Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

К курсовому проектированию по курсу

,,Логика и основы алгоритмизации в инженерных задачах”

на тему ,,Реализация алгоритма раскрашивания графа”

Выполнил:

Студент группы 21ВВ2

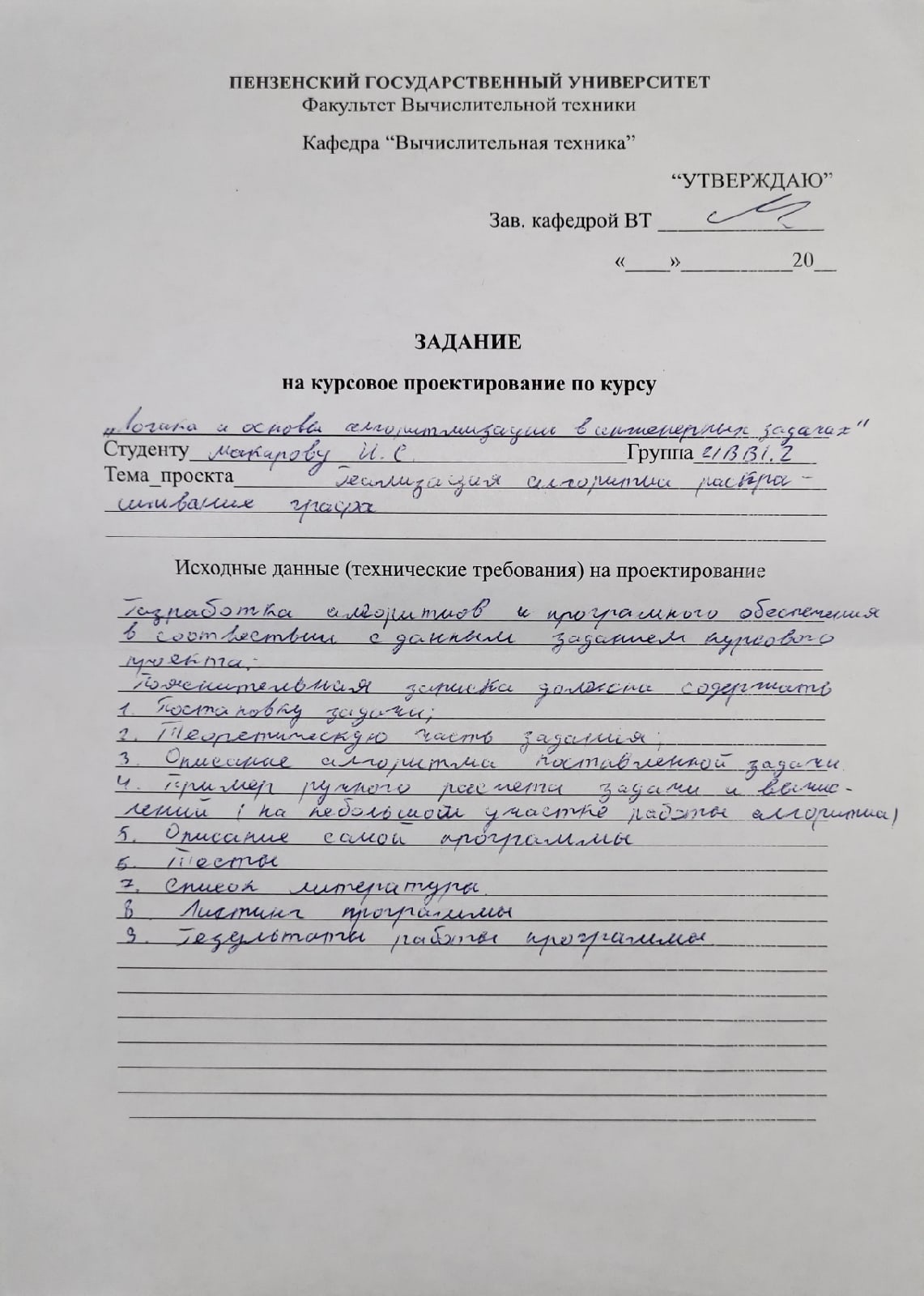
Макаров И.С.

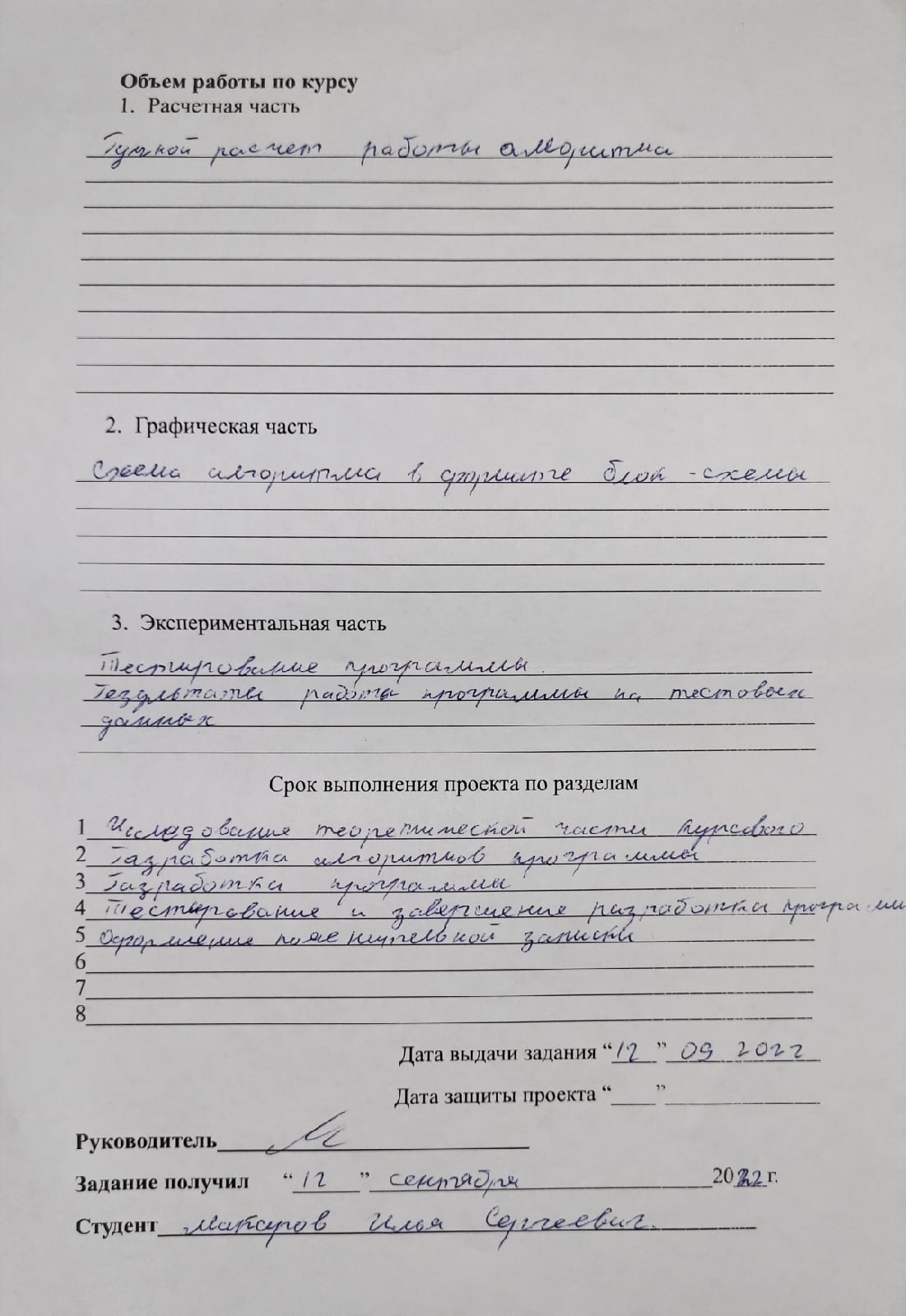
Приняли:

Митрохин М.А.

Юрова О.В.

Пенза, 2022





**Содержание**

[Реферат 4](#_Toc119614976)

[Введение 5](#_Toc119614977)

1. [Постановка задачи 7](#_Toc119614978)

2. [Теоретическая часть задания 8](#_Toc119614979)

3. [Описание алгоритма программы 11](#_Toc119614980)

4. [Описание программы 15](#_Toc119614981)

5. [Тестирование 18](#_Toc119614982)

6. [Ручной расчёт задачи 28](#_Toc119614983)

[Заключение 31](#_Toc119614984)

[Список литературы 32](#_Toc119614985)

[Листинг программы 33](#_Toc119614986)

# Реферат

# Отчет 38 стр, 22 рисунка.

# хроматическое число, раскраска графа, граф

Цель курсовой работы – изучить основные понятия теории раскрашивания плоских графов и проанализировать известные результаты о гипотезе четырех красок.

Достижению данной цели курсовой работы были поставлены и решены следующие задачи: изучить такие основополагающие понятия теории графов, как граф, маршрут и контур, раскраска и плоский граф; раскрыть понятие хроматического числа и хроматического многочлена графа; изучить достаточные условия раскраски графов и проанализировать известные результаты о гипотезе четырех красок.

# Введение

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

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.

Вообще говоря, хроматическое число графа (так же как числа независимости и доминирования, рассмотренные в предшествующей главе) нельзя найти, зная только числа вершин и ребер графа. Недостаточно также знать степень каждой вершины, чтобы вычислить хроматическое число графа. В этом можно убедиться, рассматривая графы, приведенные на рис.1(а) и рис.1(б). Эти графы имеют по n=12 вершин, m=16 ребер и одинаковое распределения степенен вершин. Однако хроматические числа данных графов равны 4 и 2 соответственно. При известных величинах n (число вершин), m (число ребер) и (степени вершин графа) можно получить верхнюю и нижнюю оценки для хроматического числа графа. Этим оценкам посвящен следующий раздел.

 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. В этой главе, однако, мы не пытаемся обсудить эти результаты или хотя бы дать их краткий обзор. Мы вводим только такие понятия, которые нужны для построения алгоритмов решения задачи о раскраске графа. Здесь мы рассматриваем в основном алгоритмы (как точные, так и «приближенные»), позволяющие находить (точное или приближенное) значение хроматического числа произвольного графа и соответствующую этому значению раскраску вершин.

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

Целью данной курсовой работы является разработка программы на языке С++, который является широко используемым. Именно с его помощью в данном курсовом проекте реализуется алгоритм раскраски графа.

# Постановка задачи

Исходный граф в программе должен задаваться матрицей смежности, причём при генерации данных должны быть предусмотрены граничные условия. Программа должна работать так, пользователь вводит количество вершин для генерации матрицы смежности, выбирает способ заполнения матрицы смежности, случайным способом или заполнить самому. После обработки этих данных на экран должна выводиться матрица смежности графа, все компоненты связности графа, результат окраски графа. Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно.

Устройство ввода – клавиатура и мышь.

Задания выполняются в соответствии с вариантом №24.

# Теоретическая часть задания

Графом называется набор точек (эти точки называются вершинами), некоторые из которых объявляются смежными (или соседними). Считается, что смежные вершины соединены между собой ребрами (или дугами). Таким образом, ребро определяется парой вершин. Два ребра, у которых есть общая вершина, также называются смежными (или соседними). Граф определяется как совокупность множества М с заданным на нем бинарным отношением G=<M.T>.

Между элементами М и Т определено отношение инцидентности, т.е. связи между двумя элементами множества М через один элемент множества Т, представлено на рисунке 1.

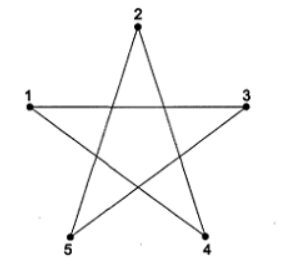


Рисунок 1 – пример графа

Хроматический многочлен считает число возможных вариантов раскраски графа с использование не более чем заданного числа цветов.

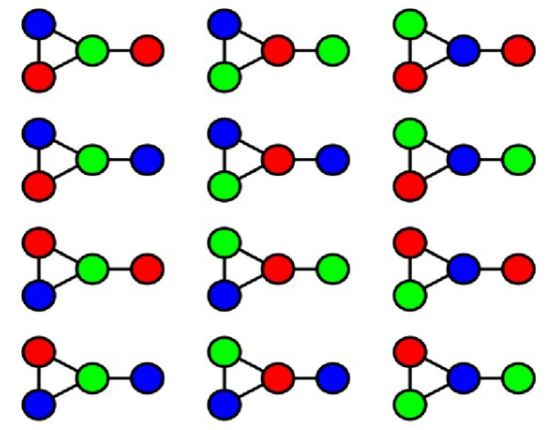


Рисунок 2 - пример раскраски

Например, граф на рисунке 2 может быть раскрашен 12способами с использованием 3 цветов. Двумя цветами он не может быть окрашен в принципе.

Используя 4 цвета, получаем 24+4\*12 = 72 вариантов раскраски: при использовании всех 4 цветов, есть 4! = 24 корректных способа (каждое присвоение 4 цветов для любого графа из 4 вершин является корректным); и для каждых 3 цветов из этих 4 есть 12способов раскраски.

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 S(G), затем окрашивается в следующий цвет множество S(X\S(G)) и так далее до тех пор, пока не будут раскрашены все вершины.

Тот факт, что такая раскраска, использующая только r цветов, всегда существует, может быть установлен следующим образом. Пусть существует раскраска в r цветов, такая, что одно или больше множеств, окрашенных в один и тот же цвет, не являются максимальными независимыми множествами в смысле, упомянутом выше. Перенумеруем цвета произвольным способом. Очевидно, что мы можем всегда покрасить в цвет 1 те вершины (пусть это множество Vi′), которые не были окрашены в этот цвет и которые образуют максимальное независимое множество вместе с множеством Vi всех вершин графа, уже окрашенных в цвет 1. Эта новая раскраска возможна потому, что никакая вершина из множества Vi′ не является смежной ни с какой вершиной из Vi′ и, следовательно, всякая вершина, которая смежна хотя бы с одной вершиной из Vi′, окрашена в цвет, отличный от цвета 1, и поэтому не затрагивается процедурой перемены цвета вершин из Vi′. Рассматривая теперь подграф (X − Vi′) и проводя с ним аналогичные манипуляции, мы окрасим в цвет 2 какое-то (новое) максимальное независимое множество и т. д.

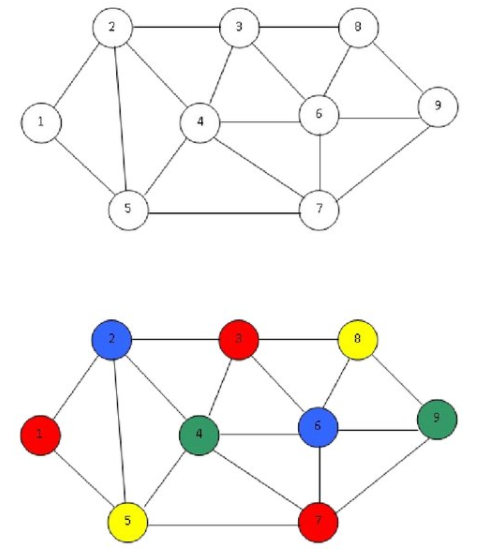


Рисунок 3-Пример раскраски графа

# 3. Описание алгоритма программы

*1. Покрасьте первую вершину в первый цвет.   
2. Сделайте следующее для оставшихся вершин V-1.   
а) Рассмотрим вершину, выбранную в данный момент, и покрасим ее в цвет с наименьшим номером, который не использовался ни в одной из ранее   
окрашенных смежных с ней вершин. Если все ранее использовавшиеся цвета   
появляются на вершинах, смежных с v, назначьте ей новый цвет.*

Существует много эвристических процедур раскрашивания графов, позволяющих находить хорошие приближения для хроматического числа графа в тех случаях, когда размеры графа слишком велики и получение оптимальной раскраски точными методами, упоминавшиеся ранее, затруднительно. В настоящем разделе дается краткое описание одной из таких процедур и ряда ее разновидностей. Данная процедура относится к последовательным методам, основанным на упорядочивании множества вершин.

В этом простейшем из методов вершины вначале располагаются в порядке не возрастания их степеней. Первая вершина окрашивается в цвет 1; затем список вершин просматривается сверху вниз (по не возрастанию степеней) и в цвет 1 окрашивается всякая вершина, которая не смежная с другой, уже окрашенной в этот цвет. Потом возвращаемся к первой в списке неокрашенной вершине, окрашиваем ее в цвет 2 и снова просматриваем список вершин сверху вниз, окрашивая в цвет 2 любую неокрашенную вершину, которая, но соединена ребром с другой, уже окрашенной в цвет 2 вершиной. Аналогично действуем с цветами 3, 4 и т.д., пока не будут окрашены все вершины. Число использованных цветов будет тогда приближенным значением хроматического числа графа.

Простая модификация описанной выше эвристической процедуры состоит в переупорядочивании неокрашенных вершин после окраски каждой очередной вершины: оставшиеся неокрашенные вершины записываются в порядке не возрастания их «относительных» степеней, т.е. степеней в таком графе, который получается из данного после удаления окрашенных вершин (вместе с ребрами, инцидентными удаленным вершинам).

В этой процедуре молчаливо предполагалось, что если две вершины имеют одинаковые степени, то их взаимное положение в списке случайно. В таких ситуациях уточнение в размещении вершин можно осуществлять с помощью двухшаговых степеней вершин, имеющих одинаковые степени (одинаковые 1-шаговые степени), где определяется как число маршрутов длины 2. исходящих из. Эти вершины могут быть размещены тогда в соответствии с величинами степенен. Если все-таки найдутся вершины, у которых совпадают и степени, и степени, то можно вычислить трехшаговые степени (определяемые аналогичным образом) и разместить вершины с учетом степеней и т.д.

Итак, имеется граф G=(V, E). Создать динамический массив result[V](тип: bool) размером количества наших вершин и первой ячейки присвоить нулевое значение (присваивает цвет). Все оставшиеся вершины V-1 как не назначенные. Временный массив для хранения доступных цветов available(тип: bool). Правда значение available[cr] будет означать что цвет cr равен присвоен одной из его смежных вершин. Назначить цвета оставшимся вершинам V-1(запуск цикла от i=1 до V, для прохода по всем вершинам). Обработал все смежные вершины и отметил их цвета как отсутствующие (запустил цикл от j(iterator)=первому элементу контейнера[i] до последнего элемента. Если resilt[\*j] != -1 то во временный массив присваиваем false.). Дальше нахожу первый допустимый цвет прогнав цикл. Присваиваю нужный цвет resilt[i] = cr, после сбрасываю до false для возврата. Алгоритм продолжается до тех пор, пока не будут найдены цвета для всех вершин графа.

Ниже представлен псевдокод функции GraphColoring:

1. Создать динамический массив resilt
2. Создать контейнер result
3. первое значение resilt = 0
4. для i=1 пока i<V делать i=i+1
5. resilt[i] = -1
6. конец цикла
7. Создать динамический массив available
8. для cr=0 пока i<V делать i=i+1
9. available [cr] = ложь
10. конец цикла
11. для i=1 пока i<V делать i=i+1
12. для итератора j=начало списка пока j != концу списка делать j=j+1
13. если значение массива resilt[итератор j] != -1
14. available[resilt[итератор j]] = Правда
15. конец условия
16. конец цикла
17. для cr=0 пока i<V делать i=i+1
18. если для массива available [cr] = ложь
19. выход из цикла
20. конец условия
21. конец цикла
22. n=cr
23. массиву resilt[i] = cr
24. для итератора j=начало списка пока j != концу списка делать j=j+1
25. если значение массива resilt[итератор j] != -1
26. available[resilt[итератор j]] = Ложь
27. конец условия
28. конец цикла
29. конец цикла
30. вывод " Результата раскраски"
31. Запись в result " Результата раскраски"
32. вывод " Результата раскраски" в виде списка
33. Освобождение памяти динамических массивов

# Описание программы

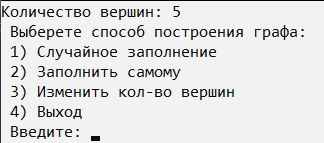
Для написания данной программы использован язык программирования С++. Язык программирования С++ - универсальный язык программирования, который завоевал особую популярность у программистов, благодаря сочетанию возможностей языков программирования высокого и низкого уровней.

Проект был создан в виде консольного приложения Win32 (Visual C++).

Работа программы начинается с ввода количества вершин.



Потом перед пользователем стоит выбор в главном меню.



Если пользователь выбрал заполнения матрицы случайным образом ему придётся ввести вероятности присутствия ребра между вершинами.

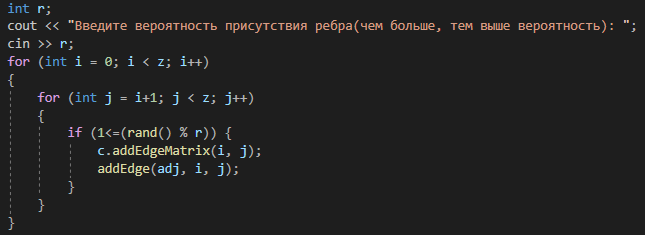


Рисунок 4

Если же пользователь выбрал заполнения матрицы самому ему придётся вводить поочередную связь между вершинами, пока он самостоятельно не выйдет из этого из цикла.

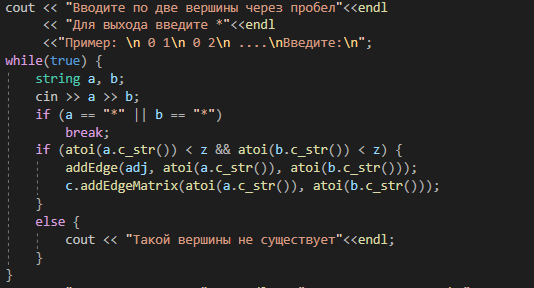


Рисунок 5

Для программной реализации алгоритма понадобиться класс Graph(рисунок 6), который содержит в себе матрицу смежности и всю информацию о нем, а также функции для её вывода или очистки для повторной работы.

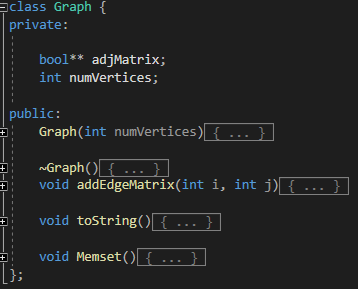


Рисунок 6

Конструктор(рисунок 7) заботится в программе за выделение динамической памяти матрице adjMatrix(тип: bool) и присвоению переменной numVertices(тип: int) количества вершин, вводимы пользователем. Деструктор(рисунок 7) выполняет обратную операцию, очищает выделенную динамическую память по завершению программы.

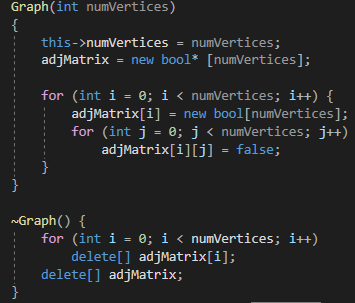


Рисунок 7

Функция addEdgeMatrix(int, int) (рисунок 8) нужна для заполнения матрицы смежности о связях между вершинами. Функция toString() (рисунок 8) выполняет задачу вывода матрицы на экран. Функция Memset() (рисунок 8) обнуляет матрицу для повторной работы над ней, чтобы программа при повторном вводе выдавала верные значения.

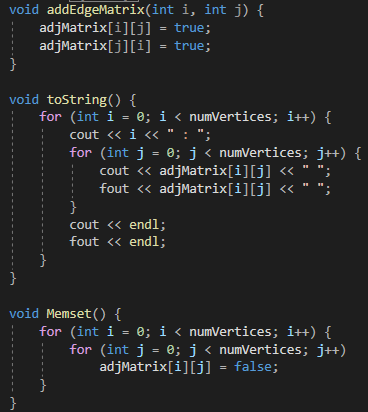


Рисунок 8

Матрица смежности нужна в программе только для вывода информации в более подробном формате, работа же над графом происходит с помощью контейнера set из STL библиотек языка C++. Контейнер не может хранить в себе одинаковые значения и т.к. он представляет собой бинарное дерево переменные всегда отсортированы. Добавление вершин происходит в отдельной функции addEdge(set<int>, int, int) (рисунок 9).

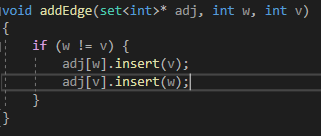


Рисунок 9

# Тестирование

Среда разработки Microsoft Visual Studio 2019 предоставляет все средства, необходимые при разработке и отладке многомодульной программы.

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с вводом данных, изменением дизайна выводимых данных, алгоритмом программы, взаимодействием функций.

Ниже продемонстрирован результат тестирования программы при вводе пользователем различных количеств вершин и вывода результата.

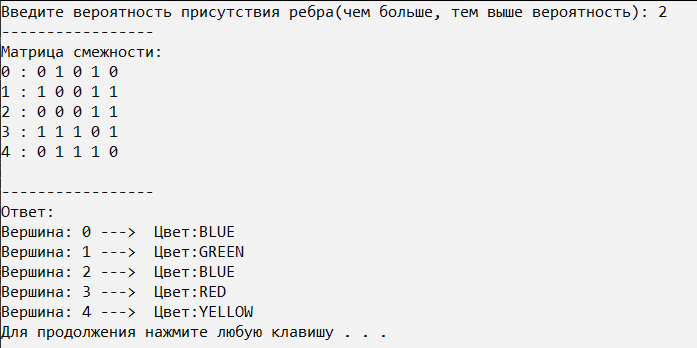
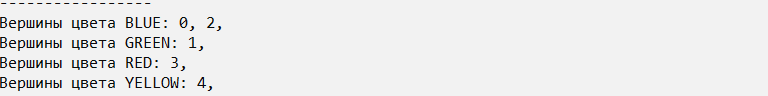


Рисунок 10 – тестирование при случайном заполнении графа при 5 вершинах и вероятности присутствия ребра равному 0.5



Рисунок 11 – вывод тестирования сверху в файл

Изменим количество вершин и вероятность присутствия вершин.

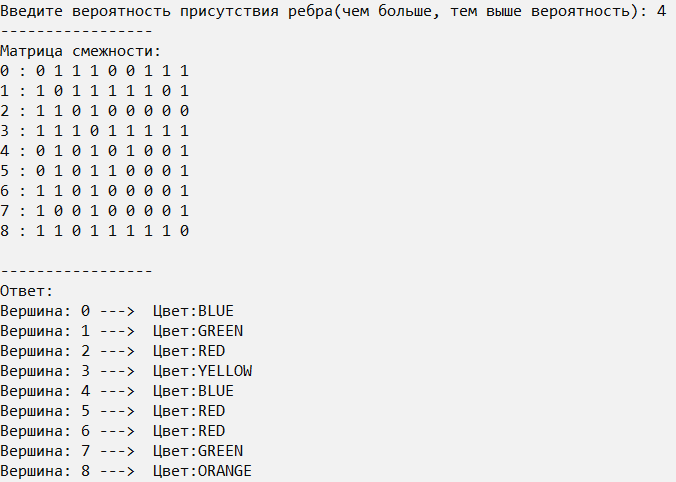


Рисунок 12 – тестирование при случайном заполнении графа при 9 вершинах и вероятности присутствия ребра равному 0.25



Рисунок 13 – вывод тестирования сверху в файл

Теперь введем связь вершин самостоятельно.

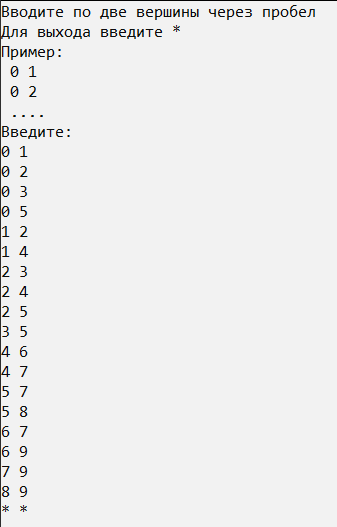


Рисунок 14 – ввод связи между вершинами

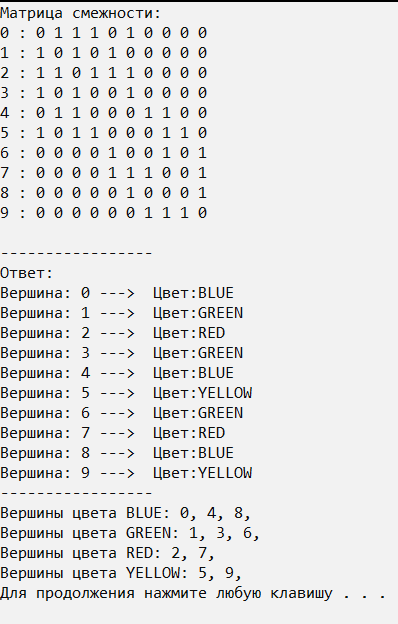


Рисунок 15 – результат тестирования введенных данный для 10 вершин

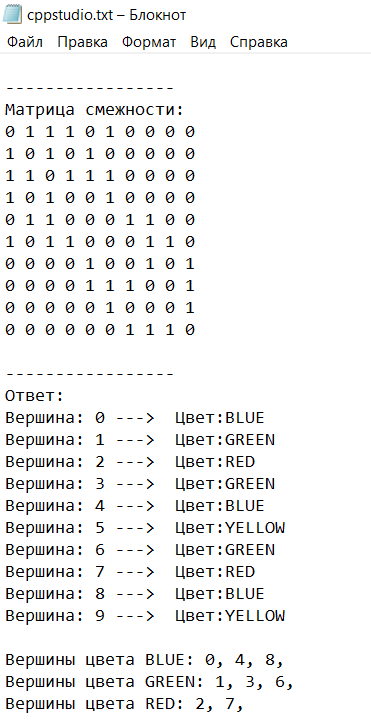


Рисунок 16 - вывод тестирования сверху в файл

Изменим количество вершин графа и протестируем по новой.

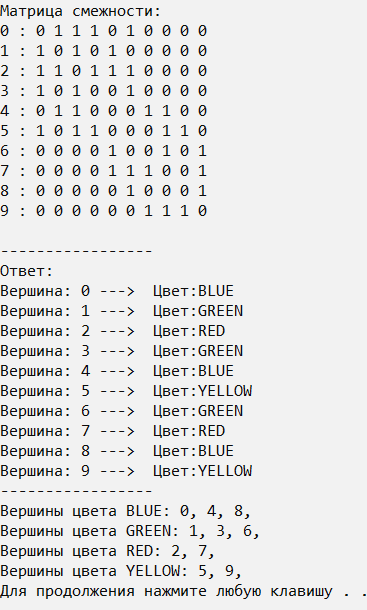


Рисунок 17 – ввод связи вершин и вывод результата для 4 вершин

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Вывод сообщения о вводе кол-ва вершин | Верно |
| Выбор генерации матрицы | Вывод меню программы и выбор пользователем нужного пункта | Верно |
| Заполнение матрицы случайными числами | Ввод вероятности и вывода матрица смежности | Верно |
| Самостоятельное и случайное заполнение матрицы смежности, корректный вывод результата | Вывод матрицы смежности с n вершинами. Вывод результата в виде раскраски графа и цветов каждой вершины соответствуя алгоритму. | Верно |

В результате тестирования было выявлено, что программа успешно выдает результат.

# Ручной расчёт задачи

Проведем проверку программы посредством ручных вычислений на примере графа самостоятельного ввода с 10 вершинами (рисунок 16) и случайного заполнения с вершинами (рисунок 10).

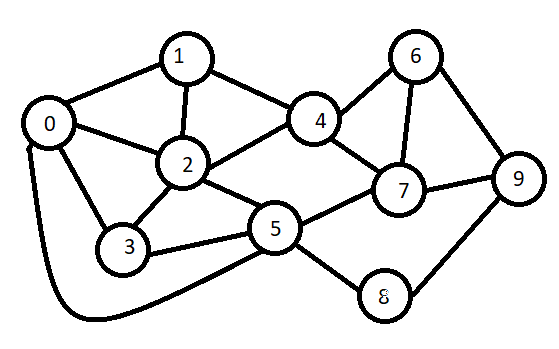
Начнем с графического изображения графа для облегчения работы и визуального представления графа. 

Рисунок 18 – графическое представление графа

Раскрасим первую (индекс 0) вершину в первый цвет (синий). После чего найдем следующую по номеру (индекс 4), но не связную с первой, вершину. Пройдемся по графу и найдем следующие вершины, которые не связаны с окрашенными вершинами. Такими оказалась одна вершина (индекс 8).

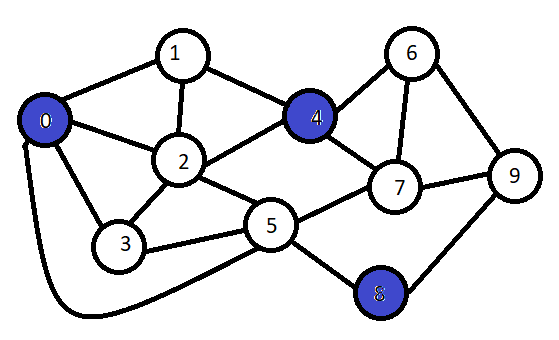


Рисунок 19 – результат окраски графа после первого прохода

Вернемся в начало графа для поиска второго цвета (на зеленый). Первая вершина уже окрашена, поэтому ищем следующую по очередности (индекс 1) и красим ее в наш цвет. Используя уже известный алгоритм, найдем другие вершины с таким же цветом, исключая уже окрашенные вершины в ранний цвет.

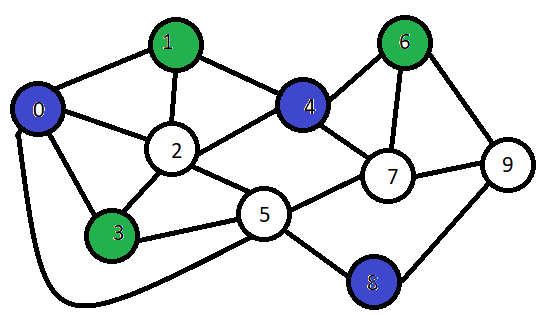


Рисунок 20 - результат окраски графа после двух проходов

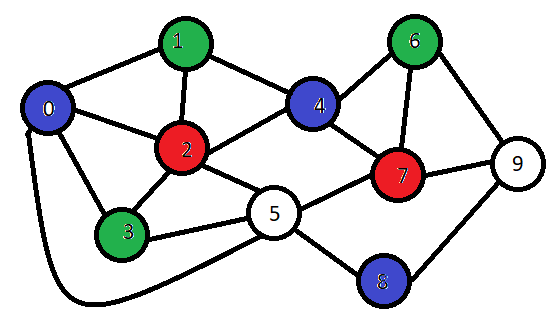
Изменяем цвет для следующих вершин (на красный). Пройдя по графу получим, что вершины, соответствующие условию, имеют индексы 2 и 7 

Рисунок 21 - результат окраски графа после трёх проходов

Следующий цвет жёлтый. Вершина 5 и 9 соответствуют условия и являются последними неокрашенными в графе, поэтому получаем окончательный результат.

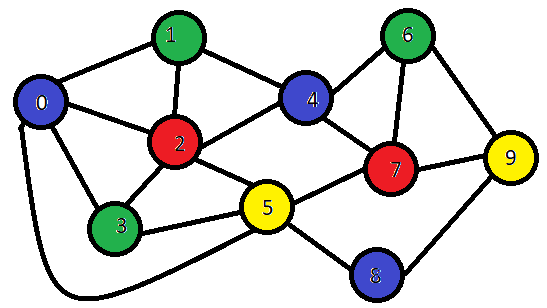


Рисунок 22 – окончательный результат раскраски графа

Сверив наш с результатом выданным программой, можно сделать вывод, что код работает верно.

# Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм раскраски графа в Microsoft Visual Studio 2019.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежностей, основанных на теории неопределенных графов. Приобретены навыки по осуществлению алгоритма раскраски графа. Углублены знания языка программирования C++.

Программа имеет небольшой, но достаточный для использования функционал возможностей.

# Список литературы

1. Кристофидес Н. «Теория графов. Алгоритмический подход» - Мир, 1978
2. Герберт Шилдт «Полный справочник по C++» - Вильямс, 2006
3. Уилсон Р. Введение в теорию графов. Пер. с анг. 1977. 208 с.
4. Лекция по дискретной математике, по теме Раскраска графа - <https://en.ppt-online.org/45056>
5. Статья о раскраске графа - https://www-geeksforgeeks-org.translate.goog/graph-coloring-applications/?\_x\_tr\_sl=auto&\_x\_tr\_tl=ru&\_x\_tr\_hl=ru

# Листинг программы

#include <iostream>

#include <string>

#include <set>

#include <fstream>

#include "Windows.h"

using namespace std;

ofstream fout("cppstudio.txt");

string color[] = {

"BLUE", "GREEN", "RED", "YELLOW", "ORANGE", "PINK",

"BLACK", "BROWN", "WHITE", "PURPLE", "VOILET", "MAGENTA",

"CYAN", "GREY", "PLUM", "GOLDEN", "SILVER"

};

// Для построения матрицы смежности

class Graph {

private:

bool\*\* adjMatrix;

int numVertices;

public:

Graph(int numVertices)

{

this->numVertices = numVertices;

adjMatrix = new bool\* [numVertices];

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

adjMatrix[i] = new bool[numVertices];

for (int j = 0; j < numVertices; j++)

adjMatrix[i][j] = false;

}

}

~Graph() {

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

delete[] adjMatrix[i];

delete[] adjMatrix;

}

void addEdgeMatrix(int i, int j) {

adjMatrix[i][j] = true;

adjMatrix[j][i] = true;

}

void toString() {

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

cout << i << " : ";

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

cout << adjMatrix[i][j] << " ";

fout << adjMatrix[i][j] << " ";

}

cout << endl;

fout << endl;

}

}

void Memset() {

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

for (int j = 0; j < numVertices; j++)

adjMatrix[i][j] = false;

}

}

};

void addEdge(set<int>\* adj, int w, int v)

{

if (w != v) {

adj[w].insert(v);

adj[v].insert(w);

}

}

void GraphColoring(set<int>\* adj, int V) {

int\* resilt = new int[V];

// Присвоить первый цвет первой вершине

resilt[0] = 0;

// Оставшиеся вершины V-1 как неназначенные

for (int i = 1; i < V; i++)

resilt[i] = -1;

// Временный массив для хранения доступных цветов. Правда значение available[cr] будет означать

// что цвет cr равен присвоен одной из его смежных вершин

bool\* available = new bool[V];

for (int cr = 0; cr < V; cr++)

available[cr] = false;

// Назначить цвета оставшимся вершинам V-1

for (int i = 1; i < V; i++)

{

// обработал все смежные вершины и отметил их цвета как отсуствующие

for (auto j = adj[i].begin(); j != adj[i].end(); ++j)

if (resilt[\*j] != -1)

available[resilt[\*j]] = true;

// Нашел первый доступный цвет

int cr;

for (cr = 0; cr < V; cr++)

if (available[cr] == false)

break;

resilt[i] = cr; // присвоение нужного цвета

// сбросил до false, для возврата

for (auto j = adj[i].begin(); j != adj[i].end(); ++j)

if (resilt[\*j] != -1)

available[resilt[\*j]] = false;

}

// Вывод рузультата

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

cout << "Вершина: " << i << " ---> Цвет:" << color[resilt[i]] << endl;

fout << "Вершина: " << i << " ---> Цвет:" << color[resilt[i]] << endl;

}

delete[] available;

delete[] resilt;

}

int main() {

system("color F0");

srand(time(NULL));

setlocale(0, "");

int z;

l:

system("cls");

cout << "Введите кол-во вершин графа: "; cin >> z;

Graph c(z);

/////////////////////////////////////////////////////////////////////////////////

set<int>\* adj = new set<int>[z];

string s;

while (true) {

system("cls");

cout << "Количество вершин: " << z

<< "\n Выберете способ построения графа: \n 1) Случайное заполнение \n"

<< " 2) Заполнить самому \n 3) Изменить кол-во вершин \n 4) Выход\n Введите: ";

cin >> s;

switch (atoi(s.c\_str()))

{

case 1:

system("cls");

int r;

cout << "Введите вероятность присутствия ребра(чем больше, тем выше вероятность): ";

cin >> r;

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

{

for (int j = i+1; j < z; j++)

{

if (1<=(rand() % r)) {

c.addEdgeMatrix(i, j);

addEdge(adj, i, j);

}

}

}

cout << "-----------------" << endl << "Матрица смежности: \n";

fout << endl << "-----------------" << endl << "Матрица смежности: \n";

c.toString();

cout << endl << "-----------------" << endl << "Ответ: \n";

fout << endl << "-----------------" << endl << "Ответ: \n";

GraphColoring(adj, z);

c.Memset();

system("pause");

break;

case 2:

system("cls");

cout << "Вводите по две вершины через пробел"<<endl

<< "Для выхода введите \*"<<endl

<<"Пример: \n 0 1\n 0 2\n ....\nВведите:\n";

while(true) {

string a, b;

cin >> a >> b;

if (a == "\*" || b == "\*")

break;

if (atoi(a.c\_str()) < z && atoi(b.c\_str()) < z) {

addEdge(adj, atoi(a.c\_str()), atoi(b.c\_str()));

c.addEdgeMatrix(atoi(a.c\_str()), atoi(b.c\_str()));

}

else {

cout << "Такой вершины не существует"<<endl;

}

}

cout << "-----------------" << endl << "Матрица смежности: \n";

fout << endl << "-----------------" << endl << "Матрица смежности: \n";

c.toString();

cout << endl << "-----------------" << endl << "Ответ: \n";

fout << endl << "-----------------" << endl << "Ответ: \n";

GraphColoring(adj, z);

c.Memset();

system("pause");

break;

case 3:

goto l;

break;

case 4:

exit(0);

break;

}

system("cls");

}

/////////////////////////////////////////////////////////////////////////////////

fout.close();

c.~Graph();

delete[] adj;

return 0;

}