.
* Загрузка из файла. Чтобы загрузить данные из текстового файла используется команда меню «Ввод» - «Из файла». Пользователь в стандартном окне диалога выбирает файл (рис. 2) и программа загружает данные в элементы ввода.



1. Выбор файла с исходными данными

Структура файла данных:

|  |  |
| --- | --- |
| № п/п | Данные |
| 1 | количество вершин графа N |
| 2 | количество ребер графа М |
| 3 | ребро 1 |
| 4 | ребро 2 |
| … | … |
| М + 2 | ребро М |

Каждое ребро описывается двумя целыми значениями, разделенные пробелом: [вершина начала ребра] [вершина конца ребра]

* Заполнение случайными числами. Из меню «Ввод» выбирается команда «Случайный». Программа производит генерацию графа с полнотой 60%.

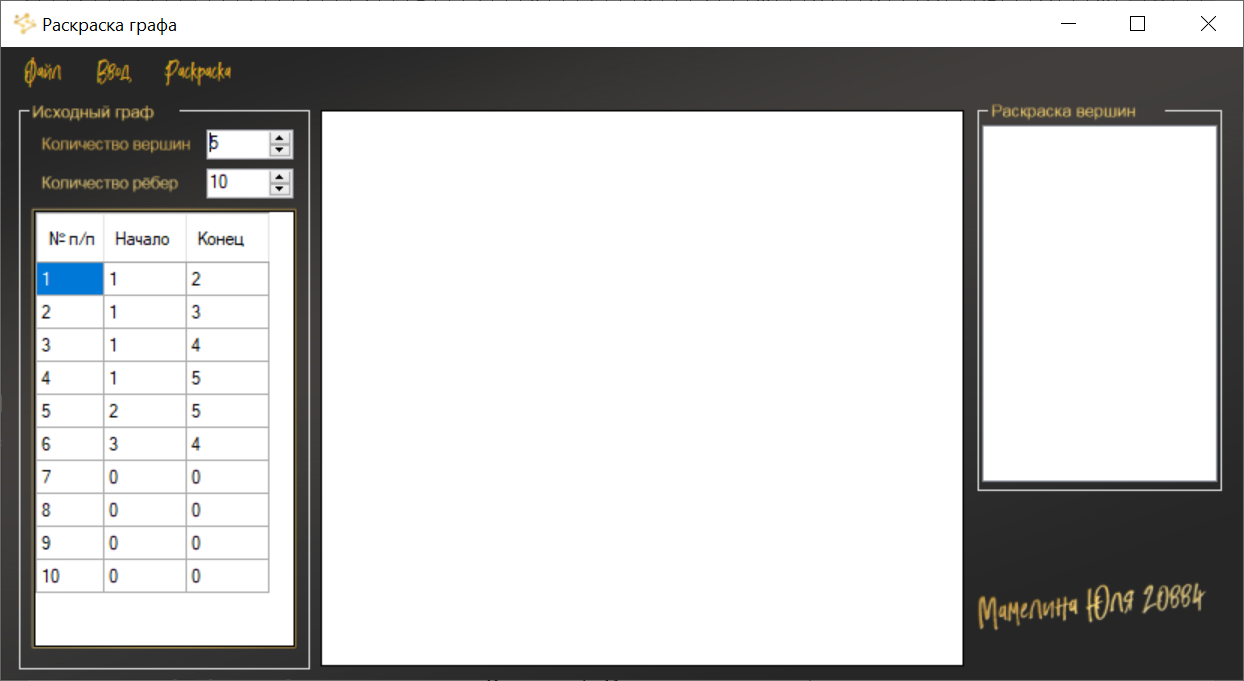
Введенные данные могут быть сохранены в текстовом файле – команда меню «Файл» - «Сохранить».

2) расчёт.

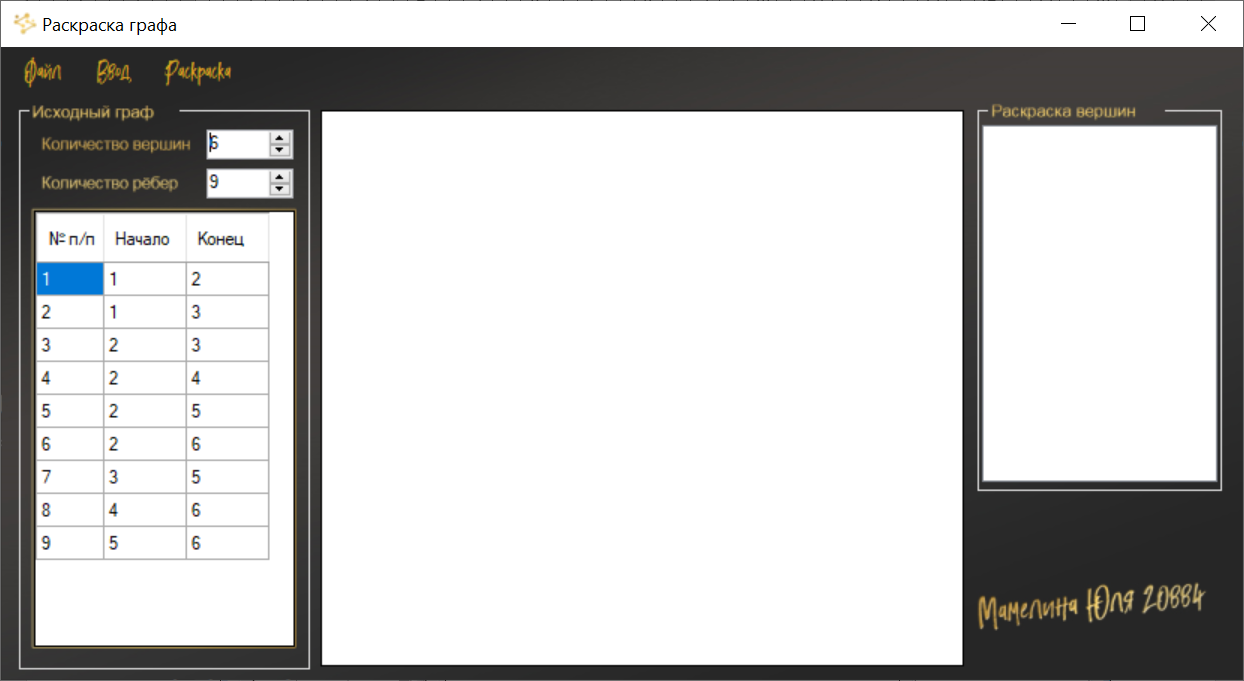
Для построения раскраски необходимо выбрать команду меню «Раскраска». Результат выводится в виде изображения графа с раскрашенными вершинами и текстовом виде: вершина - цвет. Также выводится необходимое количество цветов для раскраски.

# Тестирование

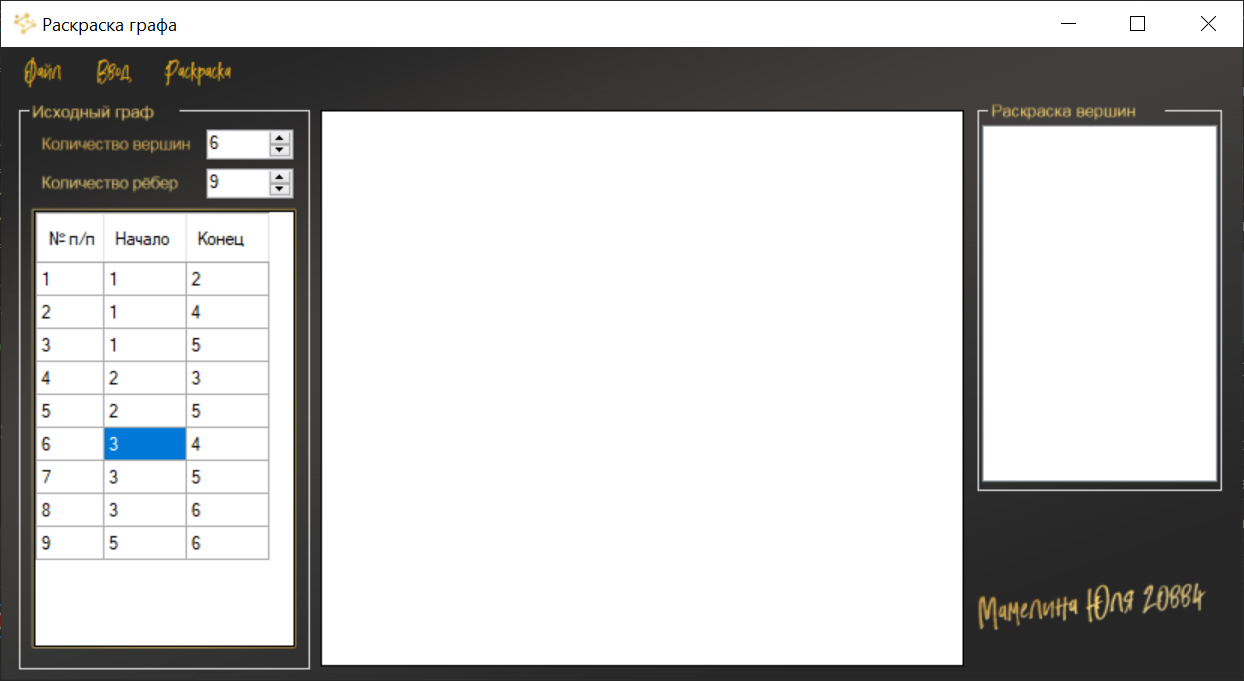
В данном разделе приведен результат тестирования программы при вводе пользователем различных графов.



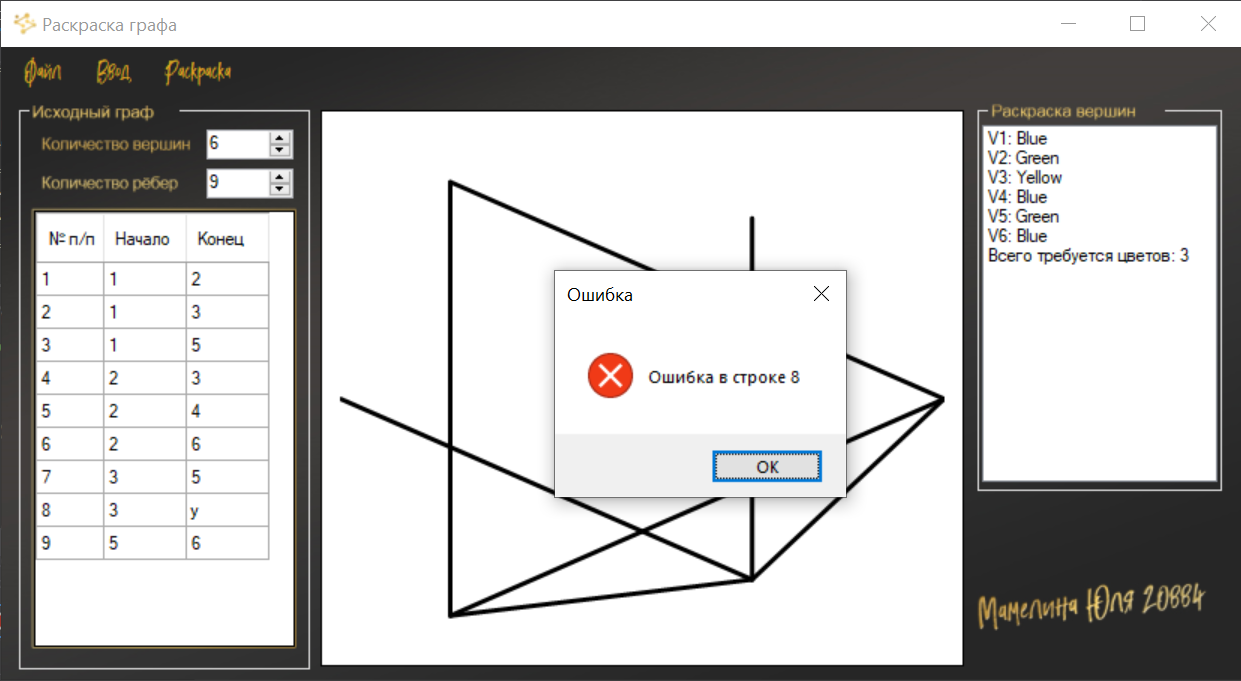
1. Изменение числа ребер

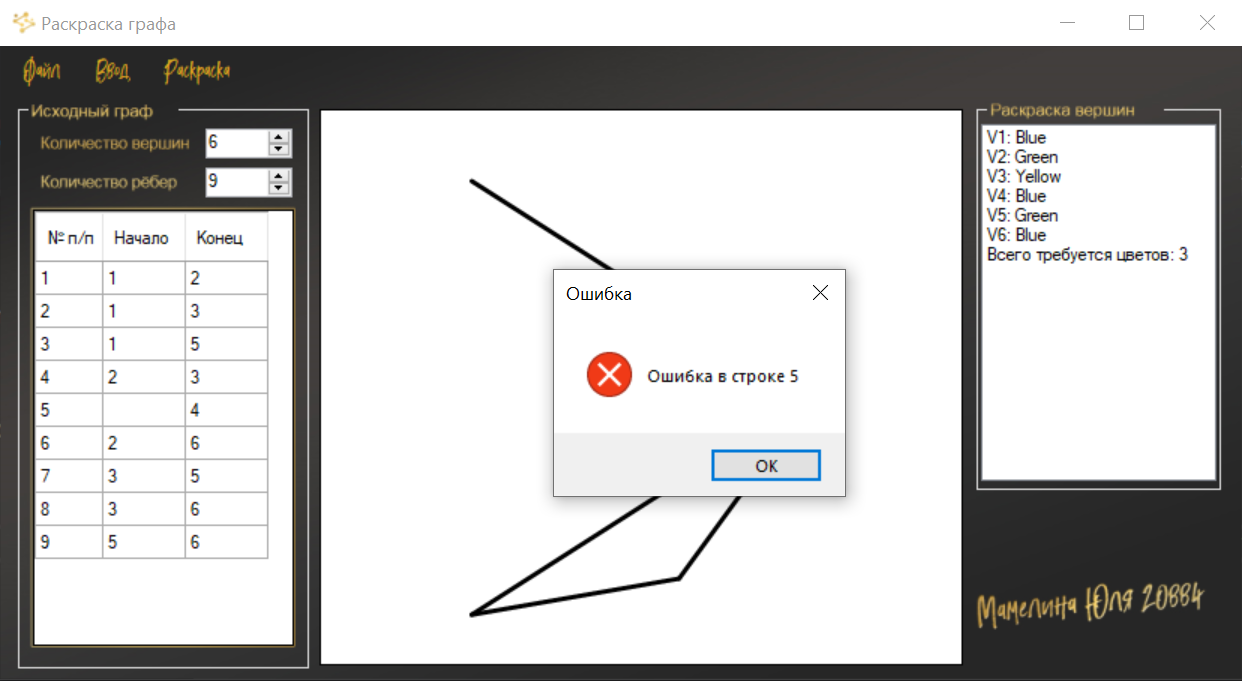


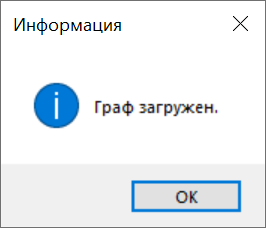
1. Формирование случайного графа



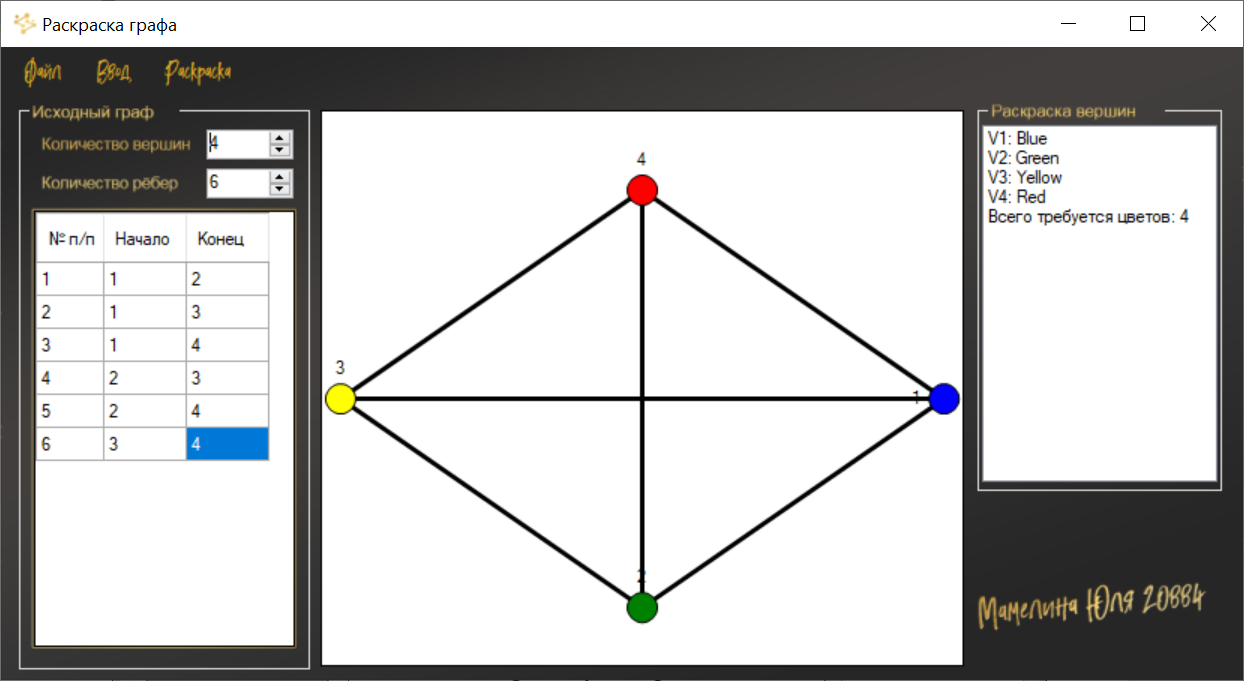
1. Ввод графа с клавиатуры



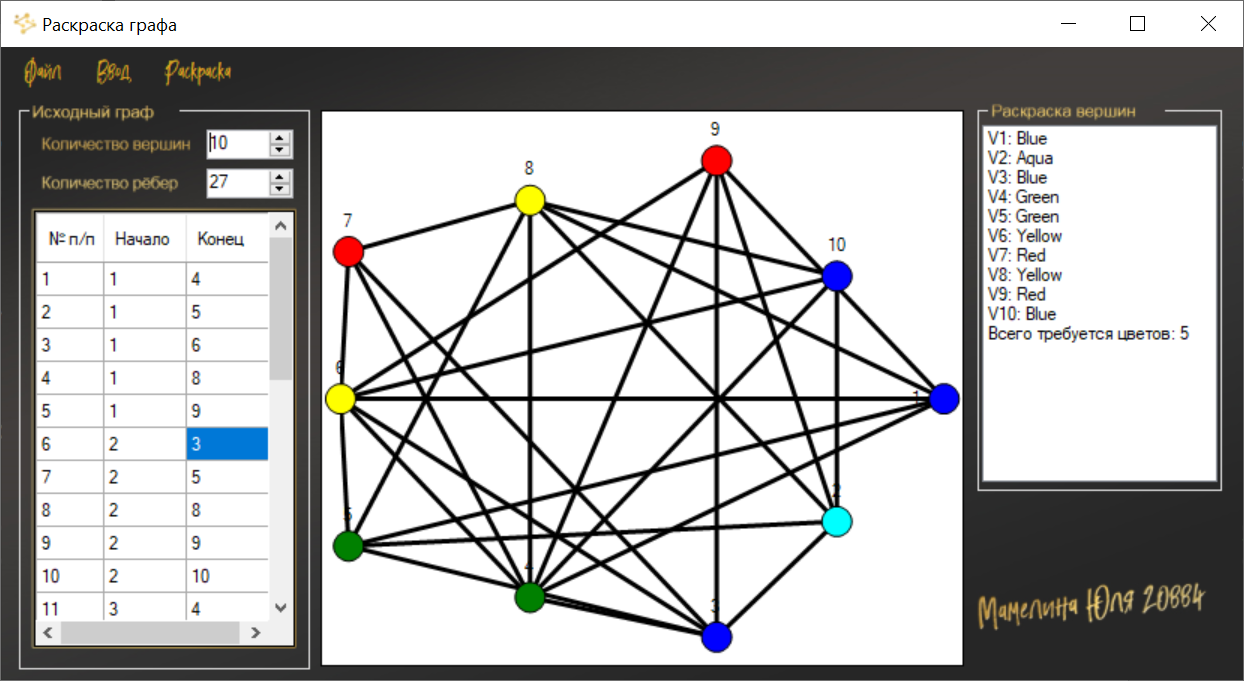
1. Нечисловое значение в ячейке таблицы ребер
2. Пустое значение в ячейке таблицы ребра



1. Загрузка графа

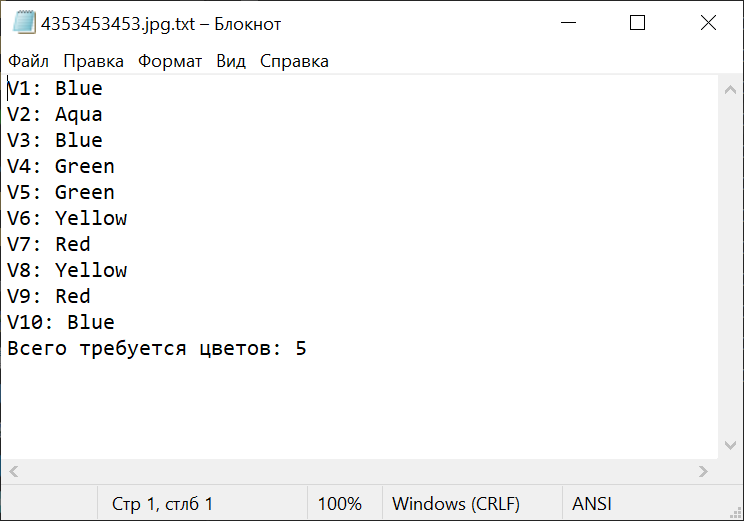
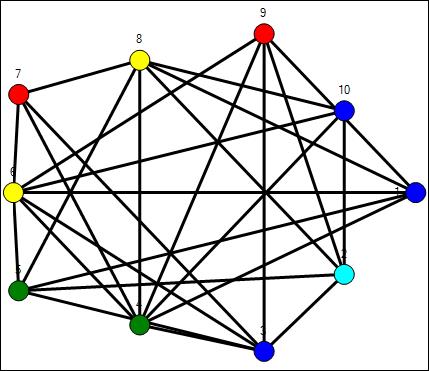


1. Раскраска полносвязного графа из 4 вершин



1. Раскраска произвольного графа из 10 вершин



1. Сохранение результата

Изображение графа Текстовое описание

1. Описание поведения программы при тестировании

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Вывод окна программы | Верно |
| Изменение числа ребер | Изменение числа строк таблицы | Верно |
| Формирование случайного графа | Заполнение таблицы ребер случайными значениями | Верно |
| Ввод графа с клавиатуры | Заполнение таблицы ребер значениями, вводимыми пользователем | Верно |
| Пустое или нечисловое значение в ячейке таблицы ребер | Вывод сообщение об ошибке | Верно |
| Загрузка графа из файла | Сообщение о выполнении загрузки и вывод данных из файла в соответствующие элементы ввода | Верно |
| Сохранение графа в файле | Сообщение о записи данных в файл | Верно |
| Раскраска полносвязного графа из 4 вершин | Вывод полносвязного графа с раскрашенными вершинами | Верно |
| Раскраска произвольного графа из 10 вершин | Вывод графа с раскрашенными вершинами | Верно |
| Сохранение результата | Сообщение о сохранении результатов. Два файла в текущем каталоге с программой: изображение (в формате jpeg и текстовый файл) | Верно |

В результате тестирования было выявлено, что программа успешно выполняет свои функции.

# Ручной расчёт задачи

Проведем проверку программы посредством ручных вычислений на примере графа с 8 вершинами.

Пример:

Раскрасить граф

1

2

3

4

5

6

7

8

1. Исходный граф

Вычислим степени:

1

2

3

4

5

6

7

8

3

3

2

2

3

3

2

2

1. Степени вершин

Рассматриваем вершины в порядке не возрастания числа связей. Те, которые не связаны с вершинами, раскрашенными в цвет №1, раскрашиваем в цвет №1.

1

2

3

4

5

6

7

8

1. Шаг 1

Рассматриваем вершины в порядке не возрастания числа связей. Те которые не связаны с вершинами раскрашенными в цвет №2 раскрашиваем в цвет №2.

1

2

3

4

5

6

7

8

1

2

3

4

5

6

7

8

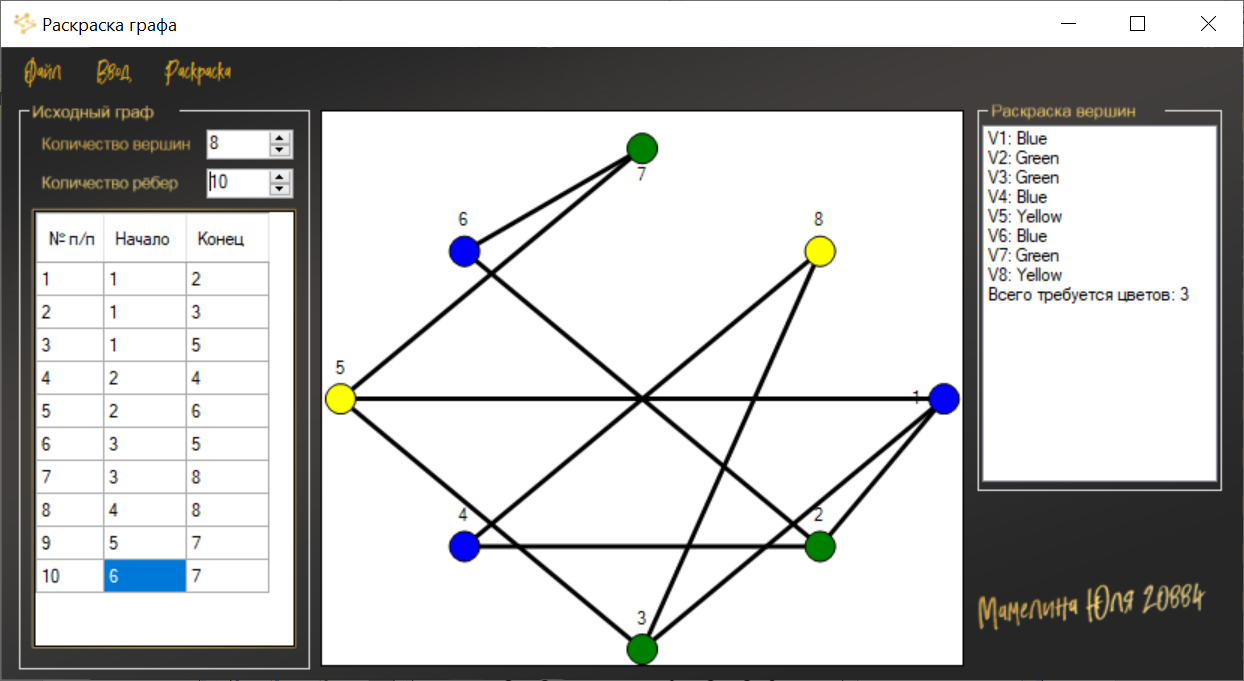
1. Шаг 2

Рассматриваем вершины в порядке не возрастания числа связей. Те, которые не связаны с вершинами, раскрашенными в цвет №3, раскрашиваем в цвет №3.

1. Шаг 3

Все вершины графа раскрашены, число цветов 3, следовательно, хроматическое число равно трём.

Загружаем граф в программу для проверки выполнения:



1. Тестирование работы программы

Исходя из полученных результатов, можно сделать вывод, что программа работает верно.

# Заключение

В результате выполнения работы была разработана программа, реализующая алгоритм построения раскраски графа в Microsoft Visual Studio 2019 на языке С++.

В процессе выполнения курсовой работы была рассмотрена такая важная тема, как раскраска графов. Задачи подобного рода встречаются довольно часто в повседневной жизни – ярким примером того может служить раскраска географических карт, на которых не должно быть граничащих областей с одинаковыми цветами. Использование алгоритмов, дающих оптимальную раскраску минимальным количеством цветов, и их реализация на ЭВМ позволяет значительно сократить время (в сравнении с ручным счетом) на получение правильной раскраски.

Созданная программа позволяет выполнять следующие действия:

* вводить граф с клавиатуры списком ребер;
* генерировать случайный граф заданной полноты;
* сохранять и загружать графы из файлов;
* выводить на экран цвета вершин найденной раскраски.

Программа имеет удобный оконный интерфейс, что позволяет наглядно выводить результат её работы.

Работа по созданию программы позволила приобрести практические навыки программирования прикладных задач теории графов.

На основе контрольных примеров получились верные результаты, что позволяет сделать вывод о правильной реализации алгоритма программы.

# Список литературы

1. Аверкин В.П., Бобровский А.И., Веснич В.В., Радушинский В.Ф., Хомоненко А.Д. Программирование на С++: Учебное пособие / Под ред. проф. А.Д. Хомоненко. -СПб.: Корона Принт, 2018. -256 с.
2. Белов В.В., Воробьев Е.М., Шаталов В.Е. Теория графов.– М.: Высшая школа, 2016. –451 с.
3. Васильев А. Н. Программирование на C++ в примерах и задачах. -М.: Эксмо, 2018. -368 с.
4. Зиборов В. В. MS Visual C++ 2010 в среде .NET. -СПб.: Питер, 2012. -320 с.
5. Кристофидес Н. Теория графов. Алгоритмический подход. –М.: Мир, 1978. –427 с.
6. Окулов С.М. Программирование в алгоритмах. –М.: Бином, 2019. –341 с.
7. Подбельский В. В. Язык Си++: Учебное пособие для вузов. -М.: Финансы и статистика, 2020. -560 с.
8. C/C++. Программирование на языке высокого уровня / Т. А. Павловская. –СПб.: Питер, 2017. –461 с.

# Приложение А. Листинг программы.

String^ sToS(string t)

//преобразование строки string в String

{

String^ t1;

t1 = gcnew String(t.c\_str());

return t1;

}

string STos(System::String^ t)

//преобразование строки String в string

{

System::IntPtr ip;

ip = Marshal::StringToHGlobalAnsi(t);

return static\_cast<const char\*>(ip.ToPointer());

}

void calcCoordinats(double\* xv, double\* yv, int n)

//формирует массив координат вершин, расположенных по двум концентрическим окружностям

//xv, yv - координаты вершин

//n - количество вершин

{

int i;//счетчики

double size = 240;//размер области вывода

double R1 = 100;//радиус внутренней окружности

double R2 = 1.2 \* R1;//радиус внешней окружности

double R;//текущий радиус

double a = 0;//начальный угол

for (i = 1; i <= n; i++)//считаем координаты

{

if (i % 2 == 0)

R = R1;

else

R = R2;

xv[i] = size / 2.0 + (R \* Math::Cos(a));

yv[i] = size / 2.0 - (R \* Math::Sin(a));

a = a + (double)(2 \* Math::PI / n);//изменяем угол

}

}

void createCharts(void)

//создает графики для отображения ребер и вершин графа

{

int i;//счетчик

for (i = 1; i <= 100; i++)//ребра

{

chart1->Series->Add("series" + (i + 1).ToString());

chart1->Series[i]->ChartType = System::Windows::Forms::DataVisualization::Charting::SeriesChartType::FastLine;

chart1->Series[i]->BorderWidth = 3;

chart1->Series[i]->BorderColor = System::Drawing::Color::Black;

chart1->Series[i]->Color = System::Drawing::Color::Black;

chart1->Series[i]->Points->Clear();

}

chart1->Series->Add("series102");//вершины

chart1->Series[101]->ChartType = System::Windows::Forms::DataVisualization::Charting::SeriesChartType::Point;

chart1->Series[101]->IsValueShownAsLabel = true;

chart1->Series[101]->IsVisibleInLegend = false;

chart1->Series[101]->Label = L"#INDEX";

chart1->Series[101]->MarkerBorderColor = System::Drawing::Color::Black;

chart1->Series[101]->MarkerSize = 20;

chart1->Series[101]->MarkerStyle = System::Windows::Forms::DataVisualization::Charting::MarkerStyle::Circle;

chart1->Series[101]->Name = L"Series102";

}

int Find(int pos, int\* Set){

//поиск минимального номера цвета, в который можно окрасить вершину

//pos - количество окрашенных вершин

//Set - массив цветов в которые уже окрашены вершины

int Min\_color, i, p;

Min\_color = 0;

do{

Min\_color++;

p = 0;

for (i = 0; i < pos; i++)

if (Set[i] == Min\_color){

p = 1;

break;

}

} while (p != 0);

return Min\_color;

}

private: System::Void edgeChange\_ValueChanged(System::Object^ sender, System::EventArgs^ e) {

//изменение числа ребер графа

int n;//число ребер

int i;//счетчик

n = Convert::ToInt32(edgeChange->Value);//читаем из строки ввода

dataGridView1->RowCount = n;//устанавливаем число строк таблицы

for (i = 1; i <= n; i++)//цикл формирования таблицы ввода

dataGridView1[0, i - 1]->Value = Convert::ToString(i);//выводим номера

}

private: System::Void vertexChange\_ValueChanged(System::Object^ sender, System::EventArgs^ e){

//изменение количества вершин

int n;//число вершин

n = Convert::ToInt32(vertexChange->Value);//читаем из строки ввода

edgeChange->Maximum = (n \* (n - 1)) / 2;//максимальное число ребер

edgeChange->Value = Convert::ToInt32((n \* (n - 1)) / 2);

}

private: System::Void menuExit\_Click(System::Object^ sender, System::EventArgs^ e) {

//выход из программы

Close();

}

private: System::Void randomGraph\_Click(System::Object^ sender, System::EventArgs^ e) {

//генератор псевдослучайного взвешенного графа

Random^ rnd = gcnew Random;//описываем генератор случайных чисел

int i, j, k,//счетчики

s;//показатель полноты графа

int RR[101];//массив меток сформированных вершин

int\*\* A;

int n;//количество вершин

int m;//количество ребер

n = Convert::ToInt32(vertexChange->Value);//читаем из строки ввода

s = 0;//обнулили

A = new int\* [n + 1];//инициализация массива

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

A[i] = new int[n + 1];

for (i = 1; i <= n; i++)

{

RR[i] = 0;

for (j = 1; j <= n; j++)

A[i][j] = 0;

}

for (i = 1; i <= n; i++)//цикл начального заполнения

if (RR[i] == 0){

do{

k = rnd->Next(n) + 1;

} while (k == i);

A[i][k] = 1;

A[k][i] = A[i][k];

RR[k] = 1;

s = s + 2;

}

s = int((n \* n - n) \* 60.0 / 100.0 - s);//определяем полноту

while (s > 1)//цикл окончательного формирования матрицы

{

i = rnd->Next(n \* n);

k = i % n + 1;

i = i / n + 1;

if (i != k && A[i][k] == 0){

A[i][k] = 1;

A[k][i] = A[i][k];

s = s - 2;

}

}

m = 0;//обнулили счетчик ребер

for (i = 1; i <= n; i++) //цикл подсчета числа ребер

for (j = i + 1; j <= n; j++)

if (A[i][j] != 0)

m++;

edgeChange->Value = m;//передаем в строку ввода

dataGridView1->RowCount = m;

m = 0;

for (i = 1; i <= n; i++){

for (j = i + 1; j <= n; j++)

if (A[i][j] != 0)//если есть ребро

{

dataGridView1[1, m]->Value = Convert::ToString(i);//выводим в ячейки

dataGridView1[2, m]->Value = Convert::ToString(j);

m++;//увеличили счетчик

}

}

}

private: System::Void menuColored\_Click(System::Object^ sender, System::EventArgs^ e) {

//раскраска графа

int i, j;//счетчики

int\*\* A;//матрица смежности графа

double\* xv, \* yv;//координаты вершин

int n;//количество вершин

int m;//количество ребер

int v1,//вершина начала ребра

v2;//вершина конца ребра

string colors[] = { "White", "Blue", "Green", "Yellow", "Red", "Aqua", "Tan", "Gold", "Fuchsia", "Gray", "Lime", "Brown", "Violet", "Purple", "Pink", "Coral", "Ivory" };//массив цветов

//"Белый", "Синий", "Зеленый", "Желтый", "Красный", "Бирюзовый", "Бежевый", //"Золотой", "Фуксия", "Серый", "Лайм", "Коричневый", "Фиалка", "Фиолетовый", //"Розовый", "Коралл", "Слоновая Кость"

int\* Colors; //массив цветов вершин

int\* FIFO; //структура данных очередь

int\* Set; //массив цветов в которые уже окрашены вершины

int cr;//количество цветов для раскраски

int Start,//номер начальной вершины

k, ////указатель на конец очереди

p, //указатель на начало очереди

cur, //номер очередной вершины

pos;//счетчик цветов в которые окрашены вершины

n = Convert::ToInt32(vertexChange->Value);//читаем из строки ввода

m = Convert::ToInt32(edgeChange->Value);//читаем из строки ввода

xv = new double[n + 1];//генерация экранных координат

yv = new double[n + 1];

A = new int\* [n];//инициализация массива

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

A[i] = new int[n];

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

for (j = 0; j < n; j++)

A[i][j] = 0;

chart1->Series[0]->Points->Clear();//очистка графика

for (i = 1; i <= 101; i++)

chart1->Series[i]->Points->Clear();

calcCoordinats(xv, yv, n);

for (i = 0; i < m; i++)

{

if (!Int32::TryParse(dataGridView1[1, i]->Value->ToString(), v1) ||//попытка преобразования строки в число

!Int32::TryParse(dataGridView1[2, i]->Value->ToString(), v2)){//ошибка при преобразовании - в строке не целое число

MessageBox::Show("Ошибка в строке " + (i + 1).ToString(), "Ошибка",

MessageBoxButtons::OK, MessageBoxIcon::Error);

for (j = 1; j <= 101; j++)

chart1->Series[j]->Points->Clear();

return;

}

A[v1 - 1][v2 - 1] = 1;

A[v2 - 1][v1 - 1] = 1;

chart1->Series[i + 1]->Points->AddXY(xv[v1], yv[v1]);

chart1->Series[i + 1]->Points->AddXY(xv[v2], yv[v2]);

}

Colors = new int[n]; //выделение памяти

FIFO = new int[n];

Set = new int[n];

for (i = 0; i < n; i++){ //обнуление массивов

Colors[i] = 0;

FIFO[i] = 0;

Set[i] = 0;

}

Start = 0; //выбираем начальную вершину

Colors[Start] = 1; //раскрашиваем ее в цвет 1

p = 0;

k = 1;

FIFO[p] = Start;

while (p != k){

cur = FIFO[p];

p++;

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

if (A[cur][i] == 1 && Colors[i] == 0){

FIFO[k] = i;

k++;

pos = 0;

for (j = 0; j < n; j++)

if (A[FIFO[k - 1]][j] == 1 && Colors[j] != 0){

Set[pos] = Colors[j];

pos++;

}

Colors[i] = Find(pos, Set);

}

}

listBox1->Items->Clear();

cr = 0;

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

if (Colors[i] > cr)

cr = Colors[i];

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

if (Colors[i] == 0)

Colors[i] = 1;

for (i = 1; i <= n; i++)//вывод решения

{

listBox1->Items->Add("V" + i.ToString() + ": " + sToS(colors[Colors[i - 1]]));

chart1->Series[101]->Points->AddXY(xv[i], yv[i]);//вывод на график

chart1->Series[101]->Points[i - 1]->Label = i.ToString();

chart1->Series[101]->Points[i - 1]->Color = Color::FromName(sToS(colors[Colors[i - 1]]));

}

listBox1->Items->Add("Всего требуется цветов: " + cr.ToString());

}

private: System::Void Form1\_Load(System::Object^ sender, System::EventArgs^ e) {//инициализация программы

createCharts();

vertexChange->Value = 5;

randomGraph\_Click(sender, e);

}

private: System::Void saveGraph\_Click(System::Object^ sender, System::EventArgs^ e)//сохранить граф в файле

{

//вывод диалога выбора файла

if (saveFileDialog1->ShowDialog() != System::Windows::Forms::DialogResult::OK)//если файл не выбран

return;

if (saveFileDialog1->FileName == "")

return;//если имя не задано, выход

int n;//количество вершин

int m;//количество ребер

int v1,//вершина начала ребра

v2;//вершина конца ребра

n = Convert::ToInt32(vertexChange->Value);//читаем из строки ввода

m = Convert::ToInt32(edgeChange->Value);//читаем из строки ввода

//открываем файл для чтения

ofstream f(STos(saveFileDialog1->FileName));

int i;//параметр цикла

f << n << endl;

f << m << endl;

for (i = 0; i < m; i++)//запись

{

if (!Int32::TryParse(dataGridView1[1, i]->Value->ToString(), v1) ||//попытка преобразования строки в число

!Int32::TryParse(dataGridView1[2, i]->Value->ToString(), v2))

{//ошибка при преобразовании - в строке не целое число

MessageBox::Show("Ошибка в строке " + (i + 1).ToString(), "Ошибка");

f.close();

return;

}

f << v1 << " " << v2 << endl;

}

f.close();

MessageBox::Show("Список ребер сохранен в файле.", "Информация",

MessageBoxButtons::OK, MessageBoxIcon::Information);

}

private: System::Void loadGraph\_Click(System::Object^ sender, System::EventArgs^ e)

//загрузить граф из файла

{

//вывод диалога выбора файла

if (openFileDialog1->ShowDialog() != System::Windows::Forms::DialogResult::OK)//если файл не выбран

return;

if (openFileDialog1->FileName == "")

return;//если имя не задано, выход

if (!File::Exists(openFileDialog1->FileName)){

MessageBox::Show("Файл не найден или к нему нет доступа!", "Ошибка");

return;

}

//открываем файл для чтения

ifstream f(STos(openFileDialog1->FileName));

int i;//параметр цикла

int n;//количество вершин

int m;//количество ребер

int v1,//вершина начала ребра

v2;//вершина конца ребра

f >> n;

f >> m;

vertexChange->Value = n;//передаем в строку ввода

edgeChange->Value = m;

dataGridView1->RowCount = m;

for (i = 0; i < m; i++)//

{

f >> v1 >> v2;

dataGridView1[1, i]->Value = Convert::ToString(v1);//выводим в ячейки

dataGridView1[2, i]->Value = Convert::ToString(v2);

}

f.close();

MessageBox::Show("Граф загружен.", "Информация",

MessageBoxButtons::OK, MessageBoxIcon::Information);

}

private: System::Void menuSaveResult\_Click(System::Object^ sender, System::EventArgs^ e)

//сохранить результат раскраски

{

if (saveFileDialog2->ShowDialog() != System::Windows::Forms::DialogResult::OK)//отмена сохранения

return;

chart1->SaveImage(saveFileDialog2->FileName, Imaging::ImageFormat::Jpeg);

ofstream f(STos(saveFileDialog2->FileName + ".txt"));

int i;//параметр цикла

for (i = 0; i < listBox1->Items->Count; i++)//запись

f << STos(listBox1->Items[i]->ToString()) << endl;

f.close();

MessageBox::Show("Результат сохранен в файле.", "Информация",

MessageBoxButtons::OK, MessageBoxIcon::Information);

}

};

};

# Приложение Б. Блок – схема алгоритма

Начало

Ввод pos, Set

Min\_color = 0

i = 0, pos

Set[i]=Min\_color

Конец

да

нет

Min\_color = Min\_color+1

p = 0

p=1

p ≠ 0

Вернуть Min\_color

да

нет

Блок – схема 1- функция поиска минимального номера цвета, в который можно окрасить вершину

Начало

Ввод N, Graf

Обнуление массивов Colors, FIFO, Set

p ≠ k

да

нет

Start = 0

Colors[Start] = 1

p = 0

k = 1

FIFO[p] = Start

i = 0, N

cur=FIFO[p]

p = p + 1

Graf[cur][i]=1 & Colors[i]=0

3

2

1

4

5

да

нет

3

2

1

4

5

FIFO[k]=i

j = 0, N

k = k + 1

pos = 0

Graf[FIFO[k–1]][j]=1 & Colors[j]≠0

да

нет

Set[pos]=Colors[j]

pos=pos+1

Colors[i]= Find(pos,Set)

Вывод массива цветов Colors

Конец

Блок – схема 2- алгоритм построения раскраски

1. Точнее, никакие две кривые, представляющие ребра графа, не пересекаются (геометрически) нигде, кроме инцидентной им обоим вершины (т.е. пересекаться могут только смежные ребра и притом лишь в «концевых точках». Белов В.В., Воробьев Е.М., Шаталов В.Е. Теория графов, 2016, 57 с. [↑](#footnote-ref-1)
2. Кристофидес Н. Теория графов. Алгоритмический подход. –М.: Мир, 1978. –188 с. [↑](#footnote-ref-2)