

Министерство образования и науки Российской Федерации
Пензенский государственный университет
Факультет вычислительной техники

Пояснительная записка

К курсовому проектированию по курсу «Логика и основы
алгоритмизации в инженерных задачах»
на тему «Разработка алгоритма нахождения хроматического числа
графа»

сост.
28.12.22

Выполнил:
Студент группы 21ВВ3
Самохвалов Я. Д.

Принял:
д.т.н., профессор, зав. каф. ВТ Митрохин М. А.

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники
Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

« » 20

ЗАДАНИЕ

на курсовое проектирование по курсу

Логика и основы алгоритмизации в микроконтроллерах
Студенту Самонвалову Ярославу Дмитриевичу Группа 21ВВЗ
Тема проекта Разработка алгоритма нахождения кратчайшего числа графа

Исходные данные (технические требования) на проектирование

Разработка алгоритмов и программного обеспечения в соответствии с данными заданиями курсового проекта. Техническая записка должна содержать:

1. Постановка задачи;
2. Теоретическая часть задания;
3. Описание алгоритма постановки задачи;
4. Пример ручного расчета задачи и выводов (на конкретном участке работы алгоритма);
5. Описание самой программы;
6. Плесты;
7. Список литературы;
8. Листинг программы;
9. Результаты работы программы.

Объем работы по курсу

1. Расчетная часть

Ручной расчёт работы алгоритма.

2. Графическая часть

Схема алгоритма в формате блок-схем.

3. Экспериментальная часть

Тестирование программы;
Результаты работы программы на тестовых данных.

Срок выполнения проекта по разделам

- 1 Изучение теоретической части курсового к
- 2 Разработка алгоритмов программы к
- 3 Разработка программы к
- 4 Тестирование и завершение разработки программы к
- 5 Оформление пояснительной записки к
- 6
- 7
- 8

Дата выдачи задания "09" сентября 2022 г.

Дата защиты проекта "28" декабря 2022 г.

Руководитель Матрохин Максим Александрович

Задание получил "09" сентября

2022 г.

Студент Самойлов Ильяв Дмитриевич

Содержание

Реферат	7
Введение	8
1. Постановка задачи	9
2. Теоретическая часть задания	10
3. Описание алгоритма программы	13
4. Описание программы	17
5. Тестирование	25
6. Ручной расчет задачи	28
Заключение	29
Список литературы	30
Приложение А	31

Реферат

Отчет 38 страниц, 11 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ, РАСКРАСКА ГРАФА, ХРОМАТИЧЕСКОЕ ЧИСЛО ГРАФА.

Цель исследования – разработка программы, способной найти хроматическое число графа (наименьшее возможное количество цветов для раскраски графа).

В работе рассмотрены 2 алгоритма для нахождения хроматического числа графа: алгоритм Красновой А. Ю. и “жадный” алгоритм. Так же идёт сравнение этих двух алгоритмов, насколько они оптимальны и действенны. В чём сложность задачи по нахождению хроматического числа графа.

Введение

Одной из трудных задач теории графов является задача отыскания хроматического числа графа, то есть минимального числа цветов, необходимых для раскраски вершин графа. Предложены различные алгоритмы решения данной задачи, однако поиск эффективного алгоритма продолжается. Раскраска вершин позволяет моделировать многие проблемы планирования. В частности, при помощи алгоритма раскраски графа может быть решена задача составления расписания.

В рамках данной работы рассматривается вершинная раскраска графа, осуществляется анализ алгоритма вершинной раскраски графа, предложенный Красновой А. Ю. и предлагается альтернативный алгоритм раскраски графа.

В качестве среды разработки мною была выбрана среда Microsoft Visual Studio 2019, язык программирования – C++. Целью данной курсовой работы является разработка программы на языке C++, который является широко используемым. Именно с его помощью в данном курсовом проекте реализуется алгоритм Красновой и “жадный” алгоритм.

1. Постановка задачи

Требуется разработать программу, которая осуществит поиск кратчайших расстояний из одной точки до всех остальных.

Исходный граф в программе должен задаваться матрицей смежности, причём при генерации данных должны быть предусмотрены граничные условия. Программа должна работать так, чтобы пользователь вводил количество вершин для генерации матрицы смежности. После обработки этих данных на экран должна выводиться матрица смежности графа. Необходимо предусмотреть различные исходы алгоритмов, чтобы программа не выдавала ошибок и работала правильно. Устройство ввода – клавиатура и мышь.

2. Теоретическая часть задания

Здесь будет рассмотрена задача вершинной раскраски обыкновенного графа. Перед тем как приступить к рассмотрению задач напомним определение графа, которое будет использовано далее, без углубления в теорию графов. Под «графом» в данной работе будет пониматься обыкновенный неориентированный или ориентированный граф (рисунок 1). Стоит заметить, что большинство определений теории графов даются по-разному в различных источниках, но они просто демонстрируют разные подходы и в большинстве своём не противоречат друг другу.

- Граф – упорядоченная пара множеств (V, E) , где V – непустое множество вершин, а E – множество неупорядоченных пар вида (v_i, v_j) , называемых рёбрами, где v_i и v_j принадлежат множеству V .
- Вершины v_i и v_j графа $G = (V, E)$ называются смежными, если они соединены ребром, то есть если в E существует ребро (v_i, v_j) .
- Одним из способов задания графа является матрица смежности – матрица A размером $[N \times N]$, где N – количество вершин, в которой элемент $a_{ij} = 1$, если в графе вершины i и j смежны, и 0 в противном случае.

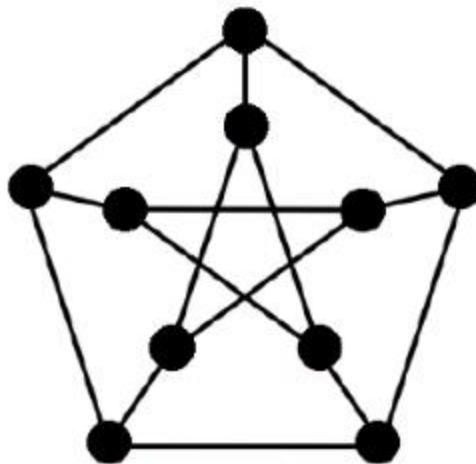


Рисунок 1 - Пример графа

Теперь рассмотрим подробнее задачу раскраски графа. Данная задача является одной из основных в теории графов, потому что входит в класс NP-полных и к ней сводятся многие другие задачи этой области. Раскраска графа

может быть вершинной, рёберной и тотальной, однако все три варианта задачи можно свести друг к другу, поэтому рассмотрим только вершинную раскраску графа.

«Раскраской графа называется такое приписывание цветов его вершинам, что никакие две смежные вершины не получают одинакового цвета.»

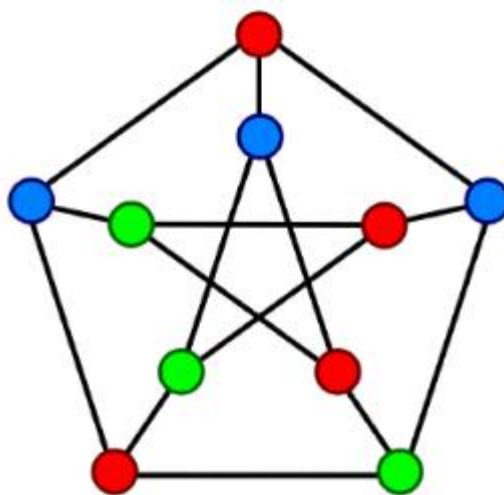


Рисунок 2 - Вариант раскраски графа на рисунке 1.

Задача же раскраски состоит в нахождении минимального количества цветов для раскраски графа. Такое решение далее будем называть оптимальным. На рисунке 2 показана оптимальная раскраска графа, приведённого на рисунке 1. Задача не имеет известного единого алгоритма для поиска решения за полиномиальное время. Существуют варианты таких решений только для определённых типов графов. Единственным подходом к нахождению оптимального решения является перебор вариантов. Однако предлагаются разные подходы и алгоритмы для поиска неточного, приближённого решения, которое найти гораздо проще оптимального.

Приведем кратко принцип работы некоторых из них.

- Алгоритмы последовательного перебора. Такие алгоритмы последовательно отделяют группы цветов. На каждом этапе создаётся группа вершин, в которую пытаются включить нераспределённые вершины, пока все они не будут распределены по цветам.

- Жадные алгоритмы. Жадные алгоритмы упорядочивают вершины по какому-либо правилу (чаще всего встречаются примеры, где используют степени вершин) и последовательно присваивают им цвета, в которые не были окрашены смежные с ними вершины.
- Алгоритм с использованием битовых операций. Алгоритм похож на первый, но работает с матрицами, используя битовые операции. Такой подход позволяет сократить время работы в несколько раз.

3. Описание алгоритма программы

В данной работе, я использовал 2 алгоритмы для поиска хроматического числа графа, следовательно, нужно подробно изучать оба.

3.1. Алгоритм раскраски графа Красновой А. Ю.

В кандидатской диссертации Красновой А. Ю. «Матрицы инцидентий и раскраски графа» предложен алгоритм раскраски обыкновенного графа, основанный на рассмотрении матрицы инцидентий и последовательном выделении непустых подмножеств множества вершин графа, включающий процедуру перемешивания, позволяющую приблизить число красок к хроматическому числу.

Алгоритм представляет собой вариант алгоритма последовательного перебора, но с перемешиваниями и повторениями. Такое усовершенствование помогает уменьшить зависимость результата от начальной нумерации вершин. Алгоритм использует матрицу инцидентий, а для сохранения исходной нумерации вершин к ней приписывается столбец I с номерами. Ниже приведен алгоритм.

1. Пусть граф $G(V, X)$ задан матрицей смежности A . По ней строим матрицу инцидентий B . Применяя к матрице алгоритм последовательного раскрашивания вершин, получаем раскраску в r цветов и матрицу перестановок P_0 такую, что $B_t = P_0 B$.
2. Задаём число перемешиваний N .
3. Выполняем n итераций процедуры перемешивания. При этом полагаем, что $B_t^{(0)} = P_0 B$ и $I_t^{(0)} = P_0 I$.

Тогда k -тая итерация будет происходить так:

1. Строим матрицу инцидентий $B_s^{(k)} = Q B_t^{(k-1)}$.
2. Применяем к матрице $B_s^{(k)}$ алгоритм последовательного раскрашивания вершин, причём для построения «расширенной» матрицы инцидентий $B_s^{(k)}$ приписываем слева к ней столбец $I_s^{(k)} = Q I_t^{(k-1)}$. Получаем раскраску в r_k

цветов и матрицу перестановок P_k такую, что $B_t^{(k)} = P_k B_s^{(k)}$. При этом $I_t^{(k)} = P_k I_s^{(k)}$.

При реализации алгоритма было принято отказаться от использования матрицы инцидентий, поскольку матрица смежности графа позволяет уменьшить объём используемой оперативной памяти.

Реализация алгоритма приведена в приложении А (Файл Kras.h).

3.2. Жадный алгоритм раскраски графа.

Далее рассмотрим новый алгоритм раскраски графа, построенный как жадный алгоритм. В большинстве случаев, как уже отмечено ранее, жадные алгоритмы упорядочивают вершины по их степени (или относительной степени, то есть степени без учёта окрашенных вершин). В этом есть смысл, потому что окрашивание таких вершин приносит наибольшее количество информации в систему: с окраской таких вершин мы не можем окрасить большее количество цветов в цвет, выбранный для этой вершины. Для разработанного алгоритма было решено использовать степень вершин только как второстепенный фактор. Первичным же фактором для сравнения было выбрано количество цветов, в которые вершину окрасить уже нельзя. Во многих случаях такой подход формирует полный пул цветов, потому что рассматривает прежде всего вершины, для которых приходится добавлять цвет. Кроме того, будем упорядочивать не только вершины, но и цвета, чтобы сделать распределение вершин, а соответственно и нагрузки, более равномерными.

В алгоритме можно выделить следующие этапы работы:

1. Считать матрицу смежности.
2. Инициализировать вектор цветов (изначально пустой), содержащий вектора вершин, окрашенных в соответствующий цвет. Сюда добавляется окрашенная вершина.

3. Сформировать вектор вершин. Для хранения информации о вершинах создан класс Vertex, содержащий номер вершины в графе, её степень и вектор цветов, в которые её нельзя окрасить.
4. Упорядочить вектор вершин по приоритетам. Приоритет вершины больше, если её нельзя окрасить в большее количество цветов, иначе приоритет по степени.
5. Подобрать цвет для вершины, проверяя с наименее распространённых. Все доступные цвета хранятся в векторе, упорядоченном по количеству вершин, окрашенных в данный цвет. Проверяется вхождение рассматриваемого цвета в вектор цветов, в которые нельзя покрасить вершину.

5.1. Если цвет найден

5.1.1. Окрасить вершину.

5.1.2. Пометить цвет как запрещённый для всех соседей. Для этого по соответствующей строке матрицы смежности находятся все соседние вершины и в вектор запрещённых для них цветов добавляется текущий, если его ещё нет.

5.1.3. Пересортировать вершины и цвета. Нет необходимости рассматривать соответствующие векторы целиком.

5.1.3.1. После окрашивания вершины увеличилось количество вершин данного цвета, поэтому проверяется соседний с ним в векторе цветов и, пока необходимо, цвета меняются местами.

5.1.3.2. Каждая соседняя вершина получает дополнительный запрещённый цвет – её приоритет увеличивается, поэтому проверяются соседние вершины в списке вершин и, пока необходимо, вершины меняются местами.

5.1.4. Исключить вершину. В матрице смежности обнуляются соответствующие строка и столбец, чтобы эта вершина не рассматривалась как соседняя, и вершина удаляется из вектора вершин.

5.2. Если цвет не найден, добавить цвет и выполнить п. 5.1.

6. Повторять п. 5 пока не будут окрашены все вершины (пока вектор вершин не опустеет).

Полный исходный код алгоритма можно найти в приложении А (Файл Greed.h).

4. Описание программы

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

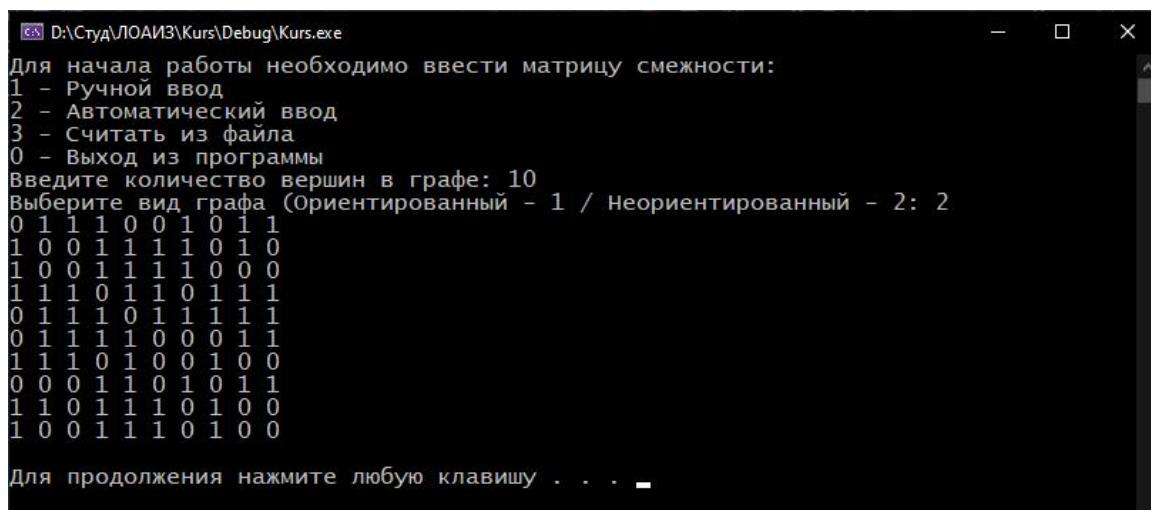
Проект был создан в виде консольного приложения Win32 (Visual C++).

Данная программа является многомодульной, поскольку состоит из многих функций, а также из 2 заглавных файлов и одного главного (Kurs.cpp, Greed.h, Kras.h).

Работа программы начинается с вывода заголовка, а затем запроса генерации матрицы. Если пользователь выбрал сгенерировать случайную матрицу, то на экран выводится запрос на количество вершин в графе, далее генерируется сам граф, программа его отобразит и запросит следующий выбор.

Если же пользователь выбрал считать готовую матрицу из файла, то сначала, она считывает количество вершин в графе, а затем и сам граф.

Ниже можно увидеть оформление начального запроса и дальнейшие действия с ним. (Рисунки 3, 4).



```
D:\Студ\ЛОАИЗ\Kurs\Debug\Kurs.exe
Для начала работы необходимо ввести матрицу смежности:
1 - Ручной ввод
2 - Автоматический ввод
3 - Считать из файла
0 - Выход из программы
Введите количество вершин в графе: 10
Выберите вид графа (Ориентированный - 1 / Неориентированный - 2: 2
0 1 1 1 0 0 1 0 1 1
1 0 0 1 1 1 1 0 1 0
1 0 0 1 1 1 1 0 0 0
1 1 1 0 1 1 0 1 1 1
0 1 1 1 0 1 1 1 1 1
0 1 1 1 1 0 0 0 1 1
1 1 1 0 1 0 0 1 0 0
0 0 0 1 1 0 1 0 1 1
1 1 0 1 1 1 0 1 0 0
1 0 0 1 1 1 0 1 0 0

Для продолжения нажмите любую клавишу . . .
```

Рисунок 3 - Пример случайной генерации матрицы смежности

```
D:\Студ\ЛОАИЗ\Kurs\Debug\Kurs.exe
Для начала работы необходимо ввести матрицу смежности:
1 - Ручной ввод
2 - Автоматический ввод
3 - Считать из файла
0 - Выход из программы
Введите имя файла: kkk.txt
0 1 0 1
1 0 1 0
0 1 0 1
1 0 1 0
Для продолжения нажмите любую клавишу . . .
```

Рисунок 4 - Граф взят из файла kkk.txt

Затем предлагают выбрать какой алгоритм использовать для нахождения хроматического числа графа. (Рисунок 5)

```
D:\Студ\ЛОАИЗ\Kurs\Debug\Kurs.exe
Для начала работы необходимо ввести матрицу смежности:
1 - Ручной ввод
2 - Автоматический ввод
3 - Считать из файла
0 - Выход из программы
Введите количество вершин в графе: 6
Выберите вид графа (Ориентированный - 1 / Неориентированный - 2: 2
0 0 1 1 0 0
0 0 1 0 0 1
1 1 0 0 1 0
1 0 0 0 0 0
0 0 1 0 0 0
0 1 0 0 0 0
Для продолжения нажмите любую клавишу . . .
Выберите какой алгоритм вы хотите использовать:
1 - Нахождение хроматического графа через алгоритм Красновой
2 - Нахождение хроматического графа с помощью 'Жадного' алгоритма
3 - Нахождение хроматического графа, используя оба алгоритма (для сравнения)
0 - Выход из программы
```

Рисунок 5 - Выбор алгоритма

Ну и после выполнения поиска, нам предлагаю выводить данные в файл или на экран. (Рисунок 6)

```
Выберите какой алгоритм вы хотите использовать:
1 - Нахождение хроматического графа через алгоритм Красновой
2 - Нахождение хроматического графа с помощью 'Жадного' алгоритма
3 - Нахождение хроматического графа, используя оба алгоритма (для сравнения)
0 - Выход из программы
Введите количество повторений данного алгоритма (для повышения точности): 100
Хотите ли вывести результат в файл?(Y/N): N
Количество вершин: 6
0 1 1 1 1 0
1 0 0 1 0 0
1 0 0 1 1 1
1 1 1 0 0 1
1 0 1 0 0 0
0 0 1 1 0 0
Хроматическое число графа: 3
D:\Студ\ЛОАИЗ\Kurs\Debug\Kurs.exe (процесс 11636) завершил работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Автоматически закрывать консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно.
```

Рисунок 6 - Результат выполнения алгоритма

Далее более подробно о самой программе. В главном файле Kurs.cpp записаны все функции для вывода информации в консоль, а также ввода необходимых данных. Пользователь сам выбирает нужный ему пункт меню, и затем в зависимости от его выбора, программа разделяется на несколько вариантов развития.

Меню содержатся, как отдельные функция. Они нужны для фиксации выбора пользователя, и дальнейшего выбора решения.

```
int _stateMenu1, _stateMenu2;
void Menu1()
{
    cout << "Для начала работы необходимо ввести матрицу смежности: " << endl
         << "1 - Ручной ввод" << endl
         << "2 - Автоматический ввод" << endl
         << "3 - Считать из файла" << endl
         << "0 - Выход из программы" << endl;
    _stateMenu1 = _getch() - 48;
}

void Menu2()
{
    cout << "Выберите какой алгоритм вы хотите использовать: " << endl
         << "1 - Нахождение хроматического графа через алгоритм Красновой" << endl
         << "2 - Нахождение хроматического графа с помощью 'Жадного' алгоритма" << endl
         << "3 - Нахождение хроматического графа, используя оба алгоритма (для сравнения)" << endl
         << "0 - Выход из программы" << endl;
    _stateMenu2 = _getch() - 48;
}
```

Также у нас в главной программе находятся функции для вывода матрицы на экран пользователя.

```
void PrintGraph(char** Graph, int vertexNumber)
{
    for (int i = 0; i < vertexNumber; i++)
    {
        for (int j = 0; j < vertexNumber; j++)
        {
            cout << Graph[i][j] << " ";
        }
        cout << endl;
    }
    cout << endl;
}
```

А еще содержится функция вывода результата в консоль или файл, где пользователь также должен выбрать тот способ, который он желает.

```
void PrintFile(int vertexNumber, char** graph, int gr)
{
    ofstream fout;
    char filename[256];
    char ans;
    while (true)
    {
        cout << "Хотите ли вывести результат в файл?(Y/N): ";
        cin >> ans;
        cout << endl;
        if (ans == 'N' || ans == 'n')
```

```

{
    cout << "Количество вершин: " << vertexNumber << endl;
    for (int i = 0; i < vertexNumber; i++)
    {
        for (int j = 0; j < vertexNumber; j++)
        {
            cout << graph[i][j] << " ";
        }
        cout << endl;
    }
    cout << "Хроматическое число графа: " << gr << endl;
    exit(EXIT_SUCCESS);
}
else if (ans == 'Y' || ans == 'y')
{
    cout << "Введите имя файла для вывода: ";
    cin >> filenameeo;
    fout.open(filenameeo);

    fout << "Количество вершин: " << vertexNumber << endl;
    for (int i = 0; i < vertexNumber; i++)
    {
        for (int j = 0; j < vertexNumber; j++)
        {
            fout << graph[i][j] << " ";
        }
        fout << endl;
    }
    fout << "Хроматическое число графа: " << gr;

    fout.close();
    cout << "Данные записаны в файл" << endl;
    exit(EXIT_SUCCESS);
}
else
{
    continue;
}
}
}

```

Далее подробно разберем сами алгоритмы работы программы. Для начала рассмотрим файл Greed.h.

В начале мы создаем класс, который будет хранить все данные об одной из вершин (её номер, степень и цвета, не доступные для покраски).

```

class Vertex
{
public:
    int number;
    int degree;
    vector<int> lockedColors;

    Vertex()
    {
        number = 0;
        degree = 0;
        lockedColors = vector<int>();
    };

    Vertex(char** graph, int vertexNumber, int index)
    {
        number = index;
        degree = 0;
        for (int i = 0; i < vertexNumber; ++i)
            if (graph[index][i] == '1')
                ++degree;
        lockedColors = vector<int>();
    }
}

```

```
};
```

Затем уже приступаем к созданию необходимых переменных.

```
char** currGraph = curGraph; //матрица смежности
int vertexNumber = vertexNumbe; //кол-во вершин в графе
vector<Vertex> vertexes; //создаем вектор вершин, представленных в виде класса
vertexes.reserve(vertexNumber); //выделяем память вектору для вершин
for (int i = 0; i < vertexNumber; ++i) //представляем вершины в виде класса и добавляем их
    vertexes.push_back(Vertex(currGraph, vertexNumber, i));

vector<int> newIndexes; //вектор номеров вершин по порядку
newIndexes.reserve(vertexNumber);
for (int i = 0; i < vertexNumber; ++i)
    newIndexes.push_back(i);

vector<vector<int>> colors; //вектор цветов
vector<int> colorFrequency; //вектор вершин окрашенных в соответствующий цвет
addColor(colors, colorFrequency); //добавляем первый цвет

quickSort(0, vertexNumber - 1, vertexes, newIndexes); // сортируем все вершины по порядку

bool islocked = false; //переменная для проверки доступности цвета
int currentColor = 0;
bool toAdd = true; //переменная для проверки необходимости добавлять цвет
```

А теперь начинаем обход по алгоритму пока у нас есть неокрашенная вершина. А затем начинаем обход по каждой вершине и ищем свободен ли данный цвет для этой вершины. И дальше идёт 2 варианта развития.

```
while (!vertexes.empty()) //пока класс не пустой
{
    for (int i = colorFrequency.size() - 1; i >= 0; --i) //проходим по вершинам
    {
        currentColor = colorFrequency[i]; //берем свободную вершину
        //поиск цвета
        islocked = false;
        for (int j = 0; j < vertexes.back().lockedColors.size(); ++j)
        {
            if (currentColor == vertexes.back().lockedColors[j])
                islocked = true;
        }
    }
}
```

Если найден цвет. Если цвет доступен для данной вершины, то мы записываем номер вершины в вектор цветов, а затем корректируем связанные вершины, то есть по соответствующей строке матрицы смежности находим все соседние вершины и в вектор запрещенных для них цветов добавляется текущий, если его нет. Каждая соседняя вершина получает дополнительный запрещённый цвет — её приоритет увеличивается, поэтому проверяются соседние вершины в списке вершин и, пока необходимо, вершины меняются местами.

```
if (!islocked)
{
    //добавляем вершину в список
    colors[currentColor].push_back(vertexes.back().number);
    ++colors[currentColor][0];
    //коррекция связанных вершин
    for (int n = 0; n < vertexNumber; ++n)
    {

```

```

        if (currGraph[vertexes.back().number][n] == '1')
        {
            //Проверка необходимости добавления заблокированного цвет (без
повтора)
            if (vertexes[newIndexes[n]].lockedColors.size() == 0)
                toAdd = true;
            else
            {
                toAdd = true;
                for (int j = 0; j <
vertexes[newIndexes[n]].lockedColors.size(); ++j)
                {
                    if (currentColor ==
vertexes[newIndexes[n]].lockedColors[j])
                    {
                        toAdd = false;
                        break;
                    }
                }
            }
            //Добавление данного цвета в заблокированные для вершины
            if (toAdd)
            {
                vertexes[newIndexes[n]].lockedColors.push_back(currentColor);
                //Пересортировка вершин по приоритету
                while (newIndexes[n] + 2 < vertexes.size() &&
vertexes[newIndexes[n]] > vertexes[newIndexes[n] + 1])
                {
                    swap(vertexes[newIndexes[n]],
vertexes[newIndexes[n] + 1]);
                    swap(newIndexes[vertexes[newIndexes[n]].number],
newIndexes[vertexes[newIndexes[n] + 1].number]);
                }
            }
        }
    }
}

```

Так же после окрашивания вершины увеличилось количество вершин данного цвета, поэтому проверяется соседний с ним в векторе цветов и, пока необходимо, цвета меняются местами. Исключаем вершину. В матрице смежности обнуляются соответствующие строка и столбец, чтобы эта вершина не рассматривалась как соседняя, и вершина удаляется из вектора вершин.

```

//Пересортировка цветов
int k = 0;
while (k < i)
{
    if (colors[colorFrequency[i - k]][0] > colors[colorFrequency[i - k -
1]][0])
    {
        swap(colorFrequency[i - k], colorFrequency[i - k - 1]);
        ++k;
    }
    else break;
}

//Удаление вершины, которая была уже окрашена, чтобы она не считалась соседней
for (int m = 0; m < vertexNumber; ++m)
{
    currGraph[m][vertexes.back().number] = '0';
    currGraph[vertexes.back().number][m] = '0';
}
vertexes.pop_back();
break;
}

```

Если не найден цвет. Мы его добавляем и выполняем алгоритм заново.

```
//Если нет свободного цвета, то добавляем новый
if (isLocked && i == 0)
{
    addColor(colors, colorFrequency);
    break;
}
```

На выход идёт количество цветов, полученное в итоге:

```
return colors.size();
```

Теперь рассмотрим второй файл Kras.h. Первое, что мы делаем, конечно же, создаем необходимые нам переменные.

```
int bestGroupCount = INT_MAX; //лучшее количество цветов
vector<int> bestGroups; //лучший вектор цветов для каждой вершины

vector<int> groups(vertexNumber, -1); //вектор цветов для каждой вершины

int currGroup = -1; //текущий цвет
int finishedVertexes = 0; //количество покрашенных вершин
vector<int> currGroupMembers; //вектор вершин текущего цвета
bool is_fit = true; //условие для окраски
```

Затем начинаются наши rounds количество итераций. Сначала обнуляем все наши переменные, используемые только в одной итерации. Затем начинаем алгоритм, пока все вершины не будут покрашены. Берем первый цвет и начинаем обход по вершинам. Если вершина окрашена, то берем следующую. Иначе, проверяем есть ли соединение с другой вершиной того же цвета, если есть, то берем следующую вершину. Иначе, мы присваиваем вершине текущий цвет, добавляем +1 к счетчику покрашенных вершин, и добавляем в вектор вершин данного цвета эту вершину, и идем к следующей итерации. После прохода по всем вершинам, проверяем все ли вершины покрашены в цвет, если нет, то всё повторяем, но уже со следующим цветом.

```
for (int round = 0; round < rounds; ++round)
{
    //Перезапуск
    groups.assign(vertexNumber, -1); currGroup = -1; finishedVertexes = 0; is_fit = true;
    while (finishedVertexes < vertexNumber)
    {
        ++currGroup;
        currGroupMembers.clear();
        is_fit = true;
        for (int i = 0; i < vertexNumber; ++i, is_fit = true)
        {
            if (groups[i] != -1) //Уже назначенный
                continue;
            for (int j = 0; j < currGroupMembers.size(); ++j)
                if (currGraph[i][currGroupMembers[j]] == '1') //соединение с другой
                    is_fit = false;
        }
        currGroupMembers.push_back(i);
        finishedVertexes++;
    }
}
```

```

        if (is_fit)
        {
            groups[i] = currGroup;
            ++finishedVertexes;
            currGroupMembers.push_back(i);
        }
    }

```

Затем мы проверяем, улучшился ли результат и записываем его, если да. А затем перемешиваем граф, и переходим к следующей итерации.

```

//улучшение алгоритма?
if (currGroup + 1 < bestGroupCount)
{
    bestGroupCount = currGroup + 1;
    bestGroups = groups;
}

//перемешиваем граф
currGraph = mix(vertexNumber, currGraph, groups, currGroup + 1);
}

```

Вот так и работает алгоритм, пока не добьется наилучшего результата. На выход идёт лучшее количество используемых цветов.

```

return bestGroupCount;

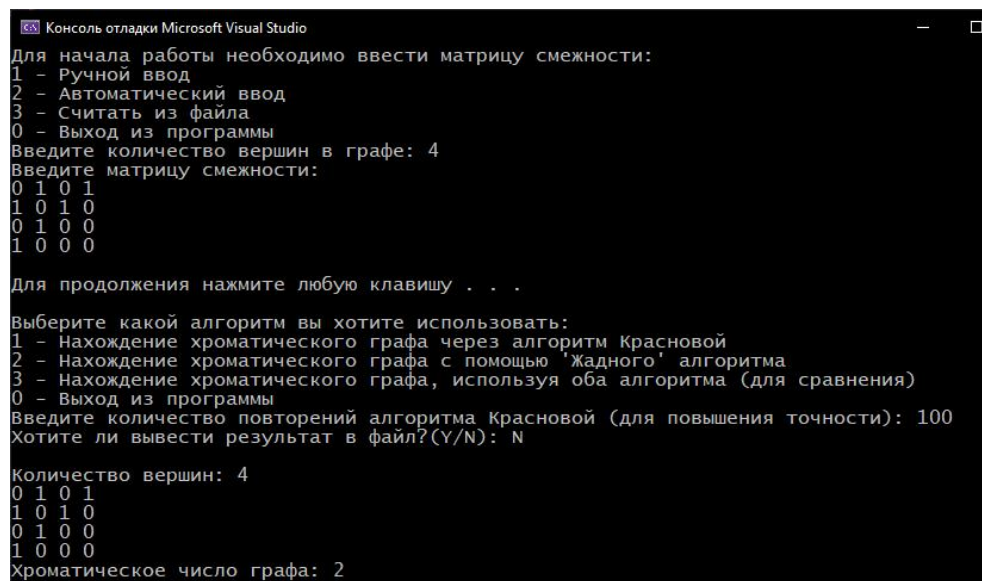
```

5. 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 (Рисунки 7, 8, 9, 10) тестирования программы при вводе пользователем различных количеств вершин.



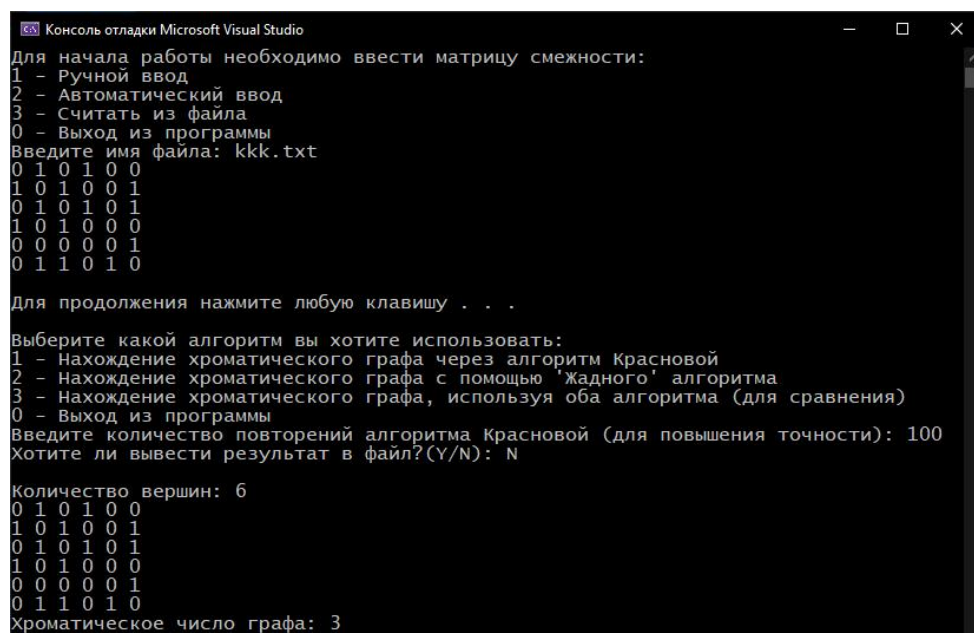
```
Консоль отладки Microsoft Visual Studio
для начала работы необходимо ввести матрицу смежности:
1 - Ручной ввод
2 - Автоматический ввод
3 - Считать из файла
0 - Выход из программы
Введите количество вершин в графе: 4
Введите матрицу смежности:
0 1 0 1
1 0 1 0
0 1 0 0
1 0 0 0

Для продолжения нажмите любую клавишу . . .

Выберите какой алгоритм вы хотите использовать:
1 - Нахождение хроматического графа через алгоритм Красновой
2 - Нахождение хроматического графа с помощью 'Жадного' алгоритма
3 - Нахождение хроматического графа, используя оба алгоритма (для сравнения)
0 - Выход из программы
Введите количество повторений алгоритма Красновой (для повышения точности): 100
Хотите ли вывести результат в файл? (Y/N): N

Количество вершин: 4
0 1 0 1
1 0 1 0
0 1 0 0
1 0 0 0
Хроматическое число графа: 2
```

Рисунок 7 - Результат работы с 4 вершинами вводимого с клавиатуры графа



```
Консоль отладки Microsoft Visual Studio
для начала работы необходимо ввести матрицу смежности:
1 - Ручной ввод
2 - Автоматический ввод
3 - Считать из файла
0 - Выход из программы
Введите имя файла: kkk.txt
0 1 0 1 0 0
1 0 1 0 0 1
0 1 0 1 0 1
1 0 1 0 0 0
0 0 0 0 0 1
0 1 1 0 1 0

Для продолжения нажмите любую клавишу . . .

Выберите какой алгоритм вы хотите использовать:
1 - Нахождение хроматического графа через алгоритм Красновой
2 - Нахождение хроматического графа с помощью 'Жадного' алгоритма
3 - Нахождение хроматического графа, используя оба алгоритма (для сравнения)
0 - Выход из программы
Введите количество повторений алгоритма Красновой (для повышения точности): 100
Хотите ли вывести результат в файл? (Y/N): N

Количество вершин: 6
0 1 0 1 0 0
1 0 1 0 0 1
0 1 0 1 0 1
1 0 1 0 0 0
0 0 0 0 0 1
0 1 1 0 1 0
Хроматическое число графа: 3
```

Рисунок 8 - Результат работы с 6 вершинами графа вводимого через файл

```
Консоль отладки Microsoft Visual Studio
Для начала работы необходимо ввести матрицу смежности:
1 - Ручной ввод
2 - Автоматический ввод
3 - Считать из файла
0 - Выход из программы
Введите количество вершин в графе: 10
Выберите вид графа (Ориентированный - 1 / Неориентированный - 2): 2
0 0 1 1 1 1 1 0 0 0
0 0 0 0 0 1 1 1 1 1
1 0 0 1 1 1 0 1 0 1
1 0 1 0 1 0 0 0 1 0
1 0 1 1 0 1 1 0 0 1
1 1 1 0 1 0 1 1 0 1
1 1 0 0 1 1 0 0 0 1
0 1 1 0 0 1 0 0 1 1
0 1 0 1 0 0 0 1 0 0
0 1 1 0 1 1 1 1 0 0

Для продолжения нажмите любую клавишу . . .

Выберите какой алгоритм вы хотите использовать:
1 - Нахождение хроматического графа через алгоритм Красновой
2 - Нахождение хроматического графа с помощью 'Жадного' алгоритма
3 - Нахождение хроматического графа, используя оба алгоритма (для сравнения)
0 - Выход из программы
Введите количество повторений алгоритма Красновой (для повышения точности): 100
Хотите ли вывести результат в файл?(Y/N): N

Количество вершин: 10
0 0 1 1 1 1 1 0 0 0
0 0 0 0 0 1 1 1 1 1
1 0 0 1 1 1 0 1 0 1
1 0 1 0 1 0 0 0 1 0
1 0 1 1 0 1 1 0 0 1
1 1 1 0 1 0 1 1 0 1
1 1 0 0 1 1 0 0 0 1
0 1 1 0 0 1 0 0 1 1
0 1 0 1 0 0 0 1 0 0
0 1 1 0 1 1 1 1 0 0

Хроматическое число графа: 5
```

Рисунок 9 - Результат работы с 10 вершинами случайного неориентированного графа

```
Выборить Консоль отладки Microsoft Visual Studio
Для начала работы необходимо ввести матрицу смежности:
1 - Ручной ввод
2 - Автоматический ввод
3 - Считать из файла
0 - Выход из программы
Введите количество вершин в графе: 10
Выберите вид графа (Ориентированный - 1 / Неориентированный - 2): 1
0 0 1 1 1 0 1 0 0 1
0 0 0 1 1 1 0 1 1 0
0 0 0 0 0 0 0 0 1 1
0 0 1 0 1 1 1 0 0 0
1 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0 1 1
0 1 0 0 1 1 0 1 0 1
1 1 0 1 1 1 0 0 1 0
1 0 0 0 1 0 1 0 0 1
0 1 1 0 0 1 1 0 1 0

Для продолжения нажмите любую клавишу . . .

Выберите какой алгоритм вы хотите использовать:
1 - Нахождение хроматического графа через алгоритм Красновой
2 - Нахождение хроматического графа с помощью 'Жадного' алгоритма
3 - Нахождение хроматического графа, используя оба алгоритма (для сравнения)
0 - Выход из программы
Введите количество повторений алгоритма Красновой (для повышения точности): 100
Хотите ли вывести результат в файл?(Y/N): N

Количество вершин: 10
0 0 1 1 1 0 1 0 0 1
0 0 0 1 1 1 0 1 1 0
0 0 0 0 0 0 0 0 1 1
0 0 1 0 1 1 1 0 0 0
1 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0 1 1
0 1 0 0 1 1 0 1 0 1
1 1 0 1 1 1 0 0 1 0
1 0 0 0 1 0 1 0 0 1
0 1 1 0 0 1 1 0 1 0

Хроматическое число графа: 3
```

Рисунок 10 - Результат работы с 10 вершинами случайного ориентированного графа

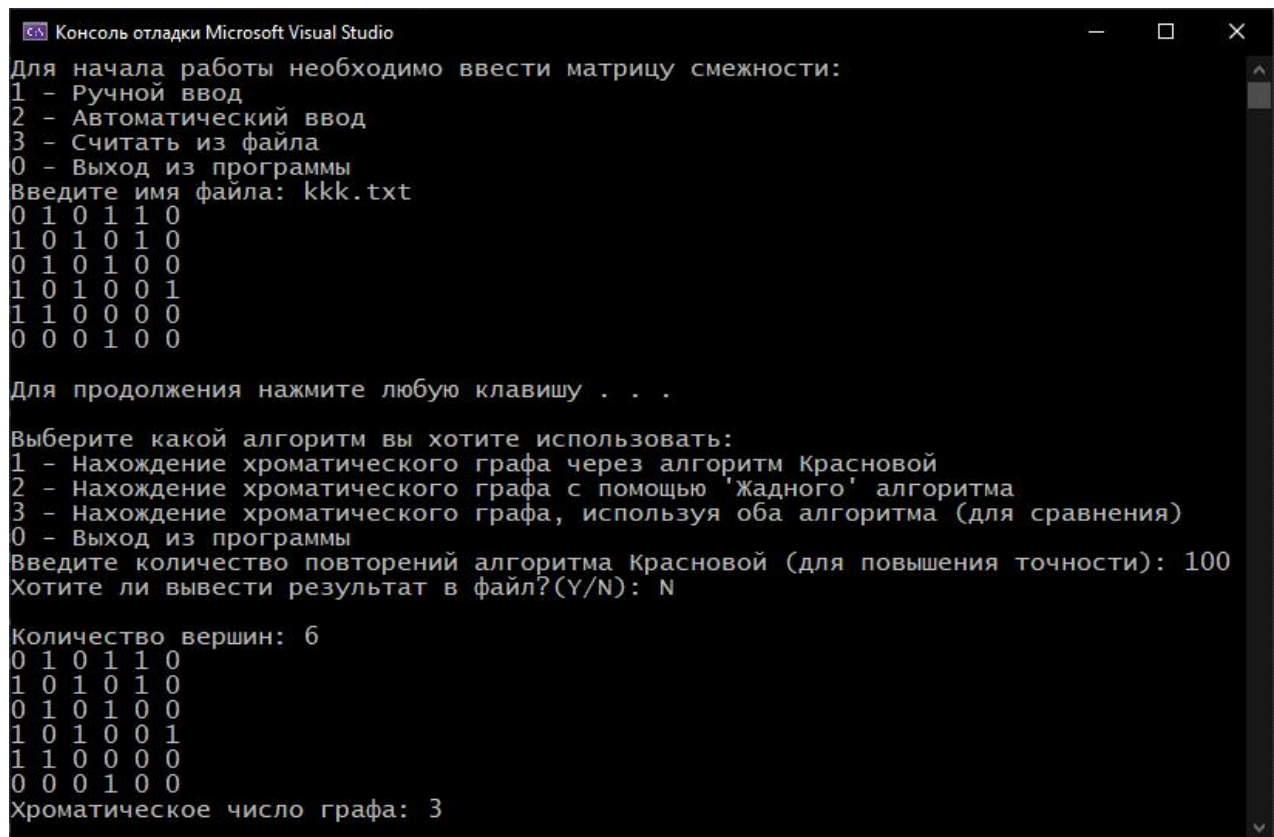
Таблица 1 – Описание поведения программы при тестировании

Описание теста	Ожидаемый результат	Полученный результат
Запуск программы	Вывод сообщения о выборе: ввести матрицу с клавиатуры, автоматически или из файла	Верно
Выбор генерации матрицы	Вывод сообщения о количестве вершин в графе и вывод самой матрицы	Верно
Выбор алгоритмы работы программы	Вывод сообщения о выборе: использование алгоритма Красновой, “жадного” или обоих. А также корректная их работа	Верно
Выбор вывода результатов программы	Корректная работа алгоритма и вывод сообщения о выборе: вывести в консоль или в файл результат работы программы	Верно

В результате тестирования было выявлено, что программа успешно проверяет данные на соответствие необходимым требованиям.

6. Ручной расчет задачи

Проведем проверку программы посредством ручных вычислений на примере графа с 6 вершинами на рисунке 11.



```
Консоль отладки Microsoft Visual Studio
Для начала работы необходимо ввести матрицу смежности:
1 - Ручной ввод
2 - Автоматический ввод
3 - Считать из файла
0 - Выход из программы
Введите имя файла: kkk.txt
0 1 0 1 1 0
1 0 1 0 1 0
0 1 0 1 0 0
1 0 1 0 0 1
1 1 0 0 0 0
0 0 0 1 0 0

Для продолжения нажмите любую клавишу . . .

Выберите какой алгоритм вы хотите использовать:
1 - Нахождение хроматического графа через алгоритм Красновой
2 - Нахождение хроматического графа с помощью 'Жадного' алгоритма
3 - Нахождение хроматического графа, используя оба алгоритма (для сравнения)
0 - Выход из программы
Введите количество повторений алгоритма Красновой (для повышения точности): 100
Хотите ли вывести результат в файл?(Y/N): N

Количество вершин: 6
0 1 0 1 1 0
1 0 1 0 1 0
0 1 0 1 0 0
1 0 1 0 0 1
1 1 0 0 0 0
0 0 0 1 0 0
Хроматическое число графа: 3
```

Рисунок 11 - Результат работы программы

Покрасим вершину 1 в цвет №1, тогда в цвет №1 можно покрасить ещё 2 вершины – это 3 и 6.

Затем берем следующую краску и следующую вершину. Следовательно, вершина 2 – краска №2, тогда мы можем покрасить вершину 4, несмежную ей так же во 2 краску. И оставшаяся 5 вершина занимает 3 краску. Никаких других менее экономных вариантов раскраски нет.

В результате программы написано верное число, это означает, что программа работает исправно.

Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм нахождения хроматического числа 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 графов. Приобретены навыки по осуществлению алгоритма Красновой и “жадного” алгоритма. Углублены знания языка программирования C++.

Недостатком разработанной программы является примитивный пользовательский интерфейс. Потому что программа работает в консольном режиме, не добавляющем к сложности языка сложность программного оконного интерфейса.

Программа имеет небольшой, но достаточный для использования функционал возможностей.

Программа попадает в категорию NP-полных, поэтому результат может отличаться от ручных расчётов.

Список литературы

1. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: Построение и анализ - М.: МЦНМО, 2001. - 960 с
2. Кристофидес Н. «Теория графов. Алгоритмический подход» - Мир, 1978
3. Герберт Шилдт «Полный справочник по C++» - Вильямс, 2006
4. З. Оре О. Графы и их применение: Пер. с англ. 1965. 176 с
5. Кандидатской диссертации Красновой А. Ю. «Матрицы инцидентий и раскраски графа»
6. Харви Дейтел, Пол Дейтел. Как программировать на C/C++. 2009 г.
7. www.wikipedia.com
8. Дискретная математика: Методические указания к лабораторным работам / Рязанская государственная радиотехническая академия; Сост. А.М. Гостин. Рязань, 2004. 24 с.

Приложение А

Листинг программы

Файл Kurs.cpp

```
#include "Kras.h"
#include "Greed.h"

using namespace std;

int _stateMenu1, _stateMenu2;
void Menu1()
{
    cout << "Для начала работы необходимо ввести матрицу смежности: " << endl
        << "1 - Ручной ввод" << endl
        << "2 - Автоматический ввод" << endl
        << "3 - Читать из файла" << endl
        << "0 - Выход из программы" << endl;
    _stateMenu1 = _getch() - 48;
}

void Menu2()
{
    cout << "Выберите какой алгоритм вы хотите использовать: " << endl
        << "1 - Нахождение хроматического графа через алгоритм Красновой" << endl
        << "2 - Нахождение хроматического графа с помощью 'Жадного' алгоритма" << endl
        << "3 - Нахождение хроматического графа, используя оба алгоритма (для сравнения)" <<
endl
        << "0 - Выход из программы" << endl;
    _stateMenu2 = _getch() - 48;
}

void PrintGraph(char** Graph, int vertexNumber)
{
    for (int i = 0; i < vertexNumber; i++)
    {
        for (int j = 0; j < vertexNumber; j++)
        {
            cout << Graph[i][j] << " ";
        }
        cout << endl;
    }
    cout << endl;
}

void PrintFile(int vertexNumber, char** graph, int gr)
{
    ofstream fout;
    char filenameo[256];
    char ans;
    while (true)
    {
        cout << "Хотите ли вывести результат в файл?(Y/N): ";
        cin >> ans;
        cout << endl;
        if (ans == 'N' || ans == 'n')
        {
            cout << "Количество вершин: " << vertexNumber << endl;
            for (int i = 0; i < vertexNumber; i++)
            {
                for (int j = 0; j < vertexNumber; j++)
                {
                    cout << graph[i][j] << " ";
                }
                cout << endl;
            }
        }
    }
}
```

```

        cout << "Хроматическое число графа: " << gr << endl;
        exit(EXIT_SUCCESS);
    }
    else if (ans == 'Y' || ans == 'y')
    {
        cout << "Введите имя файла для вывода: ";
        cin >> filenameo;
        fout.open(filenameo);

        fout << "Количество вершин: " << vertexNumber << endl;
        for (int i = 0; i < vertexNumber; i++)
        {
            for (int j = 0; j < vertexNumber; j++)
            {
                fout << graph[i][j] << " ";
            }
            fout << endl;
        }
        fout << "Хроматическое число графа: " << gr;

        fout.close();
        cout << "Данные записаны в файл" << endl;
        exit(EXIT_SUCCESS);
    }
    else
    {
        continue;
    }
}
}

```

```
int main()
```

```
{
```

```
    setlocale(LC_ALL, "rus");
```

```
    srand(time(NULL));
```

```
    ifstream fin;
```

```
    char filename[256];
```

```
    int rounds, vertexNumber, X1, X2, X;
```

```
    char** currGraph;
```

```
    char** graphh;
```

```
    cout <<
```

```
"
```

```
_" << endl
```

```
    << "|
```

```
|" << endl
```

```
    << "|
```

Курсовая работа

```
|" << endl
```

```
    << "|
```

по дисциплине 'Логика и основы алгоритмизации в инженерных

```
задачах'
```

```
|" << endl
```

на тему 'Разработка алгоритма нахождения хроматического числа

```
графа'
```

```
    << "|
```

```
|" << endl
```

```
    <<
```

```
"|
```

```
_" << endl
```

```
    << "|
```

```
|" << endl
```

```
    << "|
```

Выполнил: студент группы 21BB3 Самохвалов Ярослав

```
|" << endl
```

```
    <<
```

```
"|
```

```
_" << endl;
```

```
    system("pause");
```

```
    system("cls");
```

```
    Menu1();
```

```

while (_stateMenu1 != 0)
{
    switch (_stateMenu1)
    {
    case 1:
        cout << "Введите количество вершин в графе: ";
        cin >> vertexNumber;

        currGraph = new char* [vertexNumber];
        graphh = new char* [vertexNumber];
        cout << "Введите матрицу смежности:" << endl;
        for (int i = 0; i < vertexNumber; i++)
        {
            currGraph[i] = new char[vertexNumber];
            graphh[i] = new char [vertexNumber];
            for (int j = 0; j < vertexNumber; j++)
            {
                cin >> currGraph[i][j];
                graphh[i][j] = currGraph[i][j];
            }
        }
        cout << endl;
        system("pause");
        cout << endl;
        Menu2();
        while (_stateMenu2 != 0)
        {
            switch (_stateMenu2)
            {
            case 1:
                cout << "Введите количество повторений данного алгоритма (для
повышения точности): ";

                cin >> rounds;
                X1 = Kras(vertexNumber, currGraph, rounds);
                PrintFile(vertexNumber, graphh, X1);
            case 2:
                X2 = Greed(vertexNumber, currGraph);
                PrintFile(vertexNumber, graphh, X2);
            case 3:
                cout << "Введите количество повторений алгоритма Красновой (для
повышения точности): ";

                cin >> rounds;
                X1 = Kras(vertexNumber, currGraph, rounds);
                X2 = Greed(vertexNumber, currGraph);
                X = (X1 < X2 ? X1 : X2);
                PrintFile(vertexNumber, graphh, X);
            case 0:
                exit(EXIT_SUCCESS);
            default:
                printf("Неправильный номер!");
                system("pause");
                Menu2();
                break;
            }
        }
    case 2:
        cout << "Введите количество вершин в графе: ";
        cin >> vertexNumber;

        int k;
        graphh = new char* [vertexNumber];
        currGraph = new char* [vertexNumber];
        cout << "Выберите вид графа (Ориентированный - 1 / Неориентированный - 2: ";
        cin >> k;
        if (k == 1)
        {
            for (int i = 0; i < vertexNumber; i++)

```

```

        {
            currGraph[i] = new char[vertexNumber];
            graphh[i] = new char [vertexNumber];
            for (int j = 0; j < vertexNumber; j++)
            {
                currGraph[i][j] = (char)(((int)'0') + rand() % 2);
                if (i == j) currGraph[i][j] = '0';
                graphh[i][j] = currGraph[i][j];
            }
        }
    }
else
{
    for (int i = 0; i < vertexNumber; i++)
    {
        currGraph[i] = new char[vertexNumber];
        graphh[i] = new char [vertexNumber];
    }
    for (int i = 0; i < vertexNumber; i++)
    {
        for (int j = 0; j < i + 1; j++)
        {
            currGraph[i][j] = (char)(((int)'0') + rand() % 2);
            currGraph[j][i] = currGraph[i][j];
            if (i == j) currGraph[i][j] = '0';
            graphh[i][j] = currGraph[i][j];
            graphh[j][i] = currGraph[i][j];
        }
    }
    PrintGraph(currGraph, vertexNumber);
    system("pause");
    cout << endl;
    Menu2();
    while (_stateMenu2 != 0)
    {
        switch (_stateMenu2)
        {
            case 1:
                cout << "Введите количество повторений данного алгоритма (для
повышения точности): ";

                cin >> rounds;
                X1 = Kras(vertexNumber, currGraph, rounds);
                PrintFile(vertexNumber, graphh, X1);
            case 2:
                X2 = Greed(vertexNumber, currGraph);
                PrintFile(vertexNumber, graphh, X2);
            case 3:
                cout << "Введите количество повторений алгоритма Красновой (для
повышения точности): ";

                cin >> rounds;
                X1 = Kras(vertexNumber, currGraph, rounds);
                X2 = Greed(vertexNumber, currGraph);
                X = (X1 < X2 ? X1 : X2);
                PrintFile(vertexNumber, graphh, X);
            case 0:
                exit(EXIT_SUCCESS);
            default:
                printf("Неправильный номер!");
                system("pause");
                Menu2();
                break;
        }
    }
case 3:
    cout << "Введите имя файла: ";
    cin >> filename;
    fin.open(filename);

```

```

if (!fin.is_open())
{
    cout << "Не удалось открыть файл!" << endl;
    system("pause");
    exit(EXIT_SUCCESS);
}

fin >> vertexNumber;
currGraph = new char* [vertexNumber];
graphh = new char* [vertexNumber];
for (int i = 0; i < vertexNumber; i++)
{
    currGraph[i] = new char[vertexNumber];
    graphh[i] = new char [vertexNumber];
    for (int j = 0; j < vertexNumber; j++)
    {
        fin >> currGraph[i][j];
        if (i == j) currGraph[i][j] = '0';
        graphh[i][j] = currGraph[i][j];
    }
}
PrintGraph(currGraph, vertexNumber);
system("pause");
cout << endl;
Menu2();
while (_stateMenu2 != 0)
{
    switch (_stateMenu2)
    {
        case 1:
            cout << "Введите количество повторений данного алгоритма (для
повышения точности): ";

            cin >> rounds;
            X1 = Kras(vertexNumber, currGraph, rounds);
            PrintFile(vertexNumber, graphh, X1);
        case 2:
            X2 = Greed(vertexNumber, currGraph);
            PrintFile(vertexNumber, graphh, X2);
        case 3:
            cout << "Введите количество повторений алгоритма Красновой (для
повышения точности): ";

            cin >> rounds;
            X1 = Kras(vertexNumber, currGraph, rounds);
            X2 = Greed(vertexNumber, currGraph);
            X = (X1 < X2 ? X1 : X2);
            PrintFile(vertexNumber, graphh, X);
        case 0:
            exit(EXIT_SUCCESS);
        default:
            printf("Неправильный номер!");
            system("pause");
            Menu2();
            break;
    }
}
case 0:
    exit(EXIT_SUCCESS);
default:
    system("cls");
    printf("Неправильный номер!");
    system("pause");
    system("cls");
    Menu1();
    break;
}
}
}

```

Файл Gredd.h

```
#pragma once
#include "time.h"
#include <Windows.h>
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <conio.h>
#include <locale.h>

using namespace std;

class Vertex
{
public:
    int number;
    int degree;
    vector<int> lockedColors;

    Vertex()
    {
        number = 0;
        degree = 0;
        lockedColors = vector<int>();
    };

    Vertex(char** graph, int vertexNumber, int index)
    {
        number = index;
        degree = 0;
        for (int i = 0; i < vertexNumber; ++i)
            if (graph[index][i] == '1')
                ++degree;
        lockedColors = vector<int>();
    }
};

bool operator < (Vertex A, Vertex B)
{
    if (A.lockedColors.size() < B.lockedColors.size())
        return true;
    else if (A.lockedColors.size() > B.lockedColors.size())
        return false;
    else if (A.degree < B.degree)
        return true;
    else return false;
}

bool operator <= (Vertex A, Vertex B)
{
    if (A.lockedColors.size() < B.lockedColors.size())
        return true;
    else if (A.lockedColors.size() > B.lockedColors.size())
        return false;
    else if (A.degree <= B.degree)
        return true;
    else return false;
}

bool operator > (Vertex A, Vertex B)
{
    if (A.lockedColors.size() > B.lockedColors.size())
        return true;
    else if (A.lockedColors.size() < B.lockedColors.size())
        return false;
}
```

```

        else if (A.degree > B.degree)
            return true;
        else return false;
    }

bool operator >= (Vertex A, Vertex B)
{
    if (A.lockedColors.size() > B.lockedColors.size())
        return true;
    else if (A.lockedColors.size() < B.lockedColors.size())
        return false;
    else if (A.degree >= B.degree)
        return true;
    else return false;
}

ostream& operator << (ostream& os, const Vertex A)
{
    os << A.number;
    return os;
}

void quickSort(int l, int r, vector<Vertex>& a, vector<int>& indexes)
{
    Vertex x = a[(l + r) / 2];
    int i = l;
    int j = r;
    while (i <= j)
    {
        while (a[i] < x) i++;
        while (a[j] > x) j--;
        if (i <= j)
        {
            swap(a[i], a[j]);
            swap(indexes[a[i].number], indexes[a[j].number]);
            i++;
            j--;
        }
    }
    if (i < r)
        quickSort(i, r, a, indexes);
    if (l < j)
        quickSort(l, j, a, indexes);
}

void addColor(vector<vector<int>>& colors, vector<int>& CF)
{
    colors.push_back(vector<int>());
    colors.back().push_back(0);
    CF.push_back(CF.size());
}

int Greed(int vertexNumbe, char** curGraph)
{
    char** currGraph = curGraph;
    int vertexNumber = vertexNumbe;
    vector<Vertex> vertexes;
    vertexes.reserve(vertexNumber);
    for (int i = 0; i < vertexNumber; ++i)
        vertexes.push_back(Vertex(currGraph, vertexNumber, i));

    vector<int> newIndexes;
    newIndexes.reserve(vertexNumber);
    for (int i = 0; i < vertexNumber; ++i)
        newIndexes.push_back(i);

    vector<vector<int>> colors;
    vector<int> colorFrequency;

```

```

addColor(colors, colorFrequency);

quickSort(0, vertexNumber - 1, vertexes, newIndexes);

bool isLocked = false;
int currentColor = 0;
bool toAdd = true;

//Пакетная
while (!vertexes.empty())
{
    for (int i = colorFrequency.size() - 1; i >= 0; --i)
    {
        currentColor = colorFrequency[i];
        //поиск цвета
        isLocked = false;
        for (int j = 0; j < vertexes.back().lockedColors.size(); ++j)
        {
            if (currentColor == vertexes.back().lockedColors[j])
                isLocked = true;
        }
        //если цвет найден
        if (!isLocked)
        {
            //добавляем вершину в список
            colors[currentColor].push_back(vertexes.back().number);
            ++colors[currentColor][0];
            //коррекция связанных вершин
            for (int n = 0; n < vertexNumber; ++n)
            {
                if (currGraph[vertexes.back().number][n] == '1')
                {
                    //Проверка необходимости добавления заблокированного цвет
                    (без повтора)
                    if (vertexes[newIndexes[n]].lockedColors.size() == 0)
                        toAdd = true;
                    else
                    {
                        toAdd = true;
                        for (int j = 0; j <
vertexes[newIndexes[n]].lockedColors.size(); ++j)
                        {
                            if (currentColor ==
vertexes[newIndexes[n]].lockedColors[j])
                            {
                                toAdd = false;
                                break;
                            }
                        }
                    }
                    //Добавление соседнего цвета
                    if (toAdd)
                    {
                        vertexes[newIndexes[n]].lockedColors.push_back(currentColor);
                        //Пересортировка вершин
                        while (newIndexes[n] + 2 < vertexes.size() &&
vertexes[newIndexes[n]] > vertexes[newIndexes[n] + 1])
                        {
                            swap(vertexes[newIndexes[n]],
vertexes[newIndexes[n] + 1]);

                            swap(newIndexes[vertexes[newIndexes[n]].number], newIndexes[vertexes[newIndexes[n] +
1].number]);
                        }
                    }
                }
            }
        }
    }
}

```

```

    }

    //Пересортировка цветов
    int k = 0;
    while (k < i)
    {
        if (colors[colorFrequency[i - k]][0] > colors[colorFrequency[i -
k - 1]][0])
        {
            swap(colorFrequency[i - k], colorFrequency[i - k - 1]);
            ++k;
        }
        else break;
    }

    //Удаление вершин
    for (int m = 0; m < vertexNumber; ++m)
    {
        currGraph[m][vertexes.back().number] = '0';
        currGraph[vertexes.back().number][m] = '0';
    }
    vertexes.pop_back();
    break;
}

//Если нет свободного цвета
if (isLocked && i == 0)
{
    addColor(colors, colorFrequency);
    break;
}
}

return colors.size();
}

```

Файл Kras.h

```

#pragma once
#include "time.h"
#include <Windows.h>
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <conio.h>
#include <locale.h>

using namespace std;

int** multiply(int size, int** A, int** B)
{
    int** X = new int* [size];
    for (int i = 0; i < size; ++i)
        X[i] = new int[size];
    for (int i = 0; i < size; ++i)
        for (int j = 0; j < size; ++j)
            X[i][j] = 0;
    for (int i = 0; i < size; ++i)
        for (int j = 0; j < size; ++j)
            for (int k = 0; k < size; ++k)
                X[i][j] += A[i][k] * B[k][j];

    return X;
}

int** transp(int size, int** A)
{

```

```

int** X = new int* [size];
for (int i = 0; i < size; ++i)
    X[i] = new int[size];
for (int i = 0; i < size; ++i)
    for (int j = 0; j < size; ++j)
        X[i][j] = A[j][i];
return X;
}

char** mix(int vertexNumber, char** oldGraph, vector<int> groups, int groupNumber)
{
    vector<vector<int>> groupMembers(groupNumber);
    for (int i = 0; i < vertexNumber; ++i)
        groupMembers[groups[i]].push_back(i);
    vector<int> newOrder;
    newOrder.clear();
    for (int i = 0; i < groupMembers.size(); ++i)
        for (int j = 0; j < groupMembers[i].size(); ++j)
            newOrder.push_back(groupMembers[i][j]);
    int** p = new int* [vertexNumber];
    for (int i = 0; i < vertexNumber; ++i)
        p[i] = new int[vertexNumber];
    for (int i = 0; i < vertexNumber; ++i)
        for (int j = 0; j < vertexNumber; ++j)
            p[i][j] = 0;
    for (int i = 0; i < vertexNumber; ++i)
        p[i][vertexNumber - newOrder[i] - 1] = 1;
    int** newGraph = new int* [vertexNumber];
    for (int i = 0; i < vertexNumber; ++i)
        newGraph[i] = new int[vertexNumber];
    for (int i = 0; i < vertexNumber; ++i)
        for (int j = 0; j < vertexNumber; ++j)
            newGraph[i][j] = oldGraph[i][j] - '0';
    int** x = multiply(vertexNumber, p, newGraph);
    int** pp = transp(vertexNumber, p);
    int** newNewGraph = multiply(vertexNumber, x, pp);
    for (int i = 0; i < vertexNumber; ++i)
        for (int j = 0; j < vertexNumber; ++j)
            oldGraph[i][j] = '0' + newNewGraph[i][j];
    for (int i = 0; i < vertexNumber; ++i)
        delete[] x[i];
    delete[] x;
    for (int i = 0; i < vertexNumber; ++i)
        delete[] pp[i];
    delete[] pp;
    for (int i = 0; i < vertexNumber; ++i)
        delete[] newGraph[i];
    delete[] newGraph;
    for (int i = 0; i < vertexNumber; ++i)
        delete[] newNewGraph[i];
    delete[] newNewGraph;
    for (int i = 0; i < vertexNumber; ++i)
        delete[] p[i];
    delete[] p;
    return oldGraph;
}

int Kras(int vertexNumber, char** currGraph, int rounds)
{
    int bestGroupCount = INT_MAX;
    vector<int> bestGroups;

    vector<int> groups(vertexNumber, -1);

    int currGroup = -1;
    int finishedVertexes = 0;
    vector<int> currGroupMembers;
    bool is_fit = true;

```

```

//Начало
for (int round = 0; round < rounds; ++round)
{
    //Перезапуск
    groups.assign(vertexNumber, -1); currGroup = -1; finishedVertexes = 0; is_fit = true;
    while (finishedVertexes < vertexNumber)
    {
        ++currGroup;
        currGroupMembers.clear();
        is_fit = true;
        for (int i = 0; i < vertexNumber; ++i, is_fit = true)
        {
            if (groups[i] != -1) //Уже назначенный
                continue;
            for (int j = 0; j < currGroupMembers.size(); ++j)
                if (currGraph[i][currGroupMembers[j]] == '1') //соединение с
                    is_fit = false;
            if (is_fit)
            {
                groups[i] = currGroup;
                ++finishedVertexes;
                currGroupMembers.push_back(i);
            }
        }

        //Проверка на улучшение
        if (currGroup + 1 < bestGroupCount)
        {
            bestGroupCount = currGroup + 1;
            bestGroups = groups;
        }

        //соединение вершин
        currGraph = mix(vertexNumber, currGraph, groups, currGroup + 1);
    }

    vector<vector<int>> groupMembers(currGroup + 1);

    for (int i = 0; i < vertexNumber; ++i)
        groupMembers[bestGroups[i]].push_back(i);

    return bestGroupCount;
}

```