**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ**

**Кафедра математической кибернетики и системного анализа**

**Задача о раскраске графа**

Курсовая работа

Суражева Кирилла Леонидовича, студента 2 курса, специальность «веб-программирование и интернет-технологии»

Научный руководитель: ассистент кафедры математической кибернетики Харитонова Вероника Ренальдовна

Минск,2020

**Содержание**

Введение……………………………………………………..стр.3

Понятие раскраски графа…………………………………...стр.4

История………………………………………………………стр.4

Определение и терминология………………………………стр.6

Раскраска вершин…………………………………….стр.6

Рёберная раскраска…………………………………...стр.6

Тотальная раскраска…………………………………. стр.7

Алгоритмы раскраски……………………………………… стр.8

Полиномиальные алгоритмы……………………….. стр.8

Точные алгоритмы…………………………………... стр.8

Жадная раскраска……………………………………. стр.8

Параллельные и распределённые алгоритмы……… стр.9

Децентрализованные алгоритмы…………………… стр.10

Сложность вычислений……………………………... стр.10

Применение………………………………………………… стр.11

Планирование……………………………………….. стр.11

Распределение регистров…………………………… стр.11

Цифровые водяные знаки…………………………... стр.11

Присвоение частот подвижной радиостанции……. стр.12

Судоку……………………………………………….. стр.12

Двухсторониие графы……………………………… стр.12

Раскраска карты……………………………………. стр.12

Практическая часть……………………………………….. стр.13

1.Жадный алгоритм………………………………… стр.13

2.Через матрицу смежности……………………...... стр.17

Заключение………………………………………………... стр.20

Список используемых источников ……………………… стр.21

## **Введение**

Понятие «граф» связано с понятием «графический», «графика». Действительно, графовые модели имеют простую и понятную графическую интерпретацию, позволяющую с их помощью образно представить самые разные объекты, в то же время оставаясь в рамках строгих математических моделей. Первой работой теории графов как математической дисциплины считают статью Эйлера (1736г.), в которой рассматривалась задача о Кёнигсбергских мостах. Эйлер показал, что нельзя обойти семь городских мостов и вернуться в исходную точку, пройдя по каждому мосту ровно один раз. Следующий импульс теория графов получила спустя почти 100 лет с развитием исследований по электрическим сетям, кристаллографии, органической химии и другим наукам. С графами, сами того не замечая, мы сталкиваемся постоянно. Например, графом является схема линий метрополитена. Точками на ней представлены станции, а линиями - пути движения поездов. Исследуя свою родословную и возводя ее к далекому предку, мы стоим так называемое генеалогическое древо. И это древо - граф. Методы теории графов широко применяются в дискретной математике. Без них невозможно обойтись при анализе и синтезе различных дискретных преобразователей: функциональных блоков компьютеров, комплексов программ и т.д. В настоящее время теория графов охватывает большой материал и активно развивается.

## **Понятие раскраски графа**

**Раскраска графа** — теоретико-графовая конструкция, частный случай разметки графа. При раскраске элементам графа ставятся в соответствие метки с учётом определённых ограничений; эти метки традиционно называются «цветами». В простейшем случае такой способ окраски вершин графа, при котором любым двум смежным вершинам соответствуют разные цвета, называется **раскраской вершин**. Аналогично **раскраска рёбер** присваивает цвет каждому ребру так, чтобы любые два смежных ребра имели разные цвета. Наконец, **раскраска областей** планарного графа назначает цвет каждой области, так, что каждые две области, имеющие общую границу, не могут иметь одинаковый цвет.

**Раскраска вершин** — главная задача раскраски графов, все остальные задачи в этой области могут быть сведены к ней. Например, раскраска рёбер графа — это раскраска вершин его рёберного графа, а раскраска областей планарного графа — это раскраска вершин его двойственного графа. Тем не менее, другие проблемы раскраски графов часто ставятся и решаются в изначальной постановке. Причина этого частично заключается в том, что это даёт лучшее представление о происходящем и более показательно, а частично из-за того, что некоторые задачи таким образом решать удобнее (например, раскраска рёбер).

Раскраска графов находит применение и во многих практических областях, а не только в теоретических задачах. Помимо классических типов проблем, различные ограничения могут также быть наложены на граф, на способ присвоения цветов или на сами цвета. Этот метод, например, используется в популярной головоломке Судоку. В этой области всё ещё ведутся активные исследования.

## **История**

Первые результаты были получены для плоских графов в задаче раскрашивания карт. Пытаясь раскрасить карту округов Англии, Францис Гутри сформулировал проблему четырёх красок, отметив, что четырёх цветов достаточно, чтобы раскрасить карту так, чтобы любые два смежных региона имели разные цвета. Его брат передал вопрос своему учителю по математике, Огастесу де Моргану, который упомянул о нём в своем письме Уильяму Гамильтону в 1852 году. Артур Кэли поднял эту проблему на встрече Лондонского математического сообщества в 1878 году. В том же году Тэйтом было предложено первое решение этой задачи. Раскраску вершин первоначального графа он свел к раскраске рёбер двойственного графа и предположил, что эта задача всегда имеет решение. В 1880 году Альфред Кемпе опубликовал статью, в которой утверждалось, что ему удалось установить результат, и на десятилетие проблема четырёх цветов считалась решённой. За это достижение Кемпе был избран членом Лондонского Королевского общества и позже — президентом Лондонского математического сообщества.

В 1890 году Хивуд нашёл ошибку в доказательстве Кемпе. В этой же статье он доказал теорему пяти красок, показав, что любая плоская карта может быть раскрашена не более, чем пятью цветами. При этом он опирался на идеи Кемпе. В следующем столетии было разработано большое количество теорий в попытках уменьшить минимальное число цветов. Теорема четырёх красок была окончательно доказана в 1977 году учеными Кеннетом Аппелем и Вольфгангом Хакеном с использованием компьютерного перебора. Идея доказательства во многом опиралась на идеи Хивуда и Кемпе и игнорировала большинство промежуточных исследований. Доказательство теоремы четырёх красок является одним из первых доказательств, в которых был использован компьютер.

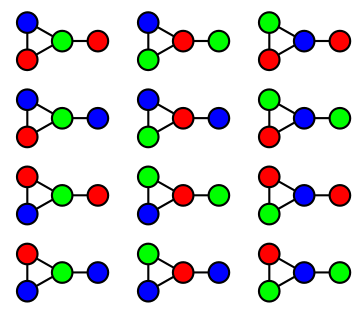
В 1912 году Джордж Дэвид Биркхоф предложил использовать для изучения задач раскраски хроматический многочлен, являющийся важной частью в алгебраической теории графов. Хроматический многочлен впоследствии был обобщён Уильямом Таттом (многочлен Татта). Кемпе в 1879 году уже обращал внимание на общий случай, когда граф не являлся плоским. Много результатов обобщений раскраски плоских графов на поверхности более высоких порядков появилось в начале 20 века.

В 1960 году Клод Бердж сформулировал гипотезу о совершенных графах, мотивированное понятием из теории информации, а именно нулевой ошибкой ёмкости графа, представленным Шенноном. Утверждение оставалось неподтвержденным на протяжении 40 лет, пока не было доказано как знаменитая строгая теорема о совершенных графах математиками Чудновской, Робертсоном, Сеймуром и Томасом в 2002 году.

Раскраска графов как алгоритмическая проблема начала изучаться с 1970-х годов: определение хроматического числа — входит в число 21 NP-полных задач Карпа (1972). И примерно в то же время были разработаны разнообразные алгоритмы на базе поиска с возвратом и рекурсивного удаления и стягивания Зыкова. С 1981 года раскраска графа применяется для распределения регистров в компиляторах.

## **Определение и терминология**

### Раскраска вершин

[](https://commons.wikimedia.org/wiki/File:Graph_with_all_three-colourings_2.svg?uselang=ru)

Этот граф может быть раскрашен 3 цветами 12 способами.

Когда говорят о раскраске графов, почти всегда подразумевают под этим раскраску их вершин, то есть присвоение цветовых меток вершинам графа так, чтобы любые две вершины, имеющие общее ребро, имели разные цвета. Так как графы, в которых есть петли, не могут быть раскрашены таким образом, они не являются предметом обсуждения.

Терминология, в которой метки называются цветами, происходит от раскраски политических карт. Такие метки как *красный* или *синий* используются, только когда число цветов мало, обычно же подразумевается, что метки являются целыми числами {1,2,3}{\displaystyle \{1,2,3,...\}}.

Раскраска с использованием {\displaystyle k}k цветов называется {\displaystyle k}k**-раскраской**. Наименьшее число цветов, необходимое для раскраски графа {\displaystyle G}G, называется его хроматическим числом и часто записывается как {\displaystyle \chi (G)}x(G). Иногда используется {\displaystyle \gamma (G)}y(G), с тех пор как y(G){\displaystyle \chi (G)} обозначает Эйлерову характеристику. Подмножество вершин, выделенных одним цветом, называется ***цветовым классом***, каждый такой класс формирует независимый набор. Таким образом, k{\displaystyle k}-раскраска — это то же самое, что и разделение вершин на k{\displaystyle k} независимых наборов.

### Рёберная раскраска

**Рёберная раскраска** графа подразумевает под собой назначение цветов ребрам так, что никакие два ребра одного цвета не принадлежат одной вершине. Эта задача эквивалентна разделению множества граней на множества независимых граней. Наименьшее число цветов, необходимое для рёберной раскраски графа {\displaystyle G}G — это его **хроматический индекс**, или **рёберное хроматическое число**{\displaystyle \chi '(G)} x’(G)

### Тотальная раскраска

**Тотальная раскраска** — это один из видов раскраски вершин **и** рёбер графа. Под ней подразумевают такое присвоение цветов, что ни соседние вершины, ни смежные ребра, ни вершины и ребра, которые их соединяют, не имеют одинакового цвета. Полное хроматическое число {\displaystyle \chi ''(G)}x’’(G) графа {\displaystyle G}G — это наименьшее число цветов, необходимое для любой полной раскраски.

## **Алгоритмы раскраски**

### Полиномиальные алгоритмы

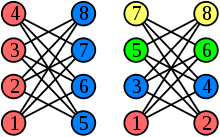
Для двудольного графа задача раскраски вычисляется за линейное время с помощью поиска в ширину. В случае совершенных графов, хроматическое число и соответствующая ему раскраска может быть найдена за полиномиальное время при использовании полуопределённого программирования. Точные формулы для нахождения хроматического числа известны для многих классов графов (леса, циклы, колеса, хордальные графы) и так же могут быть вычислены за полиномиальное время.

### Точные алгоритмы

Алгоритм полного перебора для случая k-раскраски рассматривает все {\displaystyle k^{n}}k^n комбинаций расстановки цветов в графе с n вершинами и проверяет их на корректность. Чтобы вычислить хроматическое число и хроматический полином, данный алгоритм рассматривает каждое k от 1 до n. Такой алгоритм на практике может быть применим только для небольших графов.

Используя динамическое программирование и оценку размера наибольшего независимого множества, в графе возможность k-раскраски может быть разрешена за время O(2,445^n){\displaystyle O(2,445^{n})}. Известны более быстрые алгоритмы для 3- и 4-раскрасок, работающие за время {\displaystyle O(1,3289^{n})} и {\displaystyle O(1,7272^{n})}соответственно.

### Жадная раскраска

[](https://commons.wikimedia.org/wiki/File:Greedy_colourings.svg?uselang=ru)

Два результата работы жадного алгоритма при выборе разных порядков вершин.

Жадный алгоритм упорядочивает вершины {\displaystyle v\_{1}}v1…vn{\displaystyle v\_{n}} и последовательно присваивает вершине {\displaystyle v\_{i}}vi наименьший доступный цвет, не использовавшийся для окраски соседей {\displaystyle v\_{i}}vi среди vi…vi-1{\displaystyle v\_{1}}v1//vvv либо добавляет новый. Качество полученной раскраски зависит от выбранного порядка. Всегда существует такой порядок, который приводит жадный алгоритм к оптимальному числу {\displaystyle \chi (G)}x(G) красок. С другой стороны, жадный алгоритм может быть сколь угодно плохим; например, корона с n вершинами может быть раскрашена 2 цветами, но существует порядок вершин, который приводит к жадной раскраске из {\displaystyle n/2}n/2 цветов.

Для хордального графа и для его особых случаев (например, интервальный граф) алгоритм жадной раскраски может быть использован для нахождения оптимальной раскраски за полиномиальное время, выбирая порядок вершин обратным к совершенному порядку исключения. Этот алгоритм может применен и к более широкому классу графов (совершенно упорядочиваемые графы), однако найти такой порядок для таких графов — NP-сложная задача.

Если вершины упорядочены в соответствии с их степенями, алгоритм жадной раскраски использует не более чем {\displaystyle max\_{i}min\{d(v\_{i})+1,i\}}max\_i\_{d(vi)+1,i}min цветов, что максимум на 1 больше, чем {\displaystyle \Delta }d (здесь {\displaystyle d(v\_{i})}d(vi) — степень вершины {\displaystyle v\_{i}}vi). Этот эвристический алгоритм иногда называют алгоритмом Уэлша-Пауэлла. Другой алгоритм устанавливает порядок динамично, выбирая следующую вершину той, которая имеет наибольшее число смежных вершин разных цветов. Многие другие алгоритмы раскраски графов основаны на жадной раскраске и используют статические или динамические стратегии выбора порядка вершин.

### Параллельные и распределённые алгоритмы

В области распределённых алгоритмов встречается аналогичная задача. Допустим, вершины графа — это компьютеры, которые могут общаться между собой, если они соединены ребром. Задача состоит в том, чтобы каждый компьютер выбрал для себя «цвет», так, чтобы соседние компьютеры выбрали разные цвета. Эта задача тесно связана с проблемой нарушения симметрии. Наиболее развитые вероятностные алгоритмы работают быстрее, чем детерминированные алгоритмы для графов с достаточно большой максимальной степени вершин {\displaystyle \Delta }d. Наиболее быстрые вероятностные алгоритмы используют технику множественных попыток.

В симметричных графах детерминированные распределённые алгоритмы не могут найти оптимальную раскраску вершин. Нужна дополнительная информация, чтобы избежать симметрии. Делается стандартное предположение, что первоначально каждая вершина имеет уникальный идентификатор, например, из множества {\displaystyle \{1,2,...,N\}}{1,2,..N}. Иными словами, предполагается, что нам дана *n*-раскраска. Задача состоит в том, чтобы уменьшить количество цветов от *n* до, например, {\displaystyle (\Delta +1)}d+1. Чем больше цветов используются (например, {\displaystyle O(\Delta )}O(d) вместо {\displaystyle (\Delta +1)}(d+1)), тем меньше обменов сообщений потребуется.

Простая версия распределённого жадного алгоритма для ({\displaystyle (\Delta +1)}d+1) -раскраски требует {\displaystyle \Theta (n)}O(n) раундов связи в худшем случае — информации, возможно, придется проходить с одного конца стороны сети до другого.

Наиболее простым интересным случаем является *n*-цикл. Ричард Коул и Узи Вишкин показали, что существует распределённый алгоритм, который уменьшает количество цветов от *n* до {\displaystyle O(log(n))}O(log(n)), используя лишь один раз обмен сообщениями между соседями. Повторяя ту же процедуру, можно получить 3-раскраску *n*-цикла за {\displaystyle O(log^{\*}(n))}(O(log\*(n))  раундов связи (при условии, что даны уникальные идентификаторы узлов).

Функция log\*n{\displaystyle log^{\*}}log, итерированный логарифм, является чрезвычайно медленно растущей функцией, «почти константа». Следовательно, результаты Коула и Вишкина поднимают вопрос о том, есть ли распределённый алгоритм 3-раскраски n-цикла, который выполняется за константное время. Натан Линиал показал, что это невозможно: любой детерминированный распределённый алгоритм требует {\displaystyle \Omega (log^{\*}(n))}sigma(log\*(n)) раундов связи для уменьшения *N*-раскраски до 3-раскраски в n-цикле.

Техника Коула и Вишкина также может быть применена для произвольного графа с ограниченной степенью вершин, в этом случае время работы составляет P(d)+O(log\*(n)) Этот метод был обобщен для графа единичных кругов Шнайдером и др.

Проблема раскраски рёбер также изучалась в распределённой модели. Пансонецци и Рицци достигли {\displaystyle (2\Delta -1)}(2d-1)-раскраски за {\displaystyle O(\Delta +log^{\*}(n))}O(d+log\*(n)) в этой модели. Нижняя граница для распределённой раскраски вершин, достигнутая Линиалом, также применима для задачи распределённой раскраски рёбер.

### Децентрализованные алгоритмы

Децентрализованными называются алгоритмы, в которых не разрешена внутренняя передача сообщений (в отличие от распределённых алгоритмов, где процессы обмениваются между собой данными). Существуют эффективные децентрализованные алгоритмы, успешно справляющиеся с задачей раскраски графов. Эти алгоритмы работают в предположении, что вершина способна «чувствовать», что какая-либо из её соседних вершин раскрашена в тот же цвет, что и она. Другими словами, есть возможность определить локальный конфликт. Такое условие довольно часто выполняется в реальных прикладных задачах — например, при передаче данных по беспроводному каналу передающая станция, как правило, имеет возможность детектировать, что другая станция пытается передавать одновременно в тот же канал. Способности к получению подобной информации достаточно для того, чтобы алгоритмы, основанные на обучающихся автоматах, с вероятностью единица правильно решат задачу раскраски графа.

**Сложность вычислений**

Раскраска графа является вычислительно сложной задачей. Узнать, допускает ли граф k-раскраску для заданного k — это NP-полная задача, кроме случаев k = 1 и k = 2. В частности, задача вычисления хроматического числа NP-сложна. Задача о 3-раскраске NP-полная даже для случая планарного графа степени 4.

Также NP-сложной задачей является раскраска 3-раскрашиваемого графа 4 цветами и k-раскрашиваемого графа при достаточно больших значениях k.

Вычисление коэффициентов хроматического полинома NP-сложная задача. Доказано, что не существует ни одного FPRAS-алгоритма для вычисления хроматического полинома ни для какого рационального числа k ≥ 1,5, кроме k = 2, если только не выполнится, что NP = RP.

Применение

**Планирование**

Раскраска вершин моделирует многие проблемы планирования. В своей простейшей постановке заданный набор работ должен быть распределен по временным отрезкам, каждая такая работа занимает один отрезок. Они могут быть выполнены в любом порядке, но две работы могут конфликтовать в том смысле, что не могут быть выполнены одновременно, так как, например, используют общие ресурсы. Соответствующий граф содержит вершину для каждой из работ и ребро для каждой конфликтующий пары. Хроматическое число построенного графа — это минимальное время выполнения всех работ без конфликтов.

Детали проблемы планирования определяют структуру графа. Например, когда идет распределение самолётов по рейсам, результирующий граф конфликтов является интервальным графом, так что проблема раскраски может быть решена эффективно. При распределении радиочастот получается граф единичных кругов конфликтов, и для такой задачи существует 3-аппроксимационный алгоритм.

**Распределение регистров**

Компилятор — это компьютерная программа, которая переводит один компьютерный язык в другой. Для улучшения времени выполнения результирующего кода одной из техник компиляторной оптимизации, является распределение регистров, в которой наиболее часто используемые переменные компилируемой программы хранятся в быстродействующих регистрах процессора. В идеальном случае переменные хранятся в регистрах так, что они все находятся в регистрах во время их использования.

Стандартный подход к этой задаче состоит в сведении её к задаче раскраски графов. Компилятор строит интерференционный граф, где вершины соответствуют переменным, а грань соединяет две из них, если они нужны в один и тот же момент времени. Если этот граф k-хроматический, то переменные могут храниться в k регистрах.

**Цифровые водяные знаки**

Технология цифровых водяных знаков (англ. digital watermarking) позволяет вместе с данными (будь то медиафайлы, исполняемые файлы и прочие) передать некое скрытое сообщение («водяной знак», Watermark). Такое скрытое сообщение может быть применено в защите авторских прав для идентификации владельца данных.

Это важно, например, для установления источника их распространения нелегальным образом. Или же для подтверждения прав на данные, например — программное обеспечение систем на кристалле (system-on-chip).

Сообщение можно закодировать в том числе и в способе распределения регистров. Одну из таких техник предложили Qu и Potkonjak (поэтому её иногда называют QP-алгоритмом).

### Присвоение частот подвижной радиосвязи

Когда частоты назначаются вышкам, частоты, назначенные всем вышкам в одном и том же месте, должны быть разными. Как назначить частоты с этим ограничением? Какое минимальное количество частот необходимо? Эта проблема также является примером проблемы раскраски графа, где каждая башня представляет вершину, а ребро между двумя башнями показывает, что они находятся в диапазоне друг от друга.

### Судоку

Судоку также является разновидностью проблемы раскраски Графа, где каждая ячейка представляет вершину. Между двумя вершинами есть ребро, если они находятся в одной строке, в одном столбце или в одном и том же блоке.

### Двухсторонние графы

Мы можем проверить, является ли граф двудольным или нет, раскрасив график двумя цветами. Если данный граф является 2-раскрашиваемым, то это двудольный, в противном случае нет.

### Раскраска карты

Географические карты стран или штатов, в которых двум соседним городам нельзя присвоить один и тот же цвет. Для раскраски любой карты достаточно четырех цветов (Теорема о четырех цветах)

## **Практическая часть**

Для данной курсовой я разработал несколько алгоритмов для раскраски графа

### 1.Жадный алгоритм

К сожалению, не существует эффективного алгоритма для раскраски графа с минимальным количеством цветов, так как проблема является известной NP проблемой. Есть приблизительные алгоритмы для решения этой проблемы. Ниже приведен основной жадный алгоритм для назначения цветов. Он не гарантирует использование минимальных цветов, но гарантирует верхнюю границу количества цветов. Базовый алгоритм никогда не использует более d + 1 цветов, где d - максимальная степень вершины в данном графе.

Основной жадный алгоритм раскраски:

1. Раскрасьте первую вершину первым цветом.

2. Выполните следующие действия для оставшихся вершин V-1.

… .. а) Рассмотрим выбранную вершину и раскрасим ее

цвет с наименьшим номером, который ранее не использовался

цветные вершины, прилегающие к нему. Если все ранее использованные цвета

появляются в вершинах рядом с v, назначьте ему новый цвет.

Java-реализация вышеуказанного алгоритма жадности.

|  |
| --- |
| // A Java program to implement greedy algorithm for graph coloring  import java.io.\*;  import java.util.\*;  import java.util.LinkedList;    // This class represents an undirected graph using adjacency list  class Graph  {      private int V;   // No. of vertices      private LinkedList<Integer> adj[]; //Adjacency List        //Constructor      Graph(int v)      {          V = v;          adj = new LinkedList[v];          for (int i=0; i<v; ++i)              adj[i] = new LinkedList();      }        //Function to add an edge into the graph      void addEdge(int v,int w)      {          adj[v].add(w);          adj[w].add(v); //Graph is undirected      }        // Assigns colors (starting from 0) to all vertices and      // prints the assignment of colors      void greedyColoring()      {          int result[] = new int[V];            // Initialize all vertices as unassigned          Arrays.fill(result, -1);            // Assign the first color to first vertex          result[0]  = 0;            // A temporary array to store the available colors. False          // value of available[cr] would mean that the color cr is          // assigned to one of its adjacent vertices          boolean available[] = new boolean[V];            // Initially, all colors are available          Arrays.fill(available, true);            // Assign colors to remaining V-1 vertices          for (int u = 1; u < V; u++)          {              // Process all adjacent vertices and flag their colors              // as unavailable              Iterator<Integer> it = adj[u].iterator() ;              while (it.hasNext())              {                  int i = it.next();                  if (result[i] != -1)                      available[result[i]] = false;              }                // Find the first available color              int cr;              for (cr = 0; cr < V; cr++){                  if (available[cr])                      break;              }                result[u] = cr; // Assign the found color                // Reset the values back to true for the next iteration              Arrays.fill(available, true);          }            // print the result          for (int u = 0; u < V; u++)              System.out.println("Vertex " + u + " --->  Color "                                  + result[u]);      }        // Driver method      public static void main(String args[])      {          Graph g1 = new Graph(5);          g1.addEdge(0, 1);          g1.addEdge(0, 2);          g1.addEdge(1, 2);          g1.addEdge(1, 3);          g1.addEdge(2, 3);          g1.addEdge(3, 4);          System.out.println("Coloring of graph 1");          g1.greedyColoring();            System.out.println();          Graph g2 = new Graph(5);          g2.addEdge(0, 1);          g2.addEdge(0, 2);          g2.addEdge(1, 2);          g2.addEdge(1, 4);          g2.addEdge(2, 4);          g2.addEdge(4, 3);          System.out.println("Coloring of graph 2 ");          g2.greedyColoring();      }  }  // This code is contributed by Surazhev Kirill |

Output:

Coloring of graph 1

Vertex 0 ---> Color 0

Vertex 1 ---> Color 1

Vertex 2 ---> Color 2

Vertex 3 ---> Color 0

Vertex 4 ---> Color 1

Coloring of graph 2

Vertex 0 ---> Color 0

Vertex 1 ---> Color 1

Vertex 2 ---> Color 2

Vertex 3 ---> Color 0

Vertex 4 ---> Color 3

Сложность времени: O (V ^ 2 + E) в худшем случае.

graph_coloring2

Анализ базового алгоритма

Приведенный выше алгоритм не всегда использует минимальное количество цветов. Кроме того, количество используемых цветов иногда зависит от порядка обработки вершин. Например, рассмотрим следующие два графика. Обратите внимание, что на графике справа вершины 3 и 4 поменялись местами. Если мы рассмотрим вершины 0, 1, 2, 3, 4 на левом графике, мы можем раскрасить график, используя 3 цвета. Но если мы рассмотрим вершины 0, 1, 2, 3, 4 в правом графе, нам понадобится 4 цвета.

Так что порядок, в котором выбираются вершины, важен. Многие люди предлагали различные способы найти порядок, который работает лучше, чем базовый алгоритм в среднем. Наиболее распространенным является алгоритм Уэлша-Пауэлла, который рассматривает вершины в порядке убывания степеней.

Как базовый алгоритм гарантирует верхнюю границу d + 1?

Здесь d - максимальная степень в данном графике. Поскольку d - максимальная степень, вершина не может быть присоединена к более чем d вершинам. Когда мы окрашиваем вершину, не более d цветов уже могли использоваться смежными. Чтобы раскрасить эту вершину, нам нужно выбрать наименьший пронумерованный цвет, который не используется соседними вершинами. Если цвета пронумерованы как 1, 2,…., То значение такого наименьшего числа должно быть от 1 до d + 1 (обратите внимание, что числа d уже выбраны смежными вершинами).

Это также можно доказать с помощью индукции.

### 2.Через матрицу смежности

Код:

**import** java.util.Scanner;  
  
*/\*\* Class GraphColoring \*\*/***public class** GraphColoring  
{  
 **private int V**, **numOfColors**;  
 **private int**[] **color**;  
 **private int**[][] **graph**;  
  
 */\*\* Function to assign color \*\*/* **public void** graphColor(**int**[][] g, **int** noc)  
 {  
 **V** = g.**length**;  
 **numOfColors** = noc;  
 **color** = **new int**[**V**];  
 **graph** = g;  
  
 **try** {  
 solve(0);  
 System.***out***.println(**"No solution"**);  
 }  
 **catch** (Exception e)  
 {  
 System.***out***.println(**"\nSolution exists "**);  
 display();  
 }  
 }  
 */\*\* function to assign colors recursively \*\*/* **public void** solve(**int** v) **throws** Exception  
 {  
 */\*\* base case - solution found \*\*/* **if** (v == **V**)  
 **throw new** Exception(**"Solution found"**);  
 */\*\* try all colours \*\*/* **for** (**int** c = 1; c <= **numOfColors**; c++)  
 {  
 **if** (isPossible(v, c))  
 {  
 */\*\* assign and proceed with next vertex \*\*/* **color**[v] = c;  
 solve(v + 1);  
 */\*\* wrong assignement \*\*/* **color**[v] = 0;  
 }  
 }  
 }  
 */\*\* function to check if it is valid to allot that color to vertex \*\*/* **public boolean** isPossible(**int** v, **int** c)  
 {  
 **for** (**int** i = 0; i < **V**; i++)  
 **if** (**graph**[v][i] == 1 && c == **color**[i])  
 **return false**;  
 **return true**;  
 }  
 */\*\* display solution \*\*/* **public void** display()  
 {  
 System.***out***.print(**"\nColors : "**);  
 **for** (**int** i = 0; i < **V**; i++)  
 System.***out***.print(**color**[i] +**" "**);  
 System.***out***.println();  
 }  
 */\*\* Main function \*\*/* **public static void** main (String[] args)  
 {  
 Scanner scan = **new** Scanner(System.***in***);  
 System.***out***.println(**"Graph Coloring Algorithm Test\n"**);  
 */\*\* Make an object of GraphColoring class \*\*/* GraphColoring gc = **new** GraphColoring();  
  
 */\*\* Accept number of vertices \*\*/* System.***out***.println(**"Enter number of verticesz\n"**);  
 **int** V = scan.nextInt();  
  
 */\*\* get graph \*\*/* System.***out***.println(**"\nEnter matrix\n"**);  
 **int**[][] graph = **new int**[V][V];  
 **for** (**int** i = 0; i < V; i++)  
 **for** (**int** j = 0; j < V; j++)  
 graph[i][j] = scan.nextInt();  
  
 System.***out***.println(**"\nEnter number of colors"**);  
 **int** c = scan.nextInt();  
  
 gc.graphColor(graph, c);  
  
 }  
}

Output:

Graph Coloring Algorithm Test

Enter number of vertices

10

Enter matrix

0 1 0 0 0 1 0 0 0 0

1 0 1 0 0 0 1 0 0 0

0 1 0 1 0 0 0 1 0 0

0 0 1 0 1 0 0 0 1 0

1 0 0 1 0 0 0 0 0 1

1 0 0 0 0 0 0 1 1 0

0 1 0 0 0 0 0 0 1 1

0 0 1 0 0 1 0 0 0 1

0 0 0 1 0 1 1 0 0 0

0 0 0 0 1 0 1 1 0 0

Enter number of colors

3

Solution exists

Colors : 1 2 1 2 3 2 1 3 3 2

Сложность времени: O (V ^ 2 + E) в худшем случае.

## **Заключение**

В настоящей курсовой работе рассмотрены математические графы, области их применения. Рассмотрены способы раскраски вершин и ребер графов и задачи на их применение. Графы достаточно широко применяются в математике, технике, экономике, управлении. Знание основ теории графов необходимо в различных областях, связанных с управлением производством, бизнесом (например, сетевой график строительства, графики доставки почты и др).

## **Список используемых источников**

Аляев Ю.А. , Тюрин С.Ф. Дискретная математика и математическая логика: учебник .- М.:Финансы и статистика, 2006-368с.:ил.

Белоусов А.И., Ткачев С.Б. Дискретная математика: Учеб.для вузов/ Под ред. В.С. Зарубина, А.П. Крищенко. - 3-к изд., стереотип.- М.: Изд-во МГТУ им. Н.Э. Баумана, 2004.- 744с. (Сер. Математика в техническом университете; Вып. XIX).

Берж К. "Теория графов и ее применение", М., ИЛ, 1962;

Битюцкий В.П., Соколов С.С.Основы дискретной математики: учебное пособие по дисциплине «Дискретная математика» .- Екатеринбург: ГОУ ВПО УГТУ-УПИ, 2005. Ч. 1. 96 c.

Носов. В.А. Комбинаторика и теория графов: учебное пособие. - М.: Изд-во МИЭМ( Технический университет),Москва,1999- 166с.