МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное

образовательное учреждение высшего образования

**«Челябинский государственный университет»**

**(ФГБОУ ВО «ЧелГУ»)**

Математический факультет

Кафедра компьютерной безопасности и прикладной алгебры

**КУРСОВАЯ РАБОТА**

**Совершенные графы**

Выполнил студент:

**Турлакова Анастасия Денисовна**

группы МК-201

очной формы обучения

специальности

10.05.01 Компьютерная безопасность

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

«\_\_\_» \_\_\_\_\_\_ 2023 г.

Научный руководитель:

Шалагинов Леонид Викторович

Должность: доцент

Ученая степень: канд.физ.-мат. наук

Ученое звание: доцент

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

«\_\_\_» \_\_\_\_\_\_ 2023 г.

Челябинск, 2023

**Содержание**

[Введение 2](#_Toc138858119)

[1. Цель работы 3](#_Toc138858120)

[2. Граф 4](#_Toc138858121)

[3. Характеристики и теоремы о совершенных графах 7](#_Toc138858122)

[4. Алгоритмы для поиска совершенных графов 10](#_Toc138858123)

[4.1. Алгоритм Брона-Кербоша 10](#_Toc138858124)

[4.2. Алгоритм Эдмондса-Карпа 13](#_Toc138858125)

[4.3. Алгоритм Ловаша 14](#_Toc138858126)

[5. Реализация алгоритма для нахождения совершенных графов 16](#_Toc138858127)

[5.1. Методы программы 16](#_Toc138858128)

[5.2. Алгоритм программы 17](#_Toc138858129)

[Заключение 19](#_Toc138858130)

[Список литературы 20](#_Toc138858131)

[Приложение 21](#_Toc138858132)

# Введение

В теории графов одним из наиболее интересных и важных объектов являются совершенные графы. Совершенный граф — это граф, в котором для любого непустого подмножества вершин выполняется равенство между размером этого подмножества и количеством его независимых вершинных множеств.

Совершенные графы являются объектами изучения не только теоретиков графов, но и приложений в различных областях, таких как теория кодирования, транспортная логистика, биоинформатика и др.

В данной работе мы рассмотрим основные свойства совершенных графов, их классификацию, а также приведем примеры применения в реальных задачах. Кроме того, мы рассмотрим некоторые алгоритмы для построения совершенных графов и докажем несколько теорем, связанных с этим классом графов.

## Цель работы

Цель данной работы заключается в изучении литературы по совершенным графам, а также реализации эффективного алгоритма для поиска максимальных клик в графе и тестировании программы на различных входных данных.

Для достижения поставленной цели необходимо решить следующие задачи:

1.1. Изучить литературу по совершенным графам;

1.2. Изучить алгоритмы;

1.3. Реализовать класс совершенных графов и выбранный алгоритм;

1.4. Сделать выводы об эффективности и применимости выбранного алгоритма.

## Граф

Граф — это абстрактный математический объект, представляющий из себя множество вершин графа и набор рёбер, то есть соединений между парами вершин.

Будем называть графом G пару (V,E), где V (G) — непустое конечное множество элементов, называемых вершинами графа, а E(G) — множество неупорядоченных пар различных элементов из V (G), называемых рёбрами.

Простой граф — граф, в котором нет кратных рёбер и петель.

В компьютерных программах графы часто представлены в виде матрицы смежности.

Матрица смежности графа с конечным числом вершин (пронумерованных числами от до ) — это квадратная матрица размера , в которой значение элемента равно числу рёбер из -й вершины графа в -ю вершину[2]. Пример графа и его матрицы смежности на рис. 1.

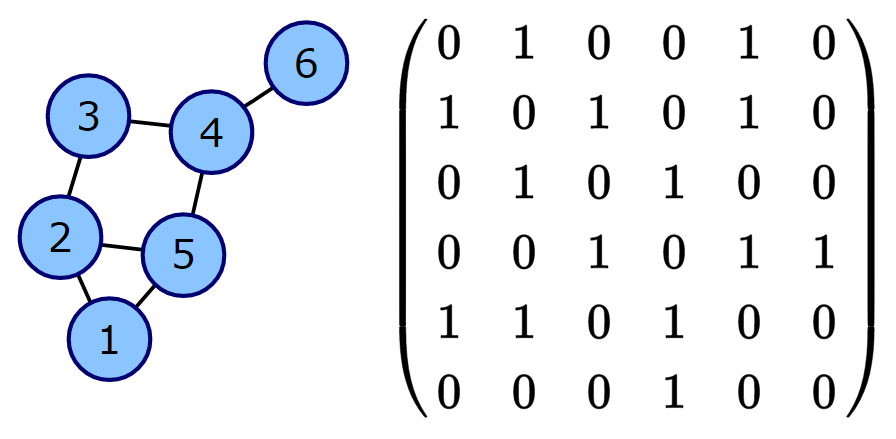


Рис. 1. Граф и его матрица смежности

Матрица смежности простого графа является бинарной матрицей и содержит нули на главной диагонали.

Граф называется -дольным(двудольным), если его вершины можно разбить на непересекающихся подмножества () так, что не будет рёбер, соединяющих вершины одного и того же подмножества.

Совершенный граф — это граф, в котором каждое независимое множество вершин имеет кликовое число, равное количеству вершин в этом множестве.

Кликовое число для множества вершин определяет количество ребер, которые связывают все вершины в данном множестве. Другими словами, совершенный граф — это граф, в котором нет ни одного лишнего ребра между независимыми вершинами и каждое максимальное независимое множество имеет максимальное число ребер. То есть, у которого выполнено равенство a(G) = o(G), то есть число стабильных множеств равно числу клик. В совершенном графе любое стабильное множество и любая клика не пересекаются.

## Характеристики и теоремы о совершенных графах

В своей первой работе о совершенных графах Берж высказал две важные гипотезы об их структуре, и они были доказаны позже. Первая из этих теорем была теорема о совершенных графах Ласло Ловаса (1972) утверждающая, что граф совершенен тогда и только тогда, когда его дополнение совершенно. Таким образом, совершенство (определённое как равенство размера максимальной клики и хроматического числа в любом порождённом подграфе) эквивалентно максимуму размера независимого множества и числа кликового покрытия.

Изображение выглядит как Красочность, размытие, снимок экрана, свет

Автоматически созданное описание

Рис. 2. Цикл с семью вершинами и его дополнение.

* 1. Утверждение теоремы

Совершенный граф — это неориентированный граф, в любом порождённом подграфе которого размер его наибольшей клики равен минимальному числу цветов раскраски подграфа. Совершенные графы включают много важных классов графов, куда входят двудольные графы, хордальные графы и графы сравнимости. Дополнение графа имеет ребро между двумя вершинами тогда и только тогда, когда исходный граф такого ребра не имеет. Таким образом, клика в исходном графе становится независимым множеством в дополнении и раскраска исходного графа становится кликовым покрытием дополнения.

Теорема о совершенных графах утверждает: Дополнение совершенного графа совершенно. Эквивалентная формулировка: В совершенном графе размер наибольшего независимого множества равен минимальному числу клик в кликовом покрытии.

* 1. Пример

Пусть G — граф-цикл нечётной длины, большей трёх (так называемая «нечётная дыра»). Тогда требуется для любой раскраски G по меньшей мере три цвета, но нет ни одного треугольника, так что граф не совершенен.

По теореме о совершенных графах дополнение графа G («нечётная антидыра») должно поэтому также быть несовершенным. Если граф G является циклом из пяти вершин, он изоморфен своему дополнению, но это неверно для более длинных циклов.

Нетривиальной задачей является вычисление кликового числа и хроматического числа в нечётной антидыре и нечётной дыре. Как утверждает строгая теорема о совершенных графах, нечётные дыры и нечётные антидыры оказываются минимальными запрещёнными порождёнными подграфами совершенных графов.

* 1. Связь двудольных графов с теорией совершенных графов и их релевантность в различных областях

В нетривиальном двудольном графе оптимальное число цветов (по определению) равно двум, и (поскольку двудольные графы не содержат треугольников) наибольший размер клики равен также двум.

Таким образом, любой порождённый подграф двудольного графа остаётся двудольным. Таким образом, двудольные графы являются совершенными. В двудольных графах с n вершинами наибольшее покрытие кликами принимает форму наибольшего паросочетания вместе с дополнительной кликой для каждой непокрытой вершины с размером n − M, где M — число элементов в паросочетании. В этом случае из теоремы о совершенных графах следует теорема Кёнига, что размер максимального независимого множества в двудольном графе в двудольном графе также равно n − M, результат, который был главным стимулом формулировки Бержем теории совершенных графов.

Теорему Мирского, описывающую высоту частично упорядоченного множества в терминах разбиения на антицепи, можно сформулировать как совершенство графа сравнимости частично упорядоченного множества, а теоремы Дилуорса, описывающие ширину частично упорядоченного множества в терминах разбиения на цепочки, можно сформулировать как совершенство дополнений этих графов. Таким образом, теорема о совершенном графе может быть использована для доказательства теоремы Дилуорса, опираясь на (более простое) доказательство теоремы Мирского, или наоборот.

## Алгоритмы для поиска совершенных графов

### Алгоритм Брона-Кербоша

Алгоритм Брона – Кербоша представляет собой алгоритм перечисления для поиска максимальных клик в неориентированном графе. То есть, он перечисляет все подмножества вершин с двумя свойствами, что каждая пара вершин в одном из перечисленных подмножеств соединена ребром, и ни одно из перечисленных подмножеств не может иметь никаких дополнительных вершин, добавленных к нему, при сохранении его полной связности.

Алгоритм Брона – Кербоша был разработан голландскими учеными Коэнрадом Броном и, опубликовавшим его описание в 1973 году. Хотя другие алгоритмы для решения проблемы клики имеют время выполнения которые теоретически лучше работают на входах, которые имеют несколько максимальных независимых наборов, алгоритм Брон-Кербоша и последующие его усовершенствования часто сообщаются как более эффективные на практике, чем альтернативы. Он хорошо известен и широко используется в прикладных областях алгоритмов графов, таких как вычислительная химия. Современный алгоритм Аккоюнлу (1973), хотя и представлен в разных терминах, можно рассматривать как то же, что и алгоритм Брона – Кербоша, поскольку он генерирует то же рекурсивное дерево поиска.

Базовая форма алгоритма Брон-Кербоша - рекурсивный алгоритм поиска с возвратом, который ищет все максимальные клики в заданном графе G. В более общем случае, учитывая три непересекающихся набора вершин R, P и X, он находит максимальные клики, которые включают все вершины в R, некоторые вершин в P и ни одной из вершин в X.

В каждом вызове алгоритма P и X являются непересекающимися множествами, объединение которых состоит из тех вершин, которые образуют клики при добавлении к R.

Другими словами, P ∪ X — это множество вершин, которые присоединяются к каждому элементу R. Когда P и X оба пусты, нет никаких дополнительных элементов, которые можно добавить к R, поэтому R является максимальной кликой и алгоритм выводит R. Рекурсия инициируется установкой R и X как пустое множество и P как набор вершин графа. В каждом рекурсивном вызове алгоритм по очереди рассматривает вершины в P; если таких вершин нет, он либо сообщает R как максимальную клику (если X пусто), либо выполняет возврат. Для каждой вершины v, выбранной из P, он выполняет рекурсивный вызов, в котором v добавляется к R и в котором P и X ограничиваются набором соседей N (v) v, который находит и сообщает обо всех расширениях клик R, содержащих v. Затем он перемещает v из P в X, чтобы исключить его из рассмотрения в будущих кликах, и переходит к следующей вершине в P.

Базовая форма алгоритма, описанная выше, неэффективна в случае графов с множеством немаксимальных клик: он выполняет рекурсивный вызов для каждой клики, максимальной или нет. Чтобы сэкономить время и позволить алгоритму более быстро возвращаться в ответвлениях поиска, которые не содержат максимальных клик, Брон и Кербош представили вариант алгоритма, включающий «поворотную вершину» u, выбранную из P (или, в более общем смысле, как более поздние исследователи реализовано, из P ⋃ X).

Любая максимальная клика должна включать либо u, либо одного из своих несоседей, иначе клика могла бы быть увеличена путем добавления к ней u. Следовательно, только u и его не-соседи должны быть проверены как варианты выбора для вершины v, которая добавляется к R при каждом рекурсивном вызове алгоритма. Если точка поворота выбрана так, чтобы минимизировать количество рекурсивных вызовов, выполняемых алгоритмом, то получается экономия времени работы. Поворотная версия алгоритма может иметь большое значение.

Изображение выглядит как круг, снимок экрана

Автоматически созданное описание

Рис. 3. Граф с пятью максимальными кликами: четырьмя ребрами и треугольником.

В показанном примере графа алгоритм изначально вызывается с R = Ø, P = {1,2,3,4,5,6} и X = Ø. Поворот u должен быть выбран как одна из вершин третьей степени, чтобы минимизировать количество рекурсивных вызовов; например, предположим, что u выбрана в качестве вершины 2. Тогда в P \ N (u) остаются три вершины: вершины 2, 4 и 6.

Итерация внутреннего цикла алгоритма для v = 2 выполняет рекурсивный вызов алгоритма с R = {2}, P = {1,3,5} и X = Ø. В пределах этого рекурсивного вызова один из 1 или 5 будет выбрано в качестве оси поворота, и там будут два вторых уровнем рекурсивных вызовов, один для вершины 3 и другие для какого бы вершина не была выбрана в качестве оси поворота. Эти два вызова, в конечном итоге, сообщат о двух кликах {1,2,5} и {2,3}. После возврата из этих рекурсивных вызовов вершина 2 добавляется к X и удаляется из P.

Итерация внутреннего цикла алгоритма для v = 4 выполняет рекурсивный вызов алгоритма с R = {4}, P = {3,5,6} и X = Ø (хотя вершина 2 принадлежит набору X во внешнем вызове алгоритма, она не является соседом v и исключается из подмножества X, переданного в рекурсивный вызов). Этот рекурсивный вызов завершится тремя рекурсивными вызовами второго уровня алгоритма, который сообщает о трех кликах {3,4}, {4,5} и {4,6}. Затем вершина 4 добавляется к X и удаляется из P.

На третьей и последней итерации внутреннего цикла алгоритма для v = 6 выполняется рекурсивный вызов алгоритма с R = { 6}, P = Ø и X = {4}. Поскольку в этом рекурсивном вызове P пусто, а X непусто, он немедленно выполняет возврат, не сообщая о каких-либо дополнительных кликах, так как не может быть максимальной клики, которая включает вершину 6 и исключает вершину 4.

Дерево вызовов для алгоритма, следовательно, выглядит так:

BronKerbosch2 (Ø, {1,2,3,4,5,6}, Ø) BronKerbosch2 ({2}, {1,3,5}, Ø) BronKerbosch2 ( {2,3}, Ø, Ø): выход {2, 3} BronKerbosch2 ({2,5}, {1}, Ø) BronKerbosch2 ({1,2,5}, Ø, Ø): выход {1, 2,5} BronKerbosch2 ({4}, {3,5,6}, Ø) BronKerbosch2 ({3,4}, Ø, Ø): выход {3,4} BronKerbosch2 ({4,5}, Ø, Ø): выход {4,5} BronKerbosch2 ({4,6}, Ø, Ø): выход {4,6} BronKerbosch2 ({6}, Ø, {4}): нет выхода

График в примере имеет вырождение два; один возможный порядок вырождения - 6,4,3,1,2,5. Если к вершинам применяется версия алгоритма Брон-Кербоша с упорядочением вершин, то в этом порядке дерево вызовов выглядит как: BronKerbosch3 (G) BronKerbosch2 ({6}, {4}, Ø) BronKerbosch2 ({6,4}, Ø, Ø): выход {6,4} BronKerbosch2 ({4}, {3,5}, {6}) BronKerbosch2 ({4,3}, Ø, Ø): выход {4, 3} BronKerbosch2 ({4,5}, Ø, Ø): вывод {4,5} BronKerbosch2 ({3}, {2}, {4}) BronKerbosch2 ({3,2}, Ø, Ø): вывод {3,2} BronKerbosch2 ({1}, {2,5}, Ø) BronKerbosch2 ({1,2}, {5}, Ø) BronKerbosch2 ({1,2,5}, Ø, Ø): выход { 1,2,5} BronKerbosch2 ({2}, {5}, {1,3}): нет вывода BronKerbosch2 ({5}, Ø, {1,2,4}): нет вывода

### Алгоритм Эдмондса-Карпа

Рассмотрим следующий алгоритм, который является модернизацией алгоритма Форда-Фалкерсона – алгоритм Эдмондса-Карпа, который был разработан в 1971 году Джеком Эдмондсом и Ричардом Карпом.

Недостатком алгоритма Форда-Фалкерсона является его зависимость от величины ответа, однако Эдмондсу и Карпу удалось справиться с этой проблемой, создав алгоритм, работающий за полиномиальное время.

Алгоритм Эдмондса-Карпа не сильно отличается от алгоритма Форда-Фалкерсона: он также находит дополняющие пути, пока они существуют и добавляет поток вдоль этих путей. Так как дополняющие пути являются простыми (то есть не содержат циклов), то их длина строго меньше количества вершин. Также известно, что изначально кратчайшие расстояния до каждой вершины больше нуля. Следовательно к тому моменту, когда вершина станет недостижимой из истока, кратчайшее расстояние до нее не превысит. А значит, что ребро могло стать критическим не более раз.

Таким образом, алгоритм Эдмондса-Карпа является первым и самым простым улучшением алгоритма Форда-Фалкерсона, который уже не зависит от величины ответа и является полностью полиномиальным. Однако, как будет рассмотрено далее, существуют и другие более эффективные модернизации алгоритмов Форда-Фалкерсона и Эдмондса-Карпа

### Алгоритм Ловаша

Алгоритм Ловаша, известный также как "алгоритм разреза и связи" (Cutting Plane Algorithm), является эффективным методом для реализации совершенных графов. Этот алгоритм был разработан Ласло Ловашем в 1972 году и представляет собой комбинаторный алгоритм, основанный на линейном программировании и теории множеств.

Идея алгоритма Ловаша состоит в последовательном добавлении неравенств, называемых "разрезами" (cuts), для описания совершенных графов. Каждый разрез определяет ограничение на множества вершин графа, которые не могут быть соединены ребром. Алгоритм стремится построить систему неравенств, которая задает точное описание всех совершенных графов.

Процесс алгоритма начинается с пустого множества разрезов. Затем на каждой итерации алгоритма выбирается совершенный граф, который не может быть описан текущим набором разрезов, и добавляется новый разрез, который уточняет ограничения. Этот процесс повторяется до тех пор, пока набор разрезов не станет достаточно полным для описания всех совершенных графов.

Для построения разрезов используются линейные программы и методы оптимизации. В каждой итерации алгоритма Ловаша решается оптимизационная задача, которая ищет совершенный граф, не удовлетворяющий текущим разрезам. Если такой граф найден, то из него извлекается новый разрез, который добавляется в систему. Этот процесс продолжается до тех пор, пока не будет достигнута точная характеристика совершенных графов.

Алгоритм Ловаша имеет важное значение в теории совершенных графов и находит применение в различных областях, таких как комбинаторика, оптимизация и математическое программирование. Он позволяет строить систему неравенств, которая точно определяет совершенные графы, и является основой для разработки других алгоритмов и методов, связанных с этой темой.

## Реализация алгоритма для нахождения совершенных графов

### Методы программы

Зная, что такое совершенный граф, хроматическое число и алгоритм Брона-Кербоша можно приступать к реализации алгоритма для нахождения совершенных графов.

Выполнение алгоритмов программы можно разбить на несколько этапов:

1. Создается класс Graph, который содержит три атрибута: V, graph, и color\_arr. V — это количество вершин в графе, graph - матрица смежности для хранения ребер графа, и color\_arr - массив для хранения цветов вершин.
2. Метод add\_edge используется для добавления ребра между двумя вершинами u и v в графе. Он устанавливает соответствующие элементы матрицы смежности в 1.
3. Метод bron\_kerbosch реализует алгоритм Брона-Кербоша для поиска всех клик в графе. Он использует стек для обхода и поиска клик. r - текущая клика, p - множество вершин, которые еще не добавлены в текущую клику, x - множество вершин, которые уже были проверены и исключены из текущей клики, cliques - список найденных клик.
4. Метод find\_cliques вызывает bron\_kerbosch для поиска всех клик в графе и возвращает список найденных клик.
5. Метод chromatic\_number вычисляет хроматическое число графа. Он применяет алгоритм жадной раскраски вершин графа, чтобы каждая вершина имела уникальный цвет. Метод возвращает хроматическое число графа.
6. Метод is\_perfect определяет, является ли граф совершенным. Он вызывает find\_cliques для получения списка клик и chromatic\_number для определения хроматического числа графа. Затем он проверяет, что каждая клика имеет размер равный хроматическому числу. Если условие не выполняется хотя бы для одной клики, метод возвращает False. В противном случае возвращает True.

### Алгоритм программы

Эта программа реализует несколько алгоритмов для работы с графами. Давайте подробнее опишем каждый из них:

1. Класс Graph:

* Инициализация объекта графа: В конструкторе класса Graph передается количество вершин vertices, инициализируются атрибуты V (количество вершин), graph (матрица смежности) и color\_arr (массив цветов вершин).
* Метод add\_edge: этот метод используется для добавления ребра между двумя вершинами в графе. Он обновляет матрицу смежности graph путем установки соответствующих элементов в 1.
* Метод bron\_kerbosch: реализует алгоритм Брона-Кербоша для поиска всех клик в графе. Он использует рекурсию и стек для обхода и поиска клик. Метод принимает текущую клику r, множество вершин, которые еще не добавлены в текущую клику p, множество вершин, которые уже были проверены и исключены из текущей клики x, и список найденных клик cliques.
* Метод find\_cliques: вызывает метод bron\_kerbosch для поиска всех клик в графе и возвращает список найденных клик.
* Метод chromatic\_number: рассчитывает хроматическое число графа. Он использует жадный алгоритм раскраски вершин графа, чтобы каждая вершина имела уникальный цвет. Метод возвращает хроматическое число графа.
* Метод is\_perfect: проверяет, является ли граф совершенным. Он вызывает метод find\_cliques для получения списка клик и метод chromatic\_number для определения хроматического числа графа. Затем он проверяет, что каждая клика имеет размер, равный хроматическому числу. Если условие не выполняется хотя бы для одной клики, метод возвращает False. В противном случае возвращает True.

2. Основная часть программы:

* Чтение количества вершин графа от пользователя.
* Создание объекта Graph с заданным количеством вершин.
* Чтение количества ребер графа от пользователя. Ввод ребер графа от пользователя и добавление их в объект Graph.
* Вывод списка найденных клик в графе с помощью метода find\_cliques.
* Проверка, является ли граф совершенным, с помощью метода is\_perfect.
* Вывод хроматического числа графа с помощью метода chromatic\_number.

Таким образом, программа позволяет пользователю вводить графические данные и выполняет различные операции на основе введенных данных, такие как поиск клик, определение совершенности графа и вычисление хроматического числа графа.

# Заключение

В данной курсовой работе было исследовано понятие совершенных графов и их свойства.

Совершенные графы представляют собой уникальный класс графов, в которых каждая клика имеет размер, равный хроматическому числу графа. Изучение совершенных графов имеет важное значение в различных областях, таких как комбинаторика, оптимизация и алгоритмы.

В ходе работы было разработано программное решение на основе алгоритмов для работы с графами. Был реализован алгоритм Брона-Кербоша для поиска всех клик в графе, а также жадный алгоритм раскраски вершин для определения хроматического числа графа. Эти алгоритмы были включены в класс Graph, который предоставляет удобный интерфейс для работы с графами. Написание алгоритма, указанного выше, представляло определенные сложности. Реализация этого алгоритма требовала внимательного анализа и понимания логики алгоритма.

В целом, написание алгоритма для работы с совершенными графами оказалось трудоемким, но интересным процессом. Это позволило лучше понять и изучить основные свойства совершенных графов и применить полученные знания на практике.

Дальнейшие исследования в области совершенных графов могут включать оптимизацию алгоритмов, разработку новых методов и решений для работы с такими графами, а также исследование различных классов совершенных графов и их свойств.

# Список литературы

* 1. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Алгоритмы: построение и анализ = IntroductiontoAlgorithms / Под ред. И. В. Красикова. — 2-е изд. — М.: Вильямс, 2005. — 1296 с. — ISBN 5-8459-0857-4.
  2. Martin Charles Golumbic - "Algorithmic Graph Theory and Perfect Graphs" (Elsevier Science, 2004).
  3. Maria Chudnovsky, Neil Robertson, Paul Seymour, Robin Thomas - "The Strong Perfect Graph Theorem" (Journal of the American Mathematical Society, Vol. 21, No. 2, 2008).
  4. Annegret Wagler - "Perfect Graphs" (London Mathematical Society Student Texts, Vol. 88, 2017).
  5. Douglas B. West - "Introduction to Graph Theory" (Pearson, 2001)

# Приложение

Код программы на языке Python.

class Graph:

def \_\_init\_\_(self, vertices):

self.V = vertices

self.graph = [[0 for column in range(vertices)] for row in range(vertices)]

self.color\_arr = [0 for i in range(vertices)]

def add\_edge(self, u, v):

self.graph[u - 1][v - 1] = 1

self.graph[v - 1][u - 1] = 1

def bron\_kerbosch(self, r, p, x, cliques):

stack = [(r, p, x)]

while stack:

r, p, x = stack.pop()

if not p and not x:

cliques.append(r)

else:

for vertex in p[:]:

r\_new = r + [vertex]

p\_new = [v for v in p if self.graph[vertex - 1][v - 1]]

x\_new = [v for v in x if self.graph[vertex - 1][v - 1]]

stack.append((r\_new, p\_new, x\_new))

def find\_cliques(self):

cliques = []

self.bron\_kerbosch([], [i+1 for i in range(self.V)], [], cliques)

return cliques

def chromatic\_number(self):

self.color\_arr[0] = 0

for u in range(1, self.V):

self.color\_arr[u] = -1

for u in range(1, self.V):

colored\_neighbors = [False] \* self.V

for i in range(self.V):

if self.graph[u][i] and self.color\_arr[i] != -1:

colored\_neighbors[self.color\_arr[i]] = True

color = 0

while colored\_neighbors[color]:

color += 1

self.color\_arr[u] = color

return max(self.color\_arr) + 1

def is\_perfect(self):

cliques = self.find\_cliques()

chromatic\_number = self.chromatic\_number()

for clique in cliques:

if len(clique) != chromatic\_number:

return False

return True

if \_\_name\_\_ == "\_\_main\_\_":

while True:

try:

vertices = int(input("Введите количество вершин в графе: "))

if vertices <= 0:

raise ValueError

break

except ValueError:

print("Некорректное значение. Введите положительное целое число.")

g = Graph(vertices)

while True:

try:

edges = int(input("Введите количество ребер: "))

if edges < 0:

raise ValueError

break

except ValueError:

print("Некорректное значение. Введите неотрицательное целое число.")

print("Введите ребра в формате 'u v':")

for \_ in range(edges):

while True:

try:

u, v = map(int, input().split())

if u <= 0 or v <= 0 or u > vertices or v > vertices:

raise ValueError

break

except ValueError:

print("Некорректное значение. Введите два положительных целых числа в диапазоне от 1 до", vertices)

g.add\_edge(u, v)

print("Клики в графе:")

cliques = g.find\_cliques()

if not cliques:

print("В графе нет клик.")

else:

for clique in cliques:

print(clique)

if g.is\_perfect():

print("Граф является совершенным.")

else:

print("Граф не является совершенным.")

print("Хроматическое число графа: ", g.chromatic\_number())