Тема 3.5. Простейшие алгоритмы теории графов

Аннотация: Простейшие алгоритмы теории графов. Путь минимальной суммарной длины во взвешенном графе. Алгоритм Дейкстры. Задача коммивояжера. Венгерский алгоритм построения совершенного паросочетания (задача об оптимальном назначении). Раскраска графа.

Понятие двудольного графа. Паросочетания. Венгерский алгоритм

Понятие двудольного графа

Двудольным называется граф, множество вершин которого можно разбить на два непересекающихся подмножества V_1 и V_2 (на две ***доли***) и при этом каждое ребро графа соединяет какую-либо вершину из V_1 с какой-либо вершиной из V_2, но никакие две вершины из одного множества не являются смежными.

Теорема (Д.Кенинга)

Граф является двудольным тогда и только тогда, когда в нем отсутствуют циклы нечетной длины.

Доказательство:

- **Необходимость.** Пусть $G = \langle V, E \rangle$ — двудольный граф, C — один из его циклов длины k. Пройдем все ребра этого цикла в той последовательности, в какой они в нем расположены, начиная с некоторой вершины v. Так как концы каждого ребра лежат в разных долях, то k - четное число.

Достаточность. Не теряя общности, можно рассматривать только связные графы. Пусть связный граф $G = \langle V, E \rangle$ не имеет циклов нечетной длины.

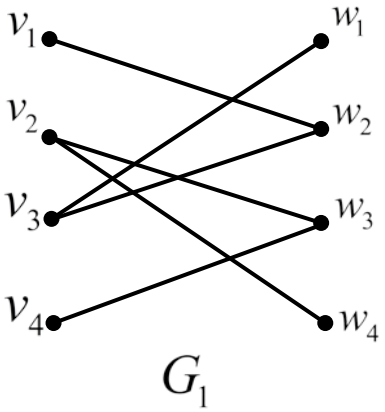
Рассмотрим произвольную вершину v_0 и построим разбиение вершин графа G на два класса V_1 и V_2 следующим образом: к классу V_1 отнесем вершину v_0 и любую такую вершину v^*, что расстояние между v^* и v_0 четное, к классу V_2 отнесем любую такую вершину v^*, что расстояние между v^* и v_0 нечетное. Остается показать, что любые две вершины из множества V_1 или любые две вершины из множества V_2 не смежны.

Пусть, например, существуют две смежные вершины u и v,

входящие в один класс. Тогда ни одна из них не совпадает с вершиной v_0, так как вторая в этом случае принадлежала бы другому классу (расстояние между смежными вершинами равно 1). Пусть далее C_1 - кратчайшая цепь, соединяющая v_0 и u, а C_2 - кратчайшая цепь соединяющая v_0 и v. Обозначим через $v_1, v_2, ..., v_p$ общие вершины, считая от вершины v_0. Поскольку C_1 и C_2 - кратчайшие цепи, то их части от вершины от вершины v_0 до вершины v_1 имеют одинаковые длины. То же самое можно сказать и о частях цепей C_1 и C_2 от любой вершины v_i до вершины v_{i+1}. Поэтому части цепей от вершины v_p до вершин u и v имеют одинаковые четности. Но тогда объединение этих частей и ребра (u,v) является циклом нечетной длины, что запрещено условиями теоремы. Теорема доказана. ◅

Полным двудольным графом называется двудольный граф, в котором каждая вершина из V_1 смежна с каждой вершиной из V_2.

Пример: На рисунке ниже G_1 - двудольный граф, не являющий полным. $V_1 = \{v_1, v_2, v_3, v_4\}$, $V_2 = \{w_1, w_2, w_3, w_4\}$.

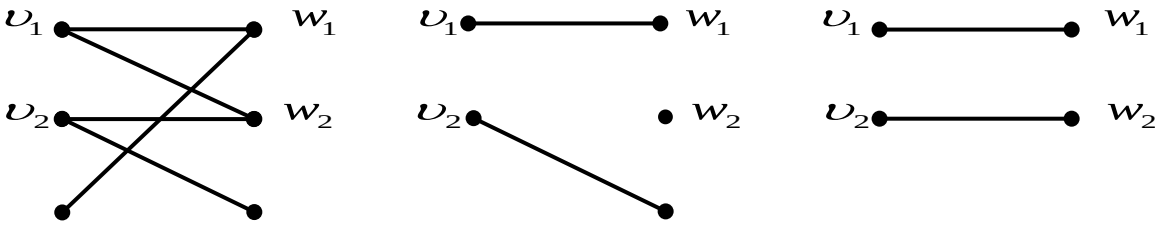


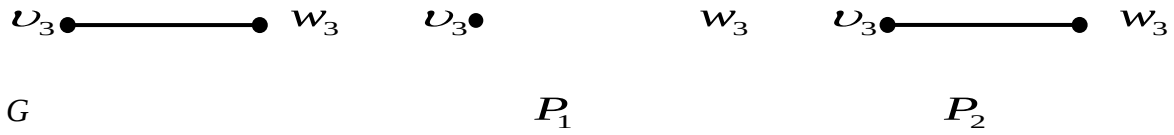
G_1

Паросочетания

Паросочетанием в двудольном графе G называется подмножество попарно несмежных ребер графа.

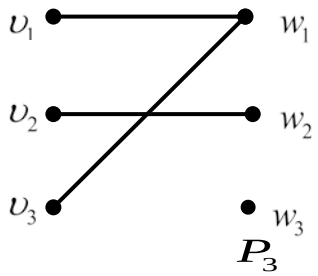
Пример: Двудольный граф G задан графически:





G P_1 P_2

P_1 и P_2 - паросочетания в графе G.



P_3

Граф P_3 не является паросочетанием в данном графе, так как ребра (υ_1, w_1) и (υ_3, w_1) смежны.

Паросочетание в двудольном графе называется ***полным***, если к нему нельзя добавить ни одного ребра, несмежного с ребрами паросочетания.

Паросочетание в двудольном графе называется ***максимальным***, если никакое другое паросочетание в не содержит ребер больше, чем данное.

Пусть $G = \langle V, E \rangle$ - двудольный граф, множество вершин которого можно разбить на две доли V_1 и V_2. Паросочетание в графе G называется ***совершенным***, если для всякой вершины из V_1 найдется инцидентное ей ребро, то есть.

Максимальное паросочетание всегда полное. Обратное неверно. Совершенное паросочетание всегда максимальное. Обратное неверно.

Венгерский алгоритм

Рассмотрим задачу: Пусть имеется n работников и m работ. Известно, какие работы может выполнять каждый из рабочих. Требуется распределить работу между работниками так, чтобы наибольшее количество людей получило работу, которую они могут выполнять. Все ли работники получат работу?

Эту задачу можно решать при помощи графов. Работникам соответствуют вершины $v_i \in V_1$, а работам вершины $w_j \left(w_j \in V_2 \right)$.- двудольного графа $G = \langle V, E \rangle$, где $V = V_1 \cup V_2$.

Если *i*-тый рабочий выполняет *j*-тую работу, то в графе существует ребро (υ_i, w_j). Если *i*-тый рабочий не может выполнять *j*-тую работу, то ребра (υ_i, w_j) в графе нет.

Т.е. задачу можно сформулировть так: найти максимальное паросочетание в двудольном графе. Существует ли в графе совершенное сочетание?

Решение этой задачи удобно находить, используя *алгоритм отыскания максимального паросочетания,* так называемый *венгерский алгоритм*, названный так в честь Гарольда Куна, впервые предложившего данный алгоритм.

Ответ на вопрос о существовании совершенного паросочетания дает нам теорема Ф. Холла. Эта теорема была получена для решения задачи о свадьбах: рассмотрим некоторое конечное множество юношей, каждый из которых знаком с несколькими девушками; спрашивается, при каких условиях можно женить юношей так, чтобы каждый из них женился на знакомой ему девушке?

Теорема (Ф. Холл)

Решение задачи о свадьбах существует тогда и только тогда, если любые k юношей из данного множества знакомы в совокупности по меньшей мере с k девушками $(1 \le k \le m)$.

➢ **Доказательство:**

Необходимость условия очевидна.

Достаточность. Воспользуемся индукцией и допустим, что утверждение справедливо, если число юношей меньше m. (Ясно, что при $m = 1$ теорема верна.) Предположим теперь, что число юношей равно m, и рассмотрим два возможных случая.

1. Сначала будем считать, что любые k юношей $(1 \le k \le m)$ в совокупности знакомы по меньшей мере с $k+1$ девушками (т.е. что наше условие всегда выполняется «с одной лишней девушкой»). Тогда, если взять любого юношу и женить его на любой знакомой ему девушке, для других $m-1$ юношей останется верным первоначальное условие. По предположению индукции мы можем женить этих $m-1$ юношей; тем самым доказательство в первом случае завершено.

2. Предположим теперь, что имеются k юношей $(k < m)$, которые в совокупности знакомы ровно с k девушками. По

индуктивному предположению этих k юношей можно женить. Остаются еще $m-k$ юношей, но любые h из них ($1 \le h \le m-k$) должны быть знакомы, по меньшей мере, с h девушками из оставшихся, поскольку в противном случае эти h юношей вместе с уже выбранными k юношами будут знакомы меньше, чем с $h+k$ девушками, а это противоречит нашему предположению. Следовательно, для этих $m-k$ юношей выполнено первоначальное условие, и по предположению индукции мы можем их женить так, чтобы каждый был счастлив. Теорема доказана. ◅

Теорему Холла можно также сформулировать на языке паросочетаний:

Теорема (Ф. Холл)

Совершенное паросочетание в двудольном графе существует тогда, и только тогда, когда число вершин во множестве $S \subset V$ не превосходит количества смежных с ней вершин

Вариацией задачи является задача о назначениях: n работников выполняют n работ. Каждый i-ый рабочий выполняет j-ую работу с эффективностью $c_{ij} \ge 0$. Требуется распределить работу между работниками таким образом, чтобы суммарная эффективность работ была бы максимальной.

Очевидно, что данную задачу можно свести к построению совершенного паросочетания в двудольном графе.

Венрегский алгоритм решения задачи оптимального назначения:

1. Если задача решается на максимум, то необходимо найти максимальный элемент, его же вычесть из каждого элемента матрицы и умножить всю матрицу на -1 (или вычесть из максимального элемента каждый элемент матрицы, что, очевидно, то же самое). Если задача решается на минимум, то этот шаг необходимо пропустить.

2. Если задача решается на минимум, то необходимо начинать с этого шага. Редукция матрицы по строкам (ищем минимальный элемент в каждой строке и вычитаем его из каждого элемента соответственно). Также проводим редукцию по столбцам:

Цель редукции по строкам и столбцам – сделать хотя бы один ноль в каждой строке и каждом столбце.

3. Теперь нам нужно выбрать в каждой строке или каждом столбце ровно один ноль (т.е. когда выбрали ноль, то остальные нули

в этой строке или в этом столбце уже не берем в расчет). Если это можно сделать – то выбранные позиции дают нам ответ. Если так сделать нельзя, то переходим к следующему шагу.

4. Вычеркиваем строки и столбцы, которые содержат нулевые элементы (**Количество вычеркиваний должно быть минимальным!**). Среди оставшихся элементов ищем минимальный, вычитаем его из оставшихся элементов (которые не зачеркнуты) и прибавляем к элементам, которые расположены на пересечении вычеркнутых строк и столбцов.

Повторяем шаги 3 и 4 пока не получим решение.

Пример: При помощи венгерского алгоритма решите задачу оптимального назначения на максимум для двудольного графа, заданного матрицей:

$$C = \begin{pmatrix} 19 & 11 & 12 & 14 \\ 4 & 15 & 5 & 11 \\ 6 & 5 & 3 & 8 \\ 15 & 4 & 6 & 0 \end{pmatrix}.$$

Решение:

Поскольку решается задача на максимум, то нам потребуется сделать один дополнительный шаг: найдем максимальный элемент – 19 и отнимем каждый элемент от него. Получим:

$$C = \begin{pmatrix} 0 & 8 & 7 & 5 \\ 15 & 4 & 14 & 8 \\ 13 & 14 & 16 & 11 \\ 4 & 15 & 13 & 19 \end{pmatrix}.$$

Задача свелась к решению задачи на минимум.

Теперь проведем редукцию по строкам (найдем в каждой строке наименьший элемент и вычтем его из всех элементов данной строки):

$$C = \begin{pmatrix} 0 & 8 & 7 & 5 \\ 11 & 0 & 10 & 4 \\ 2 & 3 & 5 & 0 \\ 0 & 11 & 9 & 15 \end{pmatrix}.$$

Теперь проведем редукцию по столбцам:

$$C = \begin{pmatrix} 0 & 8 & 2 & 5 \\ 11 & 0 & 5 & 4 \\ 2 & 3 & 0 & 0 \\ 0 & 11 & 4 & 15 \end{pmatrix}.$$

Цель редукции по строкам и столбцам – сделать хотя бы один ноль в каждой строке и каждом столбце.

Теперь нам нужно выбрать в каждой строке или каждом столбце ровно один ноль. Но здесь это сделать не возможно, т.к., например в последней строке можно выбрать только один ноль (в первом столбце), но он блокирует ноль в первой строке, а больше нулей в ней нет (аналогичная ситуация, например и с нулями в третьей строке).

Поскольку нули выбрать не получается нам нужно вычеркнуть **все** нули **минимальным** (это важно!) количеством горизонтальных и/или горизонтальных линий:

$$C=\begin{pmatrix} \cancel{0} & \cancel{8} & 2 & 5 \\ \cancel{11} & \cancel{0} & 5 & 4 \\ \cancel{2} & \cancel{3} & \cancel{0} & \cancel{0} \\ \cancel{0} & \cancel{11} & 4 & 15 \end{pmatrix}$$

Среди всех невычеркнутых элементов находим минимальный. Отнимаем его от всех невычеркнутых элементов и прибавляем в местах пересечения линий, те элементы, через которые проходит только одна линия не трогаем. Получим:

$$C=\begin{pmatrix} 0 & 8 & 0 & 3 \\ 11 & 0 & 3 & 2 \\ 4 & 5 & 0 & 0 \\ 0 & 11 & 2 & 13 \end{pmatrix}.$$

Теперь смотрим, можно ли здесь выбрать нули (в каждой строке или каждом столбце ровно один ноль). Такое возможно:

$$C=\begin{pmatrix} 0 & 8 & \boxed{0} & 3 \\ 11 & \boxed{0} & 3 & 2 \\ 4 & 5 & 0 & \boxed{0} \\ \boxed{0} & 11 & 2 & 13 \end{pmatrix}$$

Тогда ответ:

$$C=\begin{pmatrix} 19 & 11 & \boxed{12} & 14 \\ 4 & \boxed{15} & 5 & 11 \\ 6 & 5 & 3 & \boxed{8} \\ \boxed{15} & 4 & 6 & 0 \end{pmatrix}$$

Алгоритм Дейкстры в графическом виде

Ранее мы рассмотрели этот алгоритм в матричной форме. Теперь рассмотрим как он реализуется на графе. Сначала напомним основную идею.

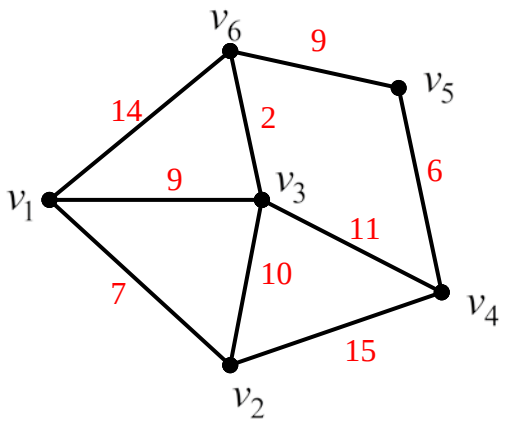
Основная идея

Его идея следующая: перед каждым этапом известно множество **отмеченных вершин** S, для которых кратчайшие пути найдены ранее; тогда на очередном этапе к нему добавляется вершина v, с самым коротким путем из v_0, проходящим по множеству S; после этого пересчитываются длины кратчайших путей из v_0 в оставшиеся вершины из $V \setminus S$ с учетом новой вершины v. Длина текущего кратчайшего пути из v_0 в v, проходящего по множеству S, заносится в строку $D^{(k)}$. В конце работы в этой строке отыскиваются длины соответствующих кратчайших путей.

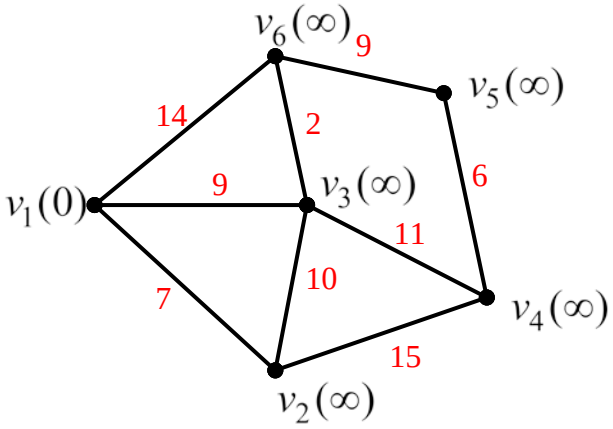
Графическая реализация

Рассмотрим графическую реализацию на примере.

Пример: Пусть требуется найти кратчайшие расстояния от 1-й вершины до всех остальных.



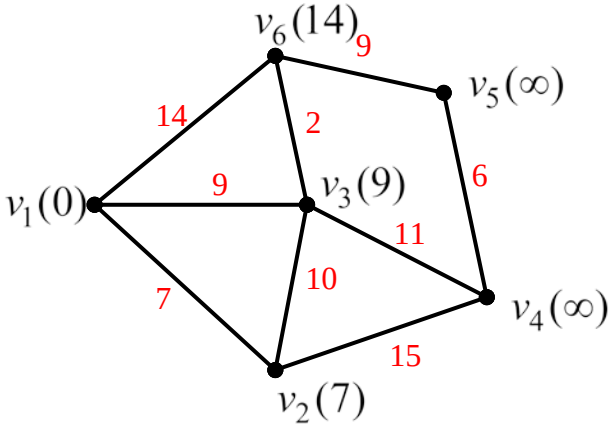
Инициализация: Метка самой вершины 1 полагается равной 0, метки остальных вершин – бесконечность (это вариант для составления программы для компьютера, для решения задачи на графах берут сразу расстояния до соседних вершин, т.е. то, что получится на следующем шаге).



Шаг 1. Соседи вершины с минимальной меткой (вершина v_1 с меткой 0) являются вершины 2, 3 и 6. Обходим соседей вершины по очереди.

Первый сосед v_1 – вершина v_2, потому что длина пути до неё минимальна. Длина пути $\rho(v_1,v_2)$ равна сумме кратчайшего расстояния до вершины v_1 (значению её метки, т.е. 0) и длины ребра, (v_1,v_2), то есть 0 + 7 = 7. Это меньше текущей метки v_2 (∞), поэтому новая метка 2-й вершины равна 7.

Аналогично находим длины пути для всех других соседей (вершины 3 и 6).



Все соседи v_1 проверены. Текущее минимальное расстояние до v_1 считается окончательным и пересмотру не подлежит. Вершина v_1 отмечается как посещенная (возьмем ее в квадратные скобки).

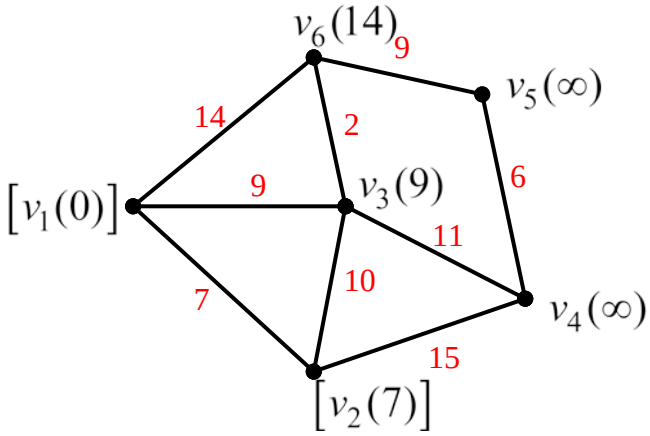
Шаг 2. Шаг 1 алгоритма повторяется. Снова находим «ближайшую» из непосещенных вершин. Это v_2 с меткой 7.

Снова пытаемся уменьшить метки соседей выбранной вершины, пытаясь пройти в них через v_2. Соседями вершины 2 являются вершины v_1, v_3 и v_4.

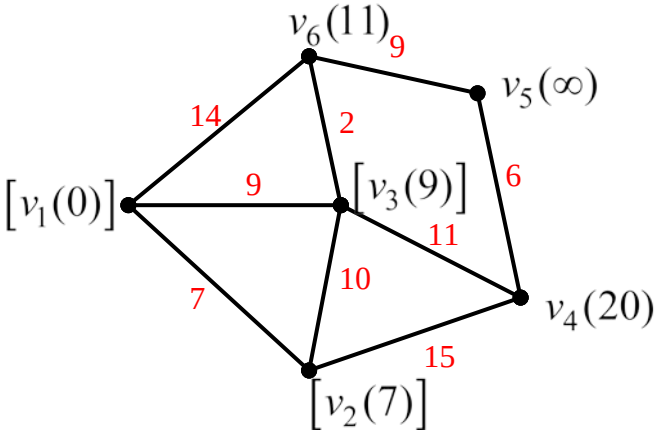
Вершина v_1 уже посещена. Следующий сосед вершины v_2 — вершина v_3, так как имеет минимальную метку из вершин, отмеченных как не посещённые. Если идти в неё через 2, то длина такого пути будет равна 17 (7 + 10 = 17). Но текущая метка третьей вершины равна 9, а $9 < 17$, поэтому метка не меняется.

Ещё один сосед v_2 — вершина v_4. Если идти в неё через 2-ю, то длина такого пути будет равна 22 (7 + 15 = 22). Поскольку $22 < \infty$, устанавливаем метку v_4 равной 22.

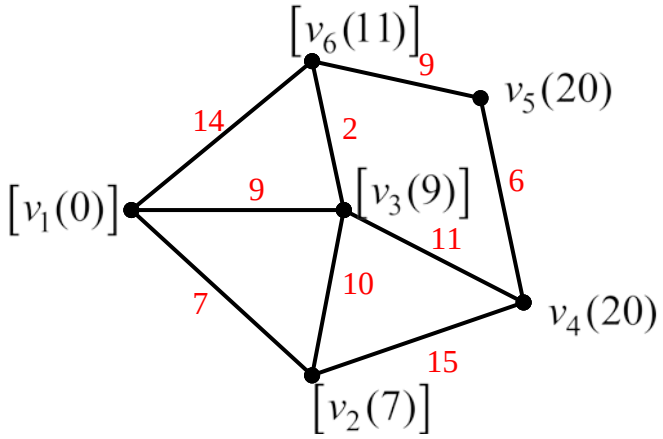
Все соседи вершины 2 просмотрены, помечаем её как посещенную.



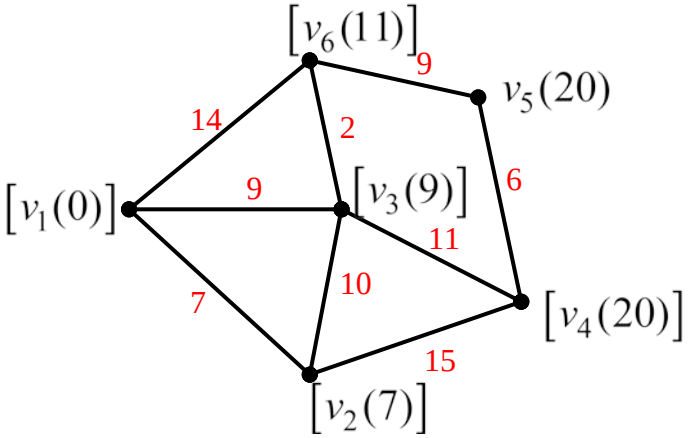
Шаг 3. Повторяем шаг алгоритма, выбрав вершину v_3. Получим:



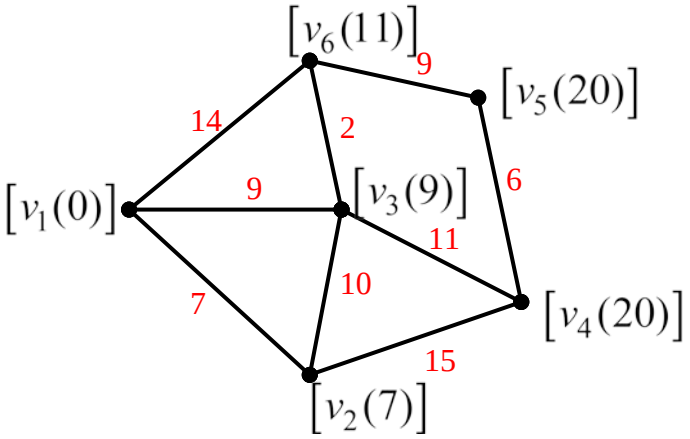
Шаг 4:



Шаг 5:



Шаг 6:



Таким образом, кратчайшие пути из вершины 1: (0,7,9,20,20,11).

Вывод кратчайшего пути получается разворачивая данную схему в обратную сторону. Рассматриваем конечную вершину (v_5 в нашей предыдущей схеме), и для всех вершин, с которой она связана, находим длину пути, вычитая вес соответствующего ребра из длины пути конечной вершины.

Так, v_5 имеет длину пути 20. Она связана с вершинами 6 и 4.

Для v_6 получим вес 20 — 9 = 11 (совпал).

Для v_4 получим вес 20 — 6 = 14 (не совпал).

Если в результате мы получим значение, которое совпадает с длиной пути рассматриваемой вершины (в данном случае v_6), то именно из нее был осуществлен переход в конечную вершину. Отмечаем эту вершину на искомом пути.

Далее определяем ребро, через которое мы попали в v_6. И так пока не дойдем до начала.

Если в результате такого обхода у нас на каком-то шаге совпадут значения для нескольких вершин, то можно взять любую из них — несколько путей будут иметь одинаковую длину.

Гамильтонов цикл и задача коммивояжера

Формулировка задачи коммивояжера

С понятием гамильтоновых графов связана, так называемая, задача коммивояжера. Задача коммивояжёра (англ. TSP - Travelling salesman problem) — одна из самых известных задач комбинаторной оптимизации, заключающаяся в поиске самого выгодного маршрута, проходящего через указанные города хотя бы по одному разу с последующим возвратом в исходный город. В условиях задачи указываются критерий выгодности маршрута (кратчайший, самый дешёвый) и соответствующие матрицы расстояний, стоимости и тому подобного. Как правило, указывается, что маршрут должен проходить через каждый город только один раз — в таком случае выбор осуществляется среди гамильтоновых циклов.

Точно неизвестно, когда проблему коммивояжера исследовали впервые. Однако, известна изданная в 1832 году книга с названием «Коммивояжёр — как он должен вести себя и что должен делать для того, чтобы доставлять товар и иметь успех в своих делах — советы старого курьера», в которой описана проблема, но математический аппарат для её решения не применяется. Первые упоминания в качестве математической задачи на оптимизацию принадлежат Карлу Менгеру, который сформулировал её на математическом коллоквиуме в 1930 году так: «Мы называем проблемой посыльного (поскольку этот вопрос возникает у каждого почтальона, в частности, её решают многие путешественники) задачу найти кратчайший путь между конечным множеством мест, расстояние между которыми известно».

Сеть дорог может быть представлена взвешенным связным графом следующим образом. Каждый город представляется

вершиной, а каждая дорога, соединяющая два города, представляется ребром, соединяющим соответствующие две вершины, причём вес ребра равен длине данной дороги. Как правило, оказывается целесообразным несколько видоизменить исходную проблему следующим образом. Связный граф, описанный выше, заменяется полным графом с одной вершиной для каждого города. Напомним, что полный граф – это такой граф, у которого имеется единственное ребро, соединяющее каждую пару отличных вершин. Каждому ребру такого графа приписывается вес, равный наикратчайшему расстоянию между соответствующими городами, в соответствии с используемой сетью дорог. Эти самые короткие расстояния могут быть найдены на основании применения алгоритма Декстра к первоначальному графу.

Не каждый граф имеет цикл Гамильтона; однако, как было доказано ранее, каждый полный граф, имеющий, по крайней мере, три вершины, имеет цикл Гамильтона. Так как наши графы конечны, то может иметься только конечное число циклов Гамильтона, и, следовательно, среди них должен быть и минимальный, т.е. проблема коммивояжёра имеет решение.

Классификация алгоритмов по их вычислительной сложности

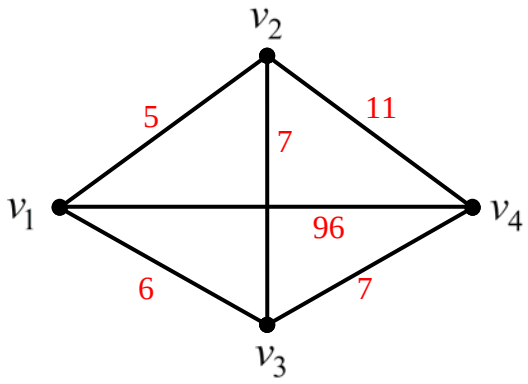
В 1965 Дж. Эдмондс и А. Кобхэм представили широкую классификацию алгоритмов, которые выполняются в полиномиальной и в показательной зависимости от времени. Считается, что алгоритм в вычислительном отношении эффективен, если его количество вычислений при его реализации не превышает некоторую степень числа n, например, n^k для некоторого целого числа k. Это так называемые алгоритмы полиномиального времени. Альтернативно, алгоритм в вычислительном отношении неэффективен, если его стоимость – показательная функция n; т.е. стоимость зависит от a^n для некоторого действительного числа $a > 1$. Такие алгоритмы называются алгоритмами показательного времени.

Необходимо отметить, что все известные алгоритмы, дающие точно минимальный цикл Гамильтона, в вычислительном отношении являются неэффективными относительно числа вершин графа (алгоритмы показательного времени). Тем не менее, известны различные эффективные алгоритмы, которые дают приблизительное решение. Другими словами, они обеспечивают циклы Гамильтона, чей вес оказывается близким к минимально возможному, однако при

этом нельзя быть уверенным в том, что данный вес является самым минимальным из возможных.

Рассмотрим так называемый **жадный алгоритм** – на каждом шаге идем в самую ближайшую вершину.

Пример: решить задачу коммивояжера для графа



Решение:

Решая данную задачу жадным алгоритмом мы должны идти следующим образом: из v_1 в v_2 - самый короткий из v_1 (длина 5), затем из v_2 в v_3 (длина 7), затем в v_4 (длина 7. В v_1 короче, но в нее нужно возвращаться в самом конце). Итак, мы прошли все вершины и должны теперь вернуться в v_1. И вот теперь наступает расплата за жадность: единственный доступный для возвращения маршрут из условия задачи – это (v_4, v_1) длиной 96!

Алгоритм «самой близкой вставки» решения задачи коммивояжера

1. Сначала выбираем любую вершину (начальную). Далее выбираем инцидентное вершине ребро с самым маленьким весом. Теперь у нас включены две вершины. Среди всех ребер, инцидентных данным вершинам выбираем ребро минимального веса и также присоединяем к нашей цепи. Замыкаем две несоединенные вершины (независимо от веса ребра). Получаем стартовый цикл P, состоящий из трех вершин.

2. Следующий шаг заключается в увеличении цикла в результате включения еще одной выбранной вершины. Среди всех ребер, инцидентных вершинам, уже включенных в P выбираем ребро минимального веса и включаем его в рассмотрение. В результате степень одной из вершин становится равной 3, чего быть не должно, следовательно одно из двух ребер, инцидентных данной вершине и включенных ранее нужно исключить. При этом необходимо будет

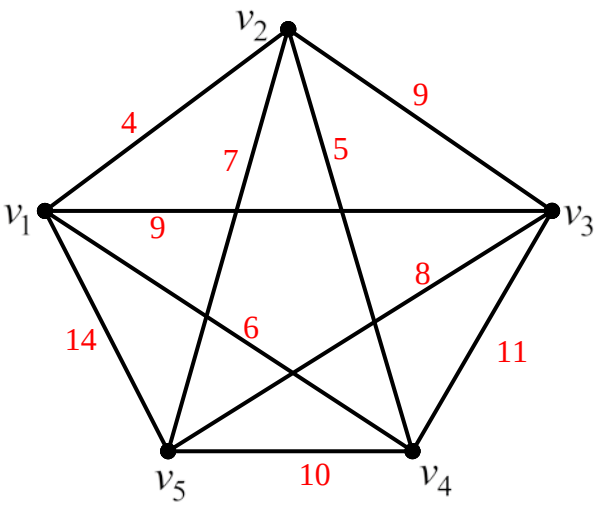
также «замкнуть» цикл. Для того, чтобы определить какое ребро нужно исключить мы проверяем следующие соотношения:

$$I(v_i,u)=W(u,v_j)-W(v_i,v_j),$$

где $W(v_i,v_j)$ - вес исключаемого ребра, а $W(u,v_j)$ - вес ребра, которое нужно добавить, чтобы «замкнуть» цикл.

3. Повторяем шаг 2, пока цикл не будет включать в себя все вершины графа.

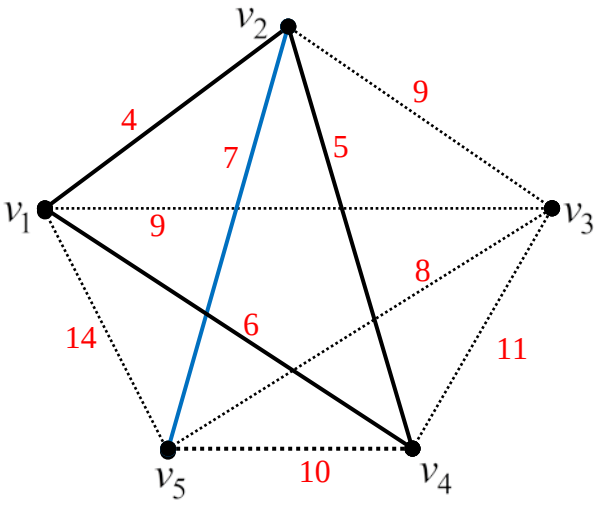
Пример: решить задачу коммивояжера для графа:



Решение:

Шаг 1. Ребро минимального веса (v_1,v_2) весом 4. Добавляем его. Среди ребер, инцидентных вершинам v_1 и v_2 минимальный вес у ребра (v_2,v_3). Добавляем его. Теперь нужно замкнуть цикл. Добавляем ребро (v_1,v_3). Получили начальный цикл.

Шаг 2. Среди ребер, инцидентных вершинам v_1, v_2 и v_3, включенным в цикл, минимальный вес имеет ребро (v_2,v_5). Добавляем его:



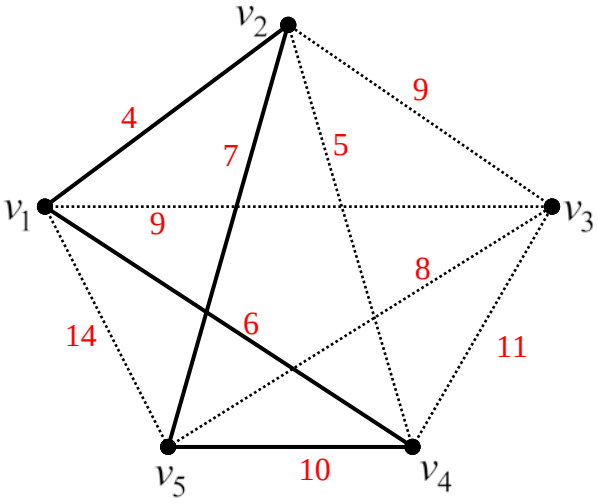
У вершины v_2 степень 3, значит одно из ребер (v_2, v_1) или (v_2, v_4) нужно исключить. Определим какое:

-(v_2, v_1): -4+14=10,

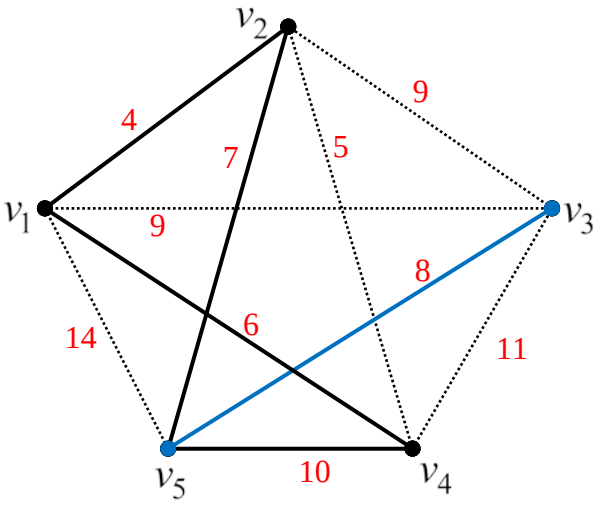
-(v_2, v_4): -5+10=5.

Это означает, что если мы убираем ребро (v_2, v_1), то из нашего цикла мы должны убрать вес этого ребра (-4) и тогда у нас получится незамкнутый цикл (висящие вершины v_1 и v_5), следовательно нужно добавить ребро (v_1, v_5) весом 14. Если же убираем ребро (v_2, v_4), то убираем его вес (-5) и добавляем ребро (v_4, v_5) весом 10.

Более эффективна вторая схема. Получим:



Не включена вершина v_3. Повторим шаг 2. Минимальный вес у ребра (v_3, v_5) - добавляем его.



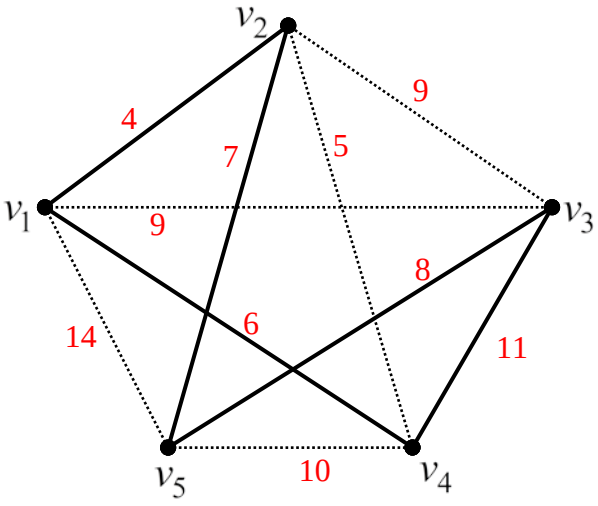
Степень вершины v_5 стала 3, значит нужно исключить либо (v_2, v_5), либо (v_4, v_5). Выполним проверку:

-(v_2, v_5): -7+9=2,

-(v_4, v_5): -10+11=1.

Это означает, что если мы убираем ребро (v_2, v_5), то из нашего цикла мы должны убрать вес этого ребра (-7) и тогда у нас получится незамкнутый цикл (висящие вершины v_2 и v_3), следовательно нужно добавить ребро (v_2, v_3) весом 9. Если же убираем ребро (v_4, v_5), то убираем его вес (-10) и добавляем ребро (v_3, v_4) весом 11.

Вторая схема эффективнее. Выберем ее получим:



Все вершины включены в цикл. Это ответ!

Раскраски графов

Задача четырех красок

Как отмечалось ранее в 1852 году Фрэнсис Гутри, составляя карту графств Англии, обратил внимание, что для такой цели хватает четырех красок. Его брат, Фредерик, передал вопрос своему учителю по математике, Огастесу де Моргану, который упомянул о нём в своем письме Уильяму Гамильтону в 1852 году, а уже в 1878 г. английский математик Артур Кэли на заседании английского королевского научного общества предложил задачу, добавив, что размышлял над ней длительное время и не смог её решить.

В 1880 году Альфред Кемпе опубликовал решение этой проблемы за которое был избран членом Лондонского Королевского общества и позже — президентом Лондонского математического сообщества. Однако, в 1890 году Хивуд нашёл ошибку в доказательстве Кемпе. В свою очередь он доказал теорему пяти красок, показав, что любая плоская карта может быть раскрашена не более, чем пятью цветами. 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.

Суть задачи: можно ли на любой политико-административной карте раскрасить страны так, чтобы никакие две страны, имеющие общую границу, не были раскрашены одинаковой краской, и чтобы были использованы всего четыре краски?

В терминах графов эта задача ставится следующим образом: можно ли каждую вершину неориентированного графа раскрасить с помощью одной из четырёх красок так, чтобы никакие две смежные вершины не были закрашены одним цветом.

Раскрашивать можно как ребра графа, так и вершины, однако, когда говорят о раскраске графов, почти всегда подразумевают под этим раскраску их вершин. Заметим, что раскрашиваемые графы не содержат петель, но могут содержать кратные рёбра. Кратность рёбер не влияет на раскрашиваемость.

Хроматическое число графа

Раскраска графа — функция, которая каждой вершине ставит в соответствие некоторое натуральное число (цвет). Раскраска называется правильной, если смежные вершины имеют разные цвета. Граф без петель называют ***k–раскрашиваемым***, если есть его правильная раскраска из k цветов.

Раскраска графа, в которой используется минимальное число красок, называется ***минимальной***. ***Хроматическое число*** $\chi(G)$ графа G равно минимально возможному числу цветов в раскраске графа.

К задаче определения минимальной раскраски графа сводятся многие задачи. В качестве примера назовем задачу нахождения минимального числа внутренних переменных в программе.


Теорема

Если наибольшая из степеней вершин графа равна k, то этот граф (k+1) – раскрашиваем.

Доказательство:

➢ Для доказательства справедливости этой формулы в общем случае воспользуемся методом математической индукции по числу вершин n в графе.

Первый шаг индукции для $k=1$ и $n=2$, т.е. имеем граф из двух вершин, которые соединены, тогда очевидно, что $\chi(G)=2=k+1$.

Предположим, что теорема верна для графа с $(n-1)$-ой вершинами. Берём граф с n вершинами и удаляем из него произвольную вершину вместе с рёбрами, у которых она является концом. В оставшемся графе будет тогда $(n-1)$ вершин, причём степени вершин по–прежнему не превосходят k.

По индуктивному предположению этот граф $k+1$ раскрашиваем. Отсюда получается $(k+1)$ раскраска для графа с n вершинами, если окрасить удалённую вершину цветом, отличным от цветов смежных с ней вершин (а их не более k). Теорема доказана. ◅

Следствие: Если граф G имеет хотя бы одно ребро, то $\chi(G)\geq 2$.


Теорема

Хроматическое число полного графа равно числу его вершин.

Доказательство очевидно, т.к. каждая вершина полного графа

соединена с каждой из других вершин.

Теорема

Хроматическое число графа равно 2 тогда и только тогда, когда он является двудольным.

Доказательство:

➢ Пусть, граф $G = \langle V, E \rangle$ – двудольный и $V = V_1 \cup V_2$, тогда окрасим вершины множества V_1 в цвет 1 (это можно сделать, поскольку они несмежны), а все вершины множества V_2 в цвет 2. Получили, что $\chi(G) = 2$.

Пусть теперь $\chi(G) = 2$, тогда, все вершины, окрашенные в цвет 1 образуют множество V_1, а окрашенные в цвет 2 – множество V_2. По определению раскраски, вершины в рамках одного множества несмежны, тогда, по определению, граф $G = \langle V, E \rangle$-двудольный. Теорема доказана. ◅

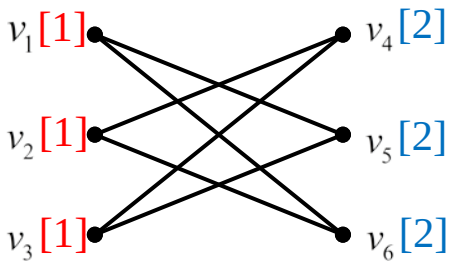
Алгоритмы раскраски графа

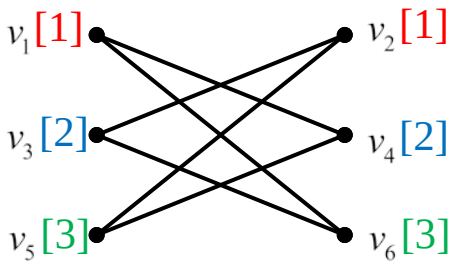
Рассмотрим теперь некоторые алгоритмы раскраски:

Жадный алгоритм раскрашивания:

1. упорядочиваем вершины $v_1, v_2, ..., v_n$,
2. последовательно присваиваем вершине v_i наименьший доступный цвет, не использовавшийся для окраски соседей v_i среди $v_1, v_2, ..., v_{i-1}$, либо добавляет новый.

Качество полученной раскраски сильно зависит от выбранного порядка вершин. Всегда существует такой порядок, который приводит жадный алгоритм к оптимальному числу $\chi(G)$ 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 $\frac{n}{2}$ цветов (в двух графах ниже цифрами в квадратных скобках показана раскраска):





Еще один жадный алгоритм раскрашивания:

1. Вычислить степени всех вершин, положить $k=1$.
2. Расположить вершины по убыванию их степеней и первую из неокрашенных вершин окрасить в цвет k.
3. Просмотреть вершины в порядке убывания степеней и окрасить в цвет k все вершины, которые несмежны с вершинами, уже окрашенными в цвет k.
4. Если все вершины уже окрашены, то $\chi(G)=k$. Если нет, то берем следующее значение и переходим к шагу 2.

Фактически здесь идет упорядочивание первого алгоритма по степени вершины.

Алгоритм последовательного раскрашивания заключается в том, что:

1. для каждой вершины v графа G определяем множество допустимых цветов $S(v)$. Для первого шага $S(v)=|V|$ для всех вершин.
2. Красим вершины последовательно, выбирая из допустимых цветов минимальный и удаляя его из множества допустимых цветов вершин, смежных с окрашенной.
3. Повторяем этот процесс, пока не будут покрашены все вершины

В общем случае, алгоритм последовательного раскрашивания не является оптимальным (это опять вариация жадного алгоритма). Однако, среди $n!$ различных способов упорядочивания вершин графа существует как минимум один, приводящий к раскраске в хроматическое число цветов.

В большинстве случаев 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.

Независимое множество вершин графа — множество попарно несмежных вершин графа.

Максимальное независимое множество вершин графа — независимое множество вершин графа, к которому невозможно добавить вершины.

Алгоритм раскраски, основанный на **методе поиска максимального независимого множества вершин**:

1. Выделяем какое-либо максимальное независимое множество вершин, красим их в первый цвет и удаляем из графа.
2. Повторяем процедуру: из оставшихся вершин выбираем максимальное независимое множество, красим во второй цвет и удаляем.
3. Продолжаем процесс до тех пор, пока не покрасим все вершины.

Сложность данного алгоритма – выделение максимального независимого множества вершин. Рассмотрим, как это можно сделать.

Пусть S –множество еще не просмотренных вершин, C – вершины, включенные в независимое множество, $\Gamma(C)$ - множество вершин, смежных с включенными (они не могут быть включены в C). Тогда:

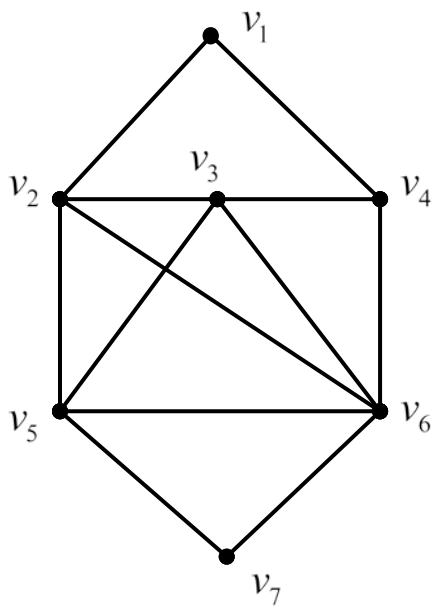
а. Если $S = \varnothing$, то алгоритм прекращает свою работу, если нет, то выбираем произвольную вершину. Строим для нее C и $\Gamma(C)$.

б. Если $S \setminus \left(C \cup \Gamma(C)\right) = \varnothing$, то C – максимальное независимое множество. Если нет, то выбираем произвольную неокрашенную вершину, не смежную с уже входящими в множество C, и добавляем ее в C и строим $\Gamma(C)$.

в. Повторяем предыдущий шаг, пока есть возможность.

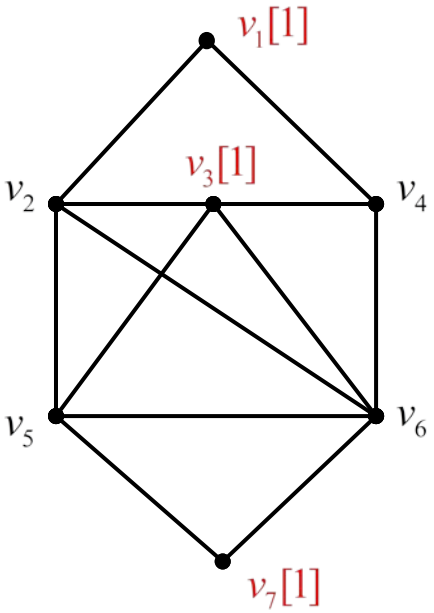
Задачу нахождения максимальных независимых множеств иногда называют «упаковкой вершин». Эта задача NP-полна, так что вряд ли ее можно решить за полиномиальное время. Тем не менее, задача о максимальном множестве может быть решена эффективнее, чем за время, которое даёт полный перебор.

Пример: раскрасить граф используя метод поиска максимального независимого множества вершин:

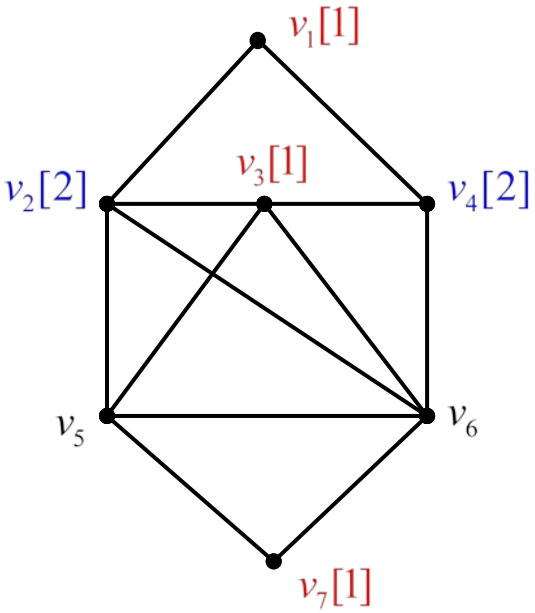


Решение:

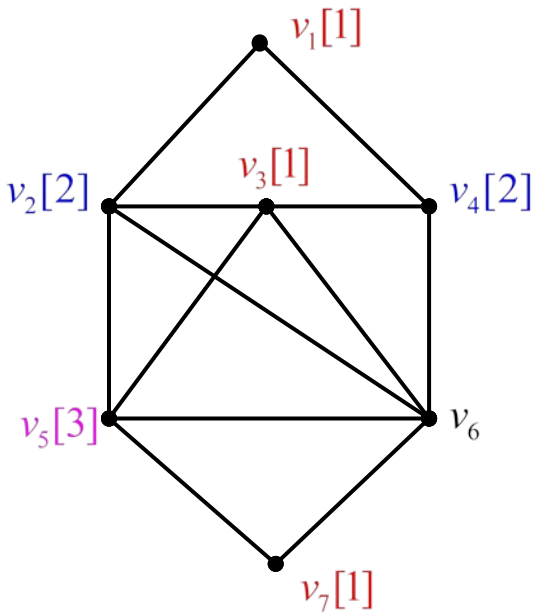
1. Построим независимое множество вершин:
   a. $S=\{v_1,v_2,v_3,v_4,v_5,v_6,v_7\}$. Выберем v_1, тогда $C_1=\{v_1\}$, $\Gamma(C_1)=\{v_2,v_4\}$ - вершины, смежные с вершинами, включенными в C_1.
   b. $S\setminus(C_1\cup\Gamma(C_1))=\{v_3,v_5,v_6,v_7\}$. Выберем v_3 и включим ее в C, тогда $C_2=\{v_1,v_3\}$, $\Gamma(C_2)=\{v_2,v_4,v_5,v_6\}$.
   c. $S\setminus(C_2\cup\Gamma(C_2))=\{v_7\}$. Включим ее в C, тогда $C_3=\{v_1,v_3,v_7\}$, $\Gamma(C_3)=\{v_2,v_4,v_5,v_6\}$.
   d. $S\setminus(C_3\cup\Gamma(C_3))=\varnothing$. Максимально независимое множество вершин найдено: $C_3=\{v_1,v_3,v_7\}$ - красим их в цвет 1:



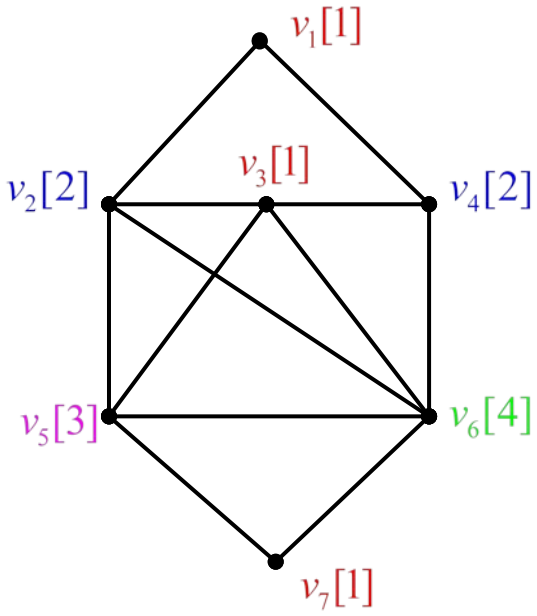
2. Поскольку не все вершины покрашены, то продолжаем раскраску. Построим новое независимое множество вершин:
   a. $S = \{v_2, v_4, v_5, v_6\}$. Выберем v_2, тогда $C_1 = \{v_2\}$, $\Gamma(C_1) = \{v_5, v_6\}$ - вершины, смежные с вершинами, включенными в C_1.
   b. $S \setminus (C_1 \cup \Gamma(C_1)) = \{v_4\}$. Включим ее в C, тогда $C_2 = \{v_2, v_4\}$, $\Gamma(C_2) = \{v_5, v_6\}$.
   c. $S \setminus (C_2 \cup \Gamma(C_2)) = \varnothing$. Максимально независимое множество вершин найдено: $C_2 = \{v_2, v_4\}$ - красим их в цвет 2:



3. Поскольку не все вершины покрашены, то продолжаем раскраску. Опять построим новое независимое множество вершин:
   a. $S = \{v_5, v_6\}$. Выберем v_5, тогда $C_1 = \{v_5\}$, $\Gamma(C_1) = \{v_6\}$.
   b. $S \setminus (C_1 \cup \Gamma(C_1)) = \varnothing$. Максимально независимое множество вершин найдено: $C_1 = \{v_5\}$ - красим их в цвет 3:



4. Очевидно, что последнюю вершину окрасим в цвет 4:



Помимо вершин можно рассматривать также независимое множество ребер графа.

Независимое множество ребер графа — множество попарно несмежных ребер графа, т.е. между двумя ребрами не должно быть общей вершины.

Максимальное независимое множество ребер графа — независимое множество ребер графа, к которому невозможно добавить никаких ребер.

Вопросы для самоконтроля:

1. Понятие двудольного графа.
2. Паросочетания.
3. Венгерский алгоритм построения совершенного паросочетания (задача об оптимальном назначении).
4. Задача о свадьбах.
5. Путь минимальной суммарной длины во взвешенном графе. Алгоритм Дейкстры.
6. Задача коммивояжера.
7. Классификация алгоритмов по их вычислительной сложности
8. Раскраска графа. Хроматическое число графа. Алгоритмы раскраски и их особенность (зависимость от порядка вершин).