Нижегородский государственный университет им. Н.И. Лобачевского

Национальный исследовательский университет

Учебно-научный и инновационный комплекс
«Модели, методы и программные средства»

Основная образовательная программа
010300 «Фундаментальная информатика и информационные технологии»,
общий профиль, квалификация (степень) бакалавр
Учебно-методический комплекс по дисциплине
«Теория конечных графов и ее приложения»

Алексеев В.Е, Захарова Д.В.

ТЕОРИЯ ГРАФОВ

Электронное учебно-методическое пособие

Мероприятие 1.2. Совершенствование образовательных технологий, укрепление материально-технической базы учебного процесса

Нижний Новгород
2012

ТЕОРИЯ ГРАФОВ. Алексеев В.Е., Захарова Д.В. Электронное учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2012. – 57 с.

В учебно-методическом пособии излагаются основные понятия и фундаментальные факты теории графов, методы метрического и структурного анализа графов, алгоритмы решения экстремальных задач на графах. Рассматриваются важнейшие классы графов: деревья, двудольные графы, планарные графы. Пособие содержит также задачи для практических занятий и задания для самостоятельной работы студентов.

Электронное учебно-методическое пособие предназначено для студентов ННГУ, обучающихся по направлению подготовки 010300 «Фундаментальная информатика и информационные технологии», изучающих курс «Теория конечных графов и ее приложения».

Оглавление

Предисловие

Настоящее пособие предназначено для литературно-методической поддержки курса «Теория конечных графов и ее приложения», читаемого на факультете ВМК ННГУ, оно может использоваться также в курсе дискретной математики и в спецкурсах, где затрагиваются вопросы теории графов и разработки алгоритмов на графах.

Пособие является комбинацией конспекта лекций и сборника задач. В нем излагаются все необходимые теоретические сведения – определения и математические факты. При этом доказательства теорем, как правило, не приводятся. Отсутствующие доказательства могут быть найдены либо в [1] или [2], либо в источнике, указанном в заголовке теоремы, либо достаточно просты и студенты могут их восстановить самостоятельно.

Значительное внимание уделяется алгоритмическому направлению в теории графов, методам исследования графов и решения оптимизационных задач на графах. Некоторые алгоритмы описываются с помощью псевдокода, подобного тем, которые используются в различных руководствах по алгоритмам, например, [8] или [9].

Основная масса задач предназначена для аудиторной работы и домашних заданий. Задачи, помеченные звездочкой, не являются обязательными, они могут использоваться для самостоятельной работы студентов, склонных к аналитической или алгоритмической деятельности.

1. Начальные понятия

Определение графа

Граф состоит из двух множеств – множества V, элементы которого называются *вершинами*, и множества E, состоящего из пар вершин, эти пары называются *ребрами* графа. Это записывают так: $G = (V, E)$, где G – имя графа.

Один момент в этом определении требует уточнения: считаем ли мы ребра (a,b) и (b,a) различными. Если да (и это распространяется на все ребра), то граф называется *ориентированным* (сокращенно *орграф*), в противном случае – *неориентированным*.

Говорят, что ребро (a,b) *соединяет* вершины a и b. Если такое ребро в графе есть, то говорят, что вершины a и b в нем *смежны*. Заметим, что в графе может быть не более одного ребра, соединяющего данную пару вершин. Ребро типа (a,a), т.е. соединяющее вершину с ней же самой, называют *петлей*.

Говорят, что ребро $e = (a,b)$ *инцидентно* каждой из вершин a и b, а каждая из этих вершин инцидентна ребру e.

В дальнейшем, если не оговаривается иное, под графом понимается неориентированный граф без петель, такие графы называют *обыкновенными*. Для обозначения числа вершин и числа ребер графа будем обычно использовать буквы n и m.

Мультиграф – обобщение понятия графа. В мультиграфе могут быть *кратные ребра*, то есть несколько ребер, соединяющих одну и ту же пару вершин. Иначе говоря, в мультиграфе E является мультимножеством, то есть одна пара может входить в него несколько раз.

Способы задания графов

Существует много способов представить граф, назовем только самые распространенные.

1. Перечисление элементов. Исходя из определения, для того, чтобы задать граф, достаточно перечислить его вершины и ребра (т.е. пары вершин).
2. Изображение. Если граф невелик, его можно нарисовать. В неориентированном графе ребра изображаются линиями, в ориентированном – стрелками.
3. Матрица смежности. Пусть G – граф с n вершинами, пронумерованными числами от 1 до n. Матрица смежности – это таблица с n строками и n столбцами, в которой элемент, стоящий на пересечении строки с номером i и столбца с номером j, равен 1, если вершины с номерами i и j смежны, и 0, если они не смежны.
4. Матрица инцидентности. Пусть G – граф, вершины которого пронумерованы числами от 1 до n, а ребра – числами от 1 до m. В матрице инцидентности строки соответствуют вершинам, а столбцы – ребрам. На пересечении строки с номером i и столбца с номером j стоит 1, если вершина с номерами i инцидентна ребру с номером j смежны, и 0 в противном случае.
5. Списки смежности. Этот способ часто используется для компьютерного представления графов. Состоит он в том, что для каждой вершины задается список всех смежных с ней вершин. В структурах данных, применяемых в программировании, списки смежности могут быть реализованы как массив линейных списков. При решении задач

будем эти списки оформлять так: пишется номер или имя вершины и после двоеточия перечисляются все смежные с ней вершины.

Окрестности и степени

Множество вершин, смежных с данной вершиной x в некотором графе, называется *окрестностью* этой вершины и обозначается через $N(x)$. Число вершин в $N(x)$ называется *степенью* вершины x и обозначается через $\deg(x)$.

Если сложить степени всех вершин графа, то каждое ребро внесет в эту сумму вклад, равный 2. Следовательно, сумма степеней всех вершин равна удвоенному числу ребер графа:

$$\sum_{x \in V} \deg(x) = 2m .$$

Это утверждение называют **теоремой о рукопожатиях**.

В ориентированном графе для каждой вершины x можно ввести два числа: $\deg^+(x)$ – число заходящих ребер и $\deg^-(x)$ – число выходящих. Их называют соответственно полустепенью захода и полустепенью исхода. Аналогом теоремы о рукопожатиях для ориентированного графа является очевидное равенство

$$\sum_{x \in V} \deg^+(x) = \sum_{x \in V} \deg^-(x) .$$

Некоторые специальные графы

Пустой граф – граф, не содержащий ни одного ребра. Пустой граф с множеством вершин $(1,2,\ldots,n\}$ обозначается O_n.

Полный граф – граф, в котором каждые две вершины смежны. Полный граф с множеством вершин $(1,2,\ldots,n\}$ обозначается K_n.

Путь P_n имеет множество вершин $(1,2,\ldots,n\}$, ребрами его являются пары $(i,i+1)$, $i = 1,2,\ldots,n-1$.

Цикл C_n получается из графа P_n добавлением ребра $(1,n)$.

Полный двудольный граф $K_{p,q}$. Множество вершин состоит из двух частей, в одной из них p вершин, в другой q. Любые две вершины из одной части не смежны, любые две вершины из разных частей смежны.

Изоморфизм

Графы $G_1 = (V_1, E_1)$ и $G_2 = (V_2, E_2)$ называются *изоморфными*, если существует такая биекция f множества V_1 на множество V_2, что $(a,b) \in E_1$ тогда и только тогда, когда $(f(a), f(b)) \in E_2$. Отображение f в этом случае называется *изоморфизмом* графа G_1 на граф G_2.

Тот факт, что графы G_1 и G_2 изоморфны, записывается так: $G_1 \cong G_2$.

Это определение изоморфизма годится и для ориентированных графов, нужно только обе упоминаемые в нем пары вершин считать упорядоченными.

Изоморфизм – бинарное отношение на множестве графов. Очевидно, это отношение рефлексивно, симметрично и транзитивно, то есть является отношением эквивалентности. Классы эквивалентности называются *абстрактными графами*. Когда говорят, что рассматриваются абстрактные графы, это означает, что изоморфные графы считаются одинаковыми. Абстрактный граф можно представлять себе как граф, у изображения

которого стерты имена (пометки) вершин, поэтому абстрактные графы иногда называют также *непомеченными* графами.

Характеристики графов, которые сохраняются при изоморфизме, называются *инвариантами*. Примеры простых инвариантов – число ребер, наличие вершины данной степени, число вершин данной степени, набор степеней (последовательность степеней, упорядоченная по неубыванию), наличие циклов данной длины, число циклов данной длины. Если удается установить, что для двух исследуемых графов некоторый инвариант принимает разные значения, то эти графы неизоморфны. Поиски полной системы легко вычисляемых инвариантов, то есть такой, что совпадение всех этих инвариантов у двух графов гарантировало бы, что эти графы изоморфны, до сих пор не увенчались успехом. Для доказательства того, что два графа изоморфны, необходимо предъявить соответствующую биекцию.

Подграфы

Граф $G' = (V', E')$ называется *подграфом* графа $G = (V, E)$, если $V' \subseteq V$, $E' \subseteq E$. Всякий подграф может быть получен из графа удалением некоторых вершин и ребер (при удалении вершины удаляются и все инцидентные ей ребра).

Подграф G' графа G называется *остовным*, если $V' = V$. Остовный подграф получается удалением из графа некоторых ребер, вершины же остаются в неприкосновенности.

Порожденный подграф получается из графа удалением некоторых вершин. Все ребра, которые были в графе между оставшимися вершинами, должны сохраниться в подграфе. Говорят, что подграф порождается оставшимися вершинами.

Операции над графами

Для графов $G_1 = (V_1, E_1)$ и $G_2 = (V_2, E_2)$ их *объединение* $G_1 \cup G_2$ определяется как граф $(V_1 \cup V_2, E_1 \cup E_2)$, а *пересечение* $G_1 \cap G_2$ – как граф $(V_1 \cap V_2, E_1 \cap E_2)$.

Дополнением (*дополнительным графом*) к графу $G = (V, E)$ называется граф $\overline{G}$, у которого множество вершин то же, что у G, а множество ребер является дополнением множества E до множества всех неупорядоченных пар вершин. Иначе говоря, две различные вершины смежны в графе $\overline{G}$ тогда и только тогда, когда они не смежны в графе G. Например, $\overline{O_n} = K_n$.

Под *суммой* $G_1 + G_2$ двух абстрактных графов понимают объединение графов с непересекающимися множествами вершин. Точнее говоря, имеется в виду следующее. Сначала вершинам графов-слагаемых присваиваются имена (пометки, номера) так, чтобы множества вершин не пересекались, затем полученные графы объединяются и пометки стираются (т.е. результат операции – тоже абстрактный граф). Операция сложения ассоциативна, то есть $(G_1 + G_2) + G_3 = G_1 + (G_2 + G_3)$ для любых трех графов. Поэтому можно образовывать сумму любого числа графов, не указывая порядка действий с помощью скобок. Если складываются k экземпляров одного и того же графа G, то полученный граф обозначается через kG. Например, $O_n \cong nK_1$.

Соединением графов G_1 и G_2 называется граф $G_1 \circ G_2$, получаемый из их суммы добавлением всех ребер, соединяющих вершины первого слагаемого с вершинами второго. Например, $K_{p,q} = O_p \circ O_q$. На рисунке 1 слева изображен граф $C_4 + 2K_2 + 3K_1$ справа – граф $P_3 \circ (P_2 + P_1)$.

Рис. 1.

Декартово произведение (далее просто произведение) $G_1 \times G_2$ графов $G_1 = (V_1, E_1)$ и $G_2 = (V_2, E_2)$ определяется следующим образом. Множеством вершин графа $G_1 \times G_2$ является декартово произведение множеств V_1 и V_2, то есть вершины этого графа – упорядоченные пары (x, y), где $x \in V_1$, $y \in V_2$. Вершины (x_1, y_1) и (x_2, y_2) в графе $G_1 \times G_2$ смежны тогда и только тогда, когда $x_1 = x_2$ и y_1 смежна с y_2 в графе G_2, или $y_1 = y_2$ и x_1 смежна с x_2 в графе G_1. С помощью операции произведения можно выразить некоторые важные графы через простейшие. Например, произведение двух путей дает *прямоугольную решетку* – см. рисунок 2.

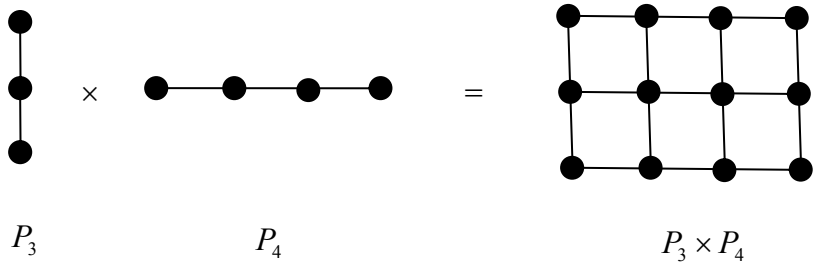


Рис. 2.

Другой пример – *k*-мерный куб Q_k, определяемый следующим образом. Вершинами его являются всевозможные упорядоченные двоичные наборы длины *k*. Всего, таким образом, в этом графе 2^k вершин. Вершины $x = (x_1, \ldots, x_k)$ и $y = (y_1, \ldots, y_k)$ смежны в нем тогда и только тогда, когда наборы *x* и *y* различаются точно в одной координате. С помощью операции произведения граф Q_k можно определить рекурсивно:

$$Q_1 = K_2, \quad Q_k = Q_{k-1} \times K_2.$$

На рисунке 3 показано, как получается Q_3 из Q_2.

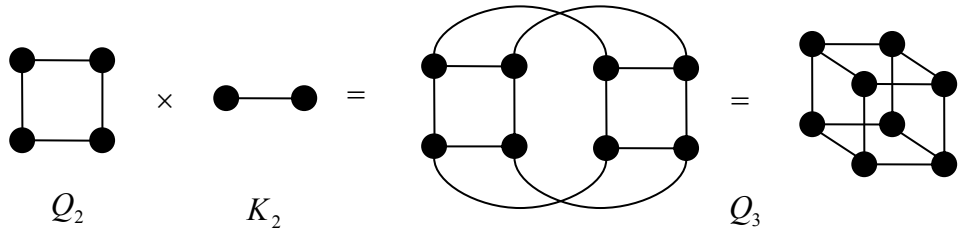


Рис. 3.

Пути, циклы, связность

Путь в графе – это последовательность вершин $x_1, x_2, \ldots, x_k$, в которой каждая пара (x_i, x_{i+1}), $i = 1,2,\ldots,k-1$, является ребром, причем все эти ребра различны. Путь *соединяет* вершины x_1 и x_k. *Длиной* пути называется число ребер в нем, т.е. $k-1$.

Цикл – путь, у которого $x_1 = x_k$.

Граф называется *связным*, если для любых двух его вершин имеется путь, соединяющий эти вершины. Если граф несвязен, то он состоит из нескольких связных подграфов, между которыми нет ребер, они называются *компонентами связности*.

Шарнир (*точка сочленения*) в графе – вершина, при удалении которой увеличивается число компонент связности.

Перешеек – ребро, при удалении которого увеличивается число компонент связности. Легко доказать следующее утверждение.

Теорема о перешейках. *Ребро является перешейком тогда и только тогда, когда через него не проходит ни один цикл.*

В ориентированном графе можно определить два типа путей.

Ориентированный путь – это, как и в неориентированном случае, последовательность вершин $x_1, x_2, \ldots, x_k$, в которой каждая (упорядоченная) пара (x_i, x_{i+1}), $i = 1,2,\ldots,k-1$, является ребром (и все эти ребра различны).

Неориентированный путь – последовательность вида $x_1, e_1, x_2, e_2, \ldots, e_{k-1}, x_k$, где $x_1, x_2, \ldots, x_k$ – вершины, а $e_1, e_2, \ldots, e_{k-1}$ – ребра графа, причем $e_i = (x_i, x_{i+1})$ или $e_i = (x_{i+1}, x_i)$ для каждого $i = 1,2,\ldots,k-1$.

Орграф называется *связным*, если между любыми двумя вершинами имеется соединяющий их неориентированный путь. Он называется *сильно связным*, если из любой вершины в любую другую имеется ориентированный путь.

Расстояния и метрические характеристики

Расстоянием между вершинами в графе называется наименьшая длина соединяющего их пути. Расстояние между вершинами x и y обозначается через $d(x, y)$.

Эксцентриситет вершины – расстояние от нее до самой удаленной вершины:

$$\mathrm{ecc}(x) = \max_{y \in V} d(x, y).$$

Диаметр графа – максимальное расстояние между вершинами, то есть наибольший эксцентриситет:

$$\mathrm{diam}(G) = \max_{x \in V, y \in V} d(x, y) = \max_{x \in V} \mathrm{ecc}(x).$$

Радиус графа – наименьший эксцентриситет:

$$\mathrm{rad}(G) = \min_{x \in V} \mathrm{ecc}(x).$$

Центральная вершина – вершина, эксцентриситет которой равен радиусу графа.

Центр – множество всех центральных вершин.

Диаметр и радиус графа связаны соотношениями:

$$\mathrm{rad}(G) \le \mathrm{diam}(G) \le 2\,\mathrm{rad}(G).$$

Графы пересечений

Пусть дано семейство множеств $F = \{S_1, S_2, \ldots, S_n\}$. Графом пересечений этого семейства называется граф с множеством вершин $(1,2,\ldots,n\}$, в котором вершины i и j смежны тогда и только тогда, когда $S_i \cap S_j \neq \varnothing$. Этот граф обозначается $\Gamma(F)$. Граф $\Gamma(F)$ содержит в компактном виде информацию о том, какие множества из семейства F имеют непустое пересечение. С другой стороны, семейство F можно рассматривать как еще один способ представления графа $\Gamma(F)$. Следующая теорема показывает, что этот способ универсален, то есть любой граф можно представить как граф пересечений некоторого семейства множеств.

Теорема о графах пересечений [12]. *Для любого графа G существует такое семейство множеств F, что $G \cong \Gamma(F)$.*

Задачи

1.1. Граф задан множеством вершин $V = \{a,b,c,d,e,f\}$ и множеством ребер $E = \{(a,c),(a,f),(b,c),(c,d),(d,f)\}$. Нарисуйте этот граф, постройте для него матрицы смежности и инцидентности, списки смежности.

1.2. Постройте матрицу инцидентности для графа, заданного списками смежности:
$a:b,d;\quad b:a,c,d,f;\quad c:b,f;\quad d:a,b,f;\quad e:;\quad f:b,c,d.$

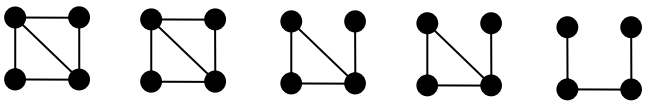
1.3. В графе 30 вершин и 80 ребер, каждая вершина имеет степень 5 или 6. Сколько в нем вершин степени 5?

1.4. В графе каждая вершина имеет степень 3, а число ребер заключено между 16 и 20. Сколько вершин в этом графе?

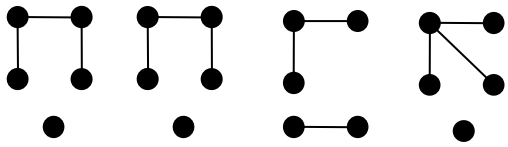
1.5. Найдите все абстрактные графы с 4 вершинами.

1.6. Найдите все абстрактные графы с набором степеней а) (2,2,2,3,3,4);
б) (2,2,2,3,3,3).

1.7. Восстановите граф по его
а) порожденным подграфам, полученным удалением одной вершины:



б) остовным подграфам, полученным удалением одного ребра:



1.8. Граф G имеет множество вершин $\{1,2,\ldots,n\}$. Число ребер в подграфе, полученном удалением вершины i, равно m_i, $i = 1,2,\ldots,n$. Сколько ребер в графе G?

1.9. Граф имеет *n* вершин и *m* ребер. Сколько у него различных а) остовных; б) порожденных подграфов?

1.10. 1) Представьте граф C_6 как объединение трех графов с множествами вершин $\{1,2,3,4\}$, $\{1,2,5,6\}$, $\{3,4,5,6\}$.
2) Верно ли, что любой граф с 6 вершинами можно представить как объединение трех графов с такими множествами вершин?
3) Верно ли, что любой граф с 6 вершинами можно представить как объединение трех графов с множествами вершин $\{1,2,3\}$, $\{3,4,5\}$, $\{5,6,1\}$?

1.11. Верно ли, что для любых графов G_1 и G_2 выполняется равенство $\overline{G_1 \cup G_2} = \overline{G_1} \cap \overline{G_2}$?

1.12. Найдите граф G с минимальным числом вершин $n > 1$ такой, что оба графа G и $\overline{G}$ связны.

1.13. Найдите граф G с минимальным числом вершин $n > 1$ такой, что оба графа G и $\overline{G}$ несвязны.

1.14. Изоморфны ли графы 1) $P_2 \times C_3$ и $K_2 \circ \overline{2K_2}$; 2) $\overline{K_{1,2}} \times C_3$ и $K_3 + \overline{C_6}$; 3) $\overline{K_4 \times P_2}$ и Q_3?

1.15. Найдите граф с минимальным числом вершин $n > 1$, который не является суммой или соединением меньших графов.

1.16. Граф задан матрицей смежности:

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

Постройте для него матрицу расстояний между вершинами, найдите эксцентриситеты вершин, диаметр, радиус, центр. Изоморфны ли этот граф и дополнительный к нему?

1.17. Каково расстояние между вершинами (0,0,..., 0) и (1,1,...,1) в графе Q_k? Сколько имеется кратчайших путей между этими вершинами?

1.18. Какой наибольший диаметр может быть у графа с n вершинами? Сколько имеется (абстрактных) графов с таким диаметром?

1.19. Найдите все (с точностью до изоморфизма) графы с 5 вершинами диаметра 3.

1.20. Может ли радиус графа в результате добавления одного нового ребра а) увеличиться; б) уменьшиться; в) остаться прежним?

1.21. Найдите все (с точностью до изоморфизма) графы с 5 вершинами радиуса 1.

1.22. Найдите все (с точностью до изоморфизма) графы с 4 вершинами, имеющие точно одну центральную вершину.

1.23. Найдите диаметр и радиус графа $C_3 \times C_5$.

1.24. Постройте граф пересечений а) граней трехмерного куба; б) ребер графа K_4.

1.25. Сколько ребер в графе пересечений ребер графа K_n?

1.26. Граф пересечений семейства интервалов на прямой называют графом интервалов. Какие из следующих графов являются графами интервалов: C_3, P_4, C_4, K_4, C_5, $K_{2,3}$, $K_{1,5}$?

1.27. Граф пересечений семейства дуг окружности называют графом дуг. Какие из графов предыдущей задачи являются графами дуг?

1.28*. Докажите, что самодополнительный граф (граф, изоморфный своему дополнению) с n вершинами существует тогда и только тогда, когда $n \equiv 0 \pmod 4$ или $n \equiv 1 \pmod 4$.

1.29*. Докажите, что если в графе нет порожденного подграфа, изоморфного P_3, то каждая его компонента связности является полным подграфом.

1.30*. Докажите, что если в графе нет порожденного подграфа, изоморфного P_4, то один из графов G, $\overline{G}$ несвязен.

2. Перечисление графов

Помеченные графы

Число помеченных обыкновенных графов с множеством вершин $\{1,2,\ldots,n\}$ равно

$$g_n = 2^{\binom{n}{2}} = 2^{\frac{n(n-1)}{2}}.$$

Говорят, что *почти все* графы обладают некоторым свойством, если отношение числа графов с n вершинами, имеющих это свойство, к числу всех графов с n вершинами (т.е. к g_n) стремится к 1 при $n \to \infty$.

Теорема о диаметре почти всех графов [4]. *Почти все графы имеют диаметр 2.*

Следствие 1. *Почти все графы имеют радиус 2.*

Следствие 2. *Почти все графы связны.*

Непомеченные графы

Подстановки. *Подстановкой* называется биекция множества на себя. Подстановка π на множестве $\{1,2,\ldots,n\}$ записывается в виде

$$\begin{pmatrix} 1 & 2 & \ldots & n \\ \pi(1) & \pi(2) & \ldots & \pi(n) \end{pmatrix},$$

при этом $(\pi(1),\pi(2),\ldots,\pi(n))$ – перестановка элементов $1,2,\ldots,n$. Подстановка на конечном множестве V может быть представлена ориентированным графов с множеством вершин V и ребрами $(x,\pi(x))$ для всех $x \in V$. В этом графе у каждой вершины x полустепени исхода и захода равны 1. Поэтому граф подстановки состоит из непересекающихся ориентированных циклов (в нем могут быть петли – это циклы длины 1). На рисунке 4 показан граф подстановки

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 4 & 8 & 3 & 7 & 5 & 9 & 1 & 2 & 6 \end{pmatrix}.$$

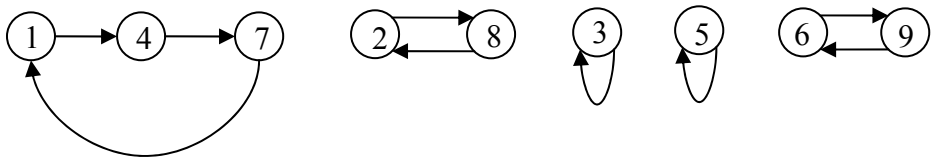


Рис. 4.

Набор чисел $(k_1,k_2,\ldots,k_n)$, где k_1 – количество циклов длины 1, k_2 – циклов длины 2, ..., k_n – циклов длины n, называется *цикловой структурой* подстановки. Эти числа должны удовлетворять равенству $k_1 + 2k_2 + \ldots + nk_n = n$. Число подстановок на множестве из n элементов с цикловой структурой $(k_1,k_2,\ldots,k_n)$ равно

$$w(k_1,k_2,\ldots,k_n) = \frac{n!}{1^{k_1}2^{k_2}\ldots n^{k_n}k_1!k_2!\ldots k_n!}.$$

Автоморфизмы. *Автоморфизм* – подстановка на множестве вершин графа, сохраняющая отношение смежности (две вершины смежны тогда и только тогда, когда смежны их образы). Иначе говоря, автоморфизм – это изоморфизм графа на себя. Множество автоморфизмов данного графа G обозначается $\mathrm{Aut}\ G$.

Каким числом способов можно абстрактный граф с n вершинами превратить в помеченный граф с множеством вершин $\{1,2,\ldots,n\}$? Иначе: сколько различных графов можно получить, переставляя пометки вершин у данного помеченного графа G? Это зависит от графа: из P_4 можно получить 12 разных графов, из C_4 – 8, а для K_4 любая перестановка дает тот же самый граф. В общем случае число различных помеченных графов, которые можно получить из графа G, дает формула

$$\frac{n!}{|\ \mathrm{Aut}\ G\ |}.$$

Орбиты пар. Пусть π – подстановка на множестве $\{1,2,\ldots,n\}$. Относительно этой подстановки множество всех неупорядоченных пар различных элементов из $\{1,2,\ldots,n\}$ разбивается на *орбиты пар* – одну пару можно превратить в другую, действуя на нее

(может быть, многократно) подстановкой π тогда и только тогда, когда обе пары принадлежат одной орбите. Если π является автоморфизмом графа G, то все пары вершин из одной орбиты одновременно смежны или несмежны в этом графе. Обозначим число орбит пар относительно подстановки π через $c(\pi)$. Любой граф, для которого эта подстановка является автоморфизмом, можно получить следующим образом: каждой орбите приписать одно из значений 0 или 1. Все пары вершин из орбиты объявить смежными, если этой орбите приписана 1, и несмежными, если ей приписан 0. Следовательно, число графов, для которых π является автоморфизмом, равно $2^{c(\pi)}$.

Число $c(\pi)$ зависит только от цикловой структуры подстановки. Оно равно

$$c(\pi) = c(k_1, k_2, \ldots, k_n) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} k_i k_j d(i,j) - \frac{1}{2}\sum_{i=1}^{n} k_i \sigma(i),$$

где $d(x,y)$ – наибольший общий делитель чисел x и y, $\sigma(i)$ – остаток от деления числа i на 2.

Число графов. Число абстрактных графов с n вершинами обозначим через $\widetilde{g}_n$. Построим граф Γ, имеющий вершины двух типов. Вершины первого типа соответствуют графам с множеством вершин $\{1,2,\ldots,n\}$, вершины второго – подстановкам на этом множестве. Вершину-граф соединим ребром с вершиной-подстановкой, если эта подстановка является автоморфизмом этого графа. Подсчитаем двумя способами число ребер в графе Γ. С одной стороны, подстановка π является автоморфизмом $2^{c(\pi)}$ графов. Поэтому число ребер в графе Γ равно

$$\sum_{\pi} 2^{c(\pi)},$$

суммирование ведется по всем подстановкам на множестве $\{1,2,\ldots,n\}$. С другой стороны, из вершины, соответствующей графу G, выходит $|\mathrm{Aut}(G)|$ ребер. Объединим все графы, изоморфные графу G, в один класс, представляющий один абстрактный граф. Из каждой вершины-графа этого класса выходит $|\mathrm{Aut}(G)|$ ребер, а всего в классе имеется $\dfrac{n!}{|\mathrm{Aut}\ G|}$ графов. Следовательно, всего из вершин этого класса выходит $n!$ ребер. Так как имеется $\widetilde{g}_n$ классов, то число ребер в графе Γ равно $n!\widetilde{g}_n$. Получаем равенство

$$\widetilde{g}_n = \frac{1}{n!}\sum_{\pi} 2^{c(\pi)}.$$

Учитывая формулу для $c(\pi)$, получаем формулу для числа абстрактных графов:

$$\widetilde{g}_n = \frac{1}{n!}\sum_{\substack{k_1,k_2,\ldots,k_n \geq 0 \\ k_1+2k_2+\ldots+nk_n=n}} w(k_1,k_2,\ldots,k_n)\, 2^{c(k_1,k_2,\ldots,k_n)}.$$

Задачи

2.1. Сколько имеется неориентированных графов с n вершинами, в которых допускаются петли?

2.2. Найдите число ориентированных графов с n вершинами, в которых
а) возможны петли;
б) петель нет;
в) петель нет и каждая пара различных вершин соединена не более чем одним ребром;

г) возможны петли и каждая пара вершин соединена не более чем одним ребром;
д) петель нет и каждая пара различных вершин соединена точно одним ребром.

2.3. Найдите число графов с n вершинами, в которых возможны и ориентированные и неориентированные ребра (но не петли), причем две вершины могут быть соединены не более чем одним ребром.

2.4. Найдите число неориентированных мультиграфов без петель, в которых для каждой пары вершин имеется не более четырех соединяющих эти вершины ребер.

2.5. Найдите число графов с n вершинами, в которых а) данные k вершин являются изолированными (имеют степень 0); б) нет изолированных вершин. Верно ли, что почти все графы не имеют изолированных вершин?

2.6. Если к графу с $n-1$ вершиной добавить еще одну вершину и соединить ее ребрами со всеми вершинами нечетной степени, то получится граф с n вершинами, в котором степени всех вершин четны. Сколько имеется графов, у которых степени всех вершин четны? Верно ли, что почти все графы имеют вершины нечетной степени?

2.7. Сколько имеется помеченных графов с n вершинами, у которых степень каждой вершины равна 1?

2.8. Сколько различных помеченных графов можно получить, добавляя одно новое ребро к графу C_n?

2.9. Сколько различных абстрактных графов можно получить, добавляя одно ребро к графу C_n?

2.10. Сколько различных помеченных графов можно получить, добавляя одно ребро к графу P_n?

2.11. Сколько различных абстрактных графов можно получить, добавляя одно ребро к графу а) P_7? б) P_8?

2.12. Сколько различных помеченных графов можно получить, перенумеровывая вершины графа а) $K_{1,q}$? б) P_5? в) C_5?

2.13. Сколько различных автоморфизмов имеет граф а) P_n; б) C_n; в) $K_{p,q}$; г) Q_3?

2.14. Найдите по возможности малый граф, не имеющий нетривиальных автоморфизмов.

2.15. Опишите единственный нетривиальный автоморфизм графа P_n с помощью формулы (нумерация вершин начинается с 1).

2.16. Выразите формулами два автоморфизма графа C_n, переводящих вершину i в вершину j (нумерация вершин начинается с 0).

2.17. Пусть $(a_1, a_2, \ldots, a_n)$ – двоичный вектор. На множестве вершин графа Q_n

рассмотрим преобразование, переводящее вершину $(x_1, x_2, \ldots, x_n)$ в вершину $(x_1 \oplus a_1, x_2 \oplus a_2, \ldots, x_n \oplus a_n)$, ($\oplus$ – сумма по модулю 2). Докажите, что оно является автоморфизмом этого графа.

2.18. Сколько имеется орбит пар относительно подстановки на множестве из n элементов, переставляющей два элемента и оставляющей остальные неподвижными?

2.19. Примените формулу для числа абстрактных графов при $n = 4$.

2.20*. Докажите, что почти все графы имеют хотя бы один треугольник.

2.21*. Докажите, что почти все графы имеют хотя бы один полный подграф из 4 вершин.

3. Важнейшие классы графов

Деревья

Деревом называется связный граф, не имеющий циклов. Граф без циклов называют *лесом*.

Во всяком дереве, в котором больше одной вершины, имеется не менее двух вершин степени 1. Такие вершины называют *листьями*.

Теорема о свойствах деревьев *Если G – дерево, то*
1) *число ребер в нем на 1 меньше числа вершин*;
2) *в G любая пара вершин соединена единственным путем*;
3) *при добавлении к G любого нового ребра образуется цикл*;
4) *при удалении из G любого ребра он превращается в несвязный граф*.

Теорема о центре дерева. *Центр любого дерева состоит из одной вершины или из двух смежных вершин.*

Центр дерева можно найти следующим образом. Если в дереве больше двух вершин, удалим все его листья. С полученным деревом поступаем так же, это продолжается, пока не останется дерево из одной или двух вершин. Эти вершины и образуют центр исходного дерева.

Наиболее компактным способом представления помеченных деревьев является *код Прюфера*. Пусть дерево T имеет множество вершин $\{1,2,\ldots,n\}$. Код Прюфера этого дерева представляет собой последовательность $p(T) = (p_1, p_2, \ldots, p_{n-2})$, элементами которой являются вершины. Он строится с помощью следующей процедуры.

Построение кода Прюфера

for $i := 1$ **to** $n - 2$ **do**
найти в дереве T наименьший лист a;
пусть b – вершина, смежная с a;
положить $p_i = b$;

удалить из T вершину a

По коду Прюфера легко определить степени вершин дерева: степень вершины на 1 больше числа вхождений этой вершины в $p(T)$. Зная степени, можно восстановить дерево по коду с помощью следующей процедуры.

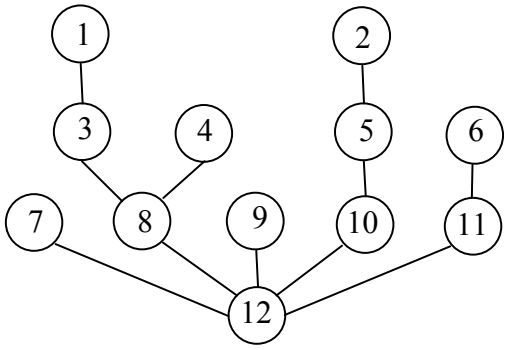
Восстановление дерева по коду Прюфера

Создать граф $T = (V, E)$ с $V = \{1,2,\ldots,n\}$, $E = \varnothing$;
for i:=1 **to** $n-2$ **do**
найти наименьшее a с $\deg(a) = 1$;
добавить к множеству E ребро (a, b_i);
уменьшить $\deg(a)$ и $\deg(b_i)$ на 1;.
добавить к множеству E ребро (x, y), где x и y – позиции ненулевых элементов в таблице степеней.

Код Прюфера устанавливает взаимно однозначное соответствие между множеством всех деревьев с множеством вершин $\{1,2,\ldots,n\}$ и множеством упорядоченных наборов длины $n-2$, составленных из элементов множества $\{1,2,\ldots,n\}$. Отсюда следует формула для числа таких деревьев.

Теорема о числе деревьев (формула Кэли). *Число помеченных деревьев с n вершинами равно* n^{n-2}.

Корневым деревом называют дерево с выделенной вершиной – *корнем*.
Высота корневого дерева – это расстояние от корня до самого удаленного листа.
Если в корневом дереве T путь, соединяющий вершину x с корнем, проходит через вершину y, то говорят, что y – *предок* x, а x – *потомок* y. В частности, каждая вершина является предком и потомком самой себя. Множество всех предков вершины x порождает путь из корня в x. Множество всех потомков вершины x порождает дерево с корнем x, оно называется *ветвью* дерева T в вершине x. Если предок и потомок соединены ребром, то они называются соответственно *отцом* и *сыном*. Компактный и полезный во многих случаях способ задать корневое дерево состоит в указании для каждой вершины ее отца (отцом корня иногда считают саму эту вершину).
Изоморфизм корневых деревьев определяется так же, как изоморфизм графов вообще, но с дополнительным требованием: корень при изоморфизме должен переходить в корень. Проверка изоморфизма корневых деревьев выполняется достаточно легко с помощью специального кодирования. *Канонический код* $c(T)$ дерева T можно построить следующим образом.
Каждой вершине x дерева T приписывается код вершины – слово $c(x)$ в алфавите $\{0,1\}$. Он строится по следующим правилам. Если x – лист (но не корень), то $c(x) = \lambda$, где λ – пустое слово. Код вершины x, не являющейся листом, определяется после того, как построены коды всех ее сыновей. Пусть $\alpha_1, \alpha_2, \ldots, \alpha_k$ – эти коды, упорядоченные лексикографически: $\alpha_1 \le \alpha_2 \le \ldots \le \alpha_k$. Тогда полагаем $c(x) = 0\alpha_1 10\alpha_2 1 \ldots 0\alpha_k 1$. Код корня и является каноническим кодом дерева. На рисунке 4 показаны дерево с корнем в вершине 12 и таблица кодов его вершин.



1	λ
2	λ
3	01
4	λ
5	01
6	λ
7	λ
8	010011
9	λ
10	0011
11	01
12	010100011100110010011 1

Рис. 4.

Канонический код любого дерева обладает свойствами:

1) число нулей в нем равно числу единиц;
2) в каждом его начальном отрезке число единиц не превосходит числа нулей.

Используя эти свойства, можно легко восстановить дерево по его коду. Код дерева T имеет вид: $c(T) = 0c(T_1)10c(T_2)1\ldots 0c(T_k)1$, где $T_1, T_2, \ldots, T_k$ – ветви в сыновьях корня. Если α_1 – первый начальный отрезок слова $c(T)$, в котором число нулей равно числу единиц, то $\alpha_1 = 0c(T_1)1$. Таким образом, код первой ветви получается из слова α_1 отбрасыванием первой и последней букв. Аналогично находятся коды остальных ветвей. Если код какой-то ветви оказывается пустым словом, то эта ветвь состоит из единственной вершины. К остальным ветвям процедура применяется рекурсивно.

Так как дерево по коду восстанавливается однозначно, то из равенства $c(T') = c(T'')$ следует, что корневые деревья T' и T'' изоморфны. Обратно, пусть T' и T'' – изоморфные корневые деревья. Докажем индукцией по числу вершин, что их коды совпадают. Пусть $T_1', T_2', \ldots, T_k'$ и $T_1'', T_2'', \ldots, T_k''$ – ветви в сыновьях корней деревьев T' и T'' соответственно. Тогда существует такая перестановка $(p_1, \ldots, p_k)$ номеров $1, \ldots, k$, что $T_i' \cong T_{p_i}''$ для каждого $i = 1, \ldots, k$. По предположению индукции $c(T_i') = c(T_{p_i}'')$. Значит, наборы $(c(T_1'), \ldots, c(T_k'))$ и $(c(T_1''), \ldots, c(T_k''))$ различаются только порядком слов. После того, как каждый из них будет лексикографически упорядочен, эти наборы станут одинаковыми. Поэтому коды $c(T')$ и $c(T'')$ совпадут. Таким образом, справедлива

Теорема об изоморфизме корневых деревьев. *Корневые деревья T_1 и T_2 изоморфны тогда и только тогда, когда $c(T_1) = c(T_2)$.*

Описанное кодирование можно применить и для распознавания изоморфизма обычных (не корневых) деревьев. Допустим, нужно сравнить деревья T_1 и T_2. Сначала находим центры обоих деревьев. Ясно, что при изоморфизме центр должен перейти в центр. Если центр каждого из деревьев состоит из одной вершины, выберем эти вершины в качестве корней, затем построим канонические коды корневых деревьев и сравним их. Если центр одного дерева – одна вершина, а другого – две, то эти деревья не изоморфны. Допустим, каждый из центров состоит из двух вершин. Если удалить ребро, соединяющее две центральные вершины, то дерево разобьется на два корневых дерева (с корнями в

бывших центральных вершинах). Построив коды этих корневых деревьев, получаем для дерева T_1 пару слов (α_1, β_1), а для дерева T_2 – пару слов (α_2, β_2). Остается сравнить эти (неупорядоченные) пары.

Пусть G – связный граф. Его *каркас* (*остов*, *остовное дерево*) – это остовный подграф, являющийся деревом. В общем случае (граф может быть несвязным) каркас определяется как лес, в котором каждая компонента связности является каркасом компоненты связности графа. У любого графа есть хотя бы один каркас. Его можно найти, удаляя цикловые ребра (ребра, принадлежащие циклам) до тех пор, пока все циклы не будут разрушены.

Если в графе есть циклы, то у него больше одного каркаса. Определить точное число каркасов связного графа позволяет так называемая матричная теорема Кирхгофа. Для графа G определим матрицу $K(G)$ – квадратную матрицу порядка n с элементами

$$k_{ij} = \begin{cases} -1, & \text{если вершины } i \text{ и } j \text{ смежны,} \\ 0, & \text{если вершины } i \text{ и } j \text{ не смежны и } i \neq j, \\ \deg(i), & \text{если } i = j. \end{cases}$$

Заметим, что матрица $K(G)$ – вырожденная, так как сумма элементов каждой строки равна 0.

Теорема Кирхгофа [4]. *Если G – связный граф с не менее чем двумя вершинами, то алгебраические дополнения всех элементов матрицы $K(G)$ равны между собой и равны числу каркасов графа G.*

Двудольные графы

Граф называется *двудольным*, если множество его вершин можно так разбить на два подмножества (*доли*), чтобы каждое ребро соединяло вершины из разных подмножеств. Иначе говоря, каждое из подмножеств порождает пустой подграф.

Теорема Кёнига. *Граф является двудольным тогда и только тогда, когда в нем нет циклов нечетной длины.*

Планарные графы

Плоский граф – это граф, вершинами которого являются точки плоскости, а ребра представлены линиями, соединяющими смежные вершины, при этом никакие два ребра не должны иметь общих точек, кроме инцидентной им обоим вершины. Граф называется *планарным*, если он изоморфен плоскому графу. Этот плоский граф называют также *плоской укладкой* планарного графа.

Если плоскость разрезать по ребрам плоского графа, она распадется на связные части, которые называют *гранями*. Всегда имеется одна неограниченная *внешняя* грань, все остальные грани называются *внутренними*.

Теорема о числе граней (формула Эйлера). *Количество граней в любой плоской укладке планарного графа, имеющего n вершин, m ребер и k компонент связности, равно* $m - n + k + 1$.

Следствие 1. *Если в планарном графе n вершин,* $n \geq 3$, *и m ребер, то* $m \leq 3(n-2)$.

Следствие 2. *Если в планарном графе n вершин,* $n \geq 3$, *m ребер и нет циклов длины* 3, *то* $m \leq 2(n-2)$.

Следствие 3. *В любом планарном графе имеется вершина степени не более 5.*

Операция *подразбиения* ребра (a,b) действует следующим образом. Из графа удаляется это ребро, к нему добавляется новая вершина *c* и два новых ребра (a,c) и (b,c). Граф H называется подразбиением графа G, если первый можно получить из второго последовательностью подразбиений ребер.

Теорема Понтрягина–Куратовского) [4,5]. *Граф планарен тогда и только тогда, когда у него нет подграфа, являющегося подразбиением графа* K_5 *или графа* $K_{3,3}$.

Операция *стягивания* ребра (a,b) определяется следующим образом. Вершины *a* и *b* удаляются из графа, к нему добавляется новая вершина *c* и она соединяется ребром с каждой вершиной, с которой была смежна хотя бы одна из вершин a,b. Граф *G* называется *стягиваемым* к графу *H*, если *H* можно получить из G последовательностью операций стягивания ребер.

Теорема Вагнера [5]. *Граф планарен тогда и только тогда, когда у него нет подграфа, стягиваемого к графу* K_5 *или графу* $K_{3,3}$.

Рассмотрим один из алгоритмов проверки планарности. Предполагается, что граф связен и в нем нет шарниров. Алгоритм строит плоскую укладку графа или определяет, что граф не планарный.

Сначала находится и укладывается какой-нибудь простой цикл, при каждой следующей итерации к уложенному подграфу добавляются новые вершины и ребра.

Пусть подграф F есть уложенная к настоящему моменту часть графа, а H – подграф, порожденный не уложенными еще вершинами. *Сегментом* назовем компоненту связности графа H с добавленными к ней *контактными вершинами* – вершинами графа F, смежными с вершинами этой компоненты, и ребрами, соединяющими контактные вершины с вершинами компоненты. Сегментом считается также ребро, соединяющее две вершины подграфа F, но само этому подграфу не принадлежащее, вместе с инцидентными ему вершинами.

В примере на рисунке 5 выделены вершины и ребра подграфа F, $\Gamma_1 - \Gamma_4$ – грани этого подграфа. Справа показаны три сегмента относительно этого подграфа.

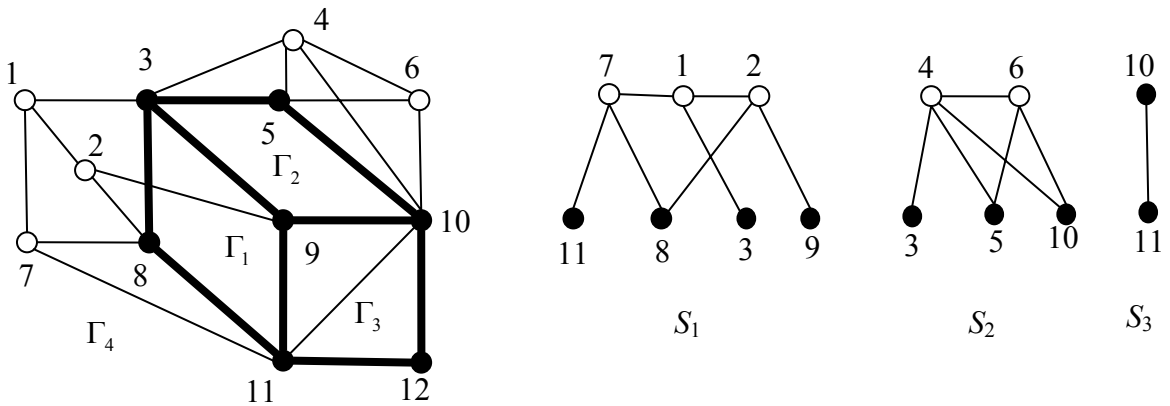


Рис. 5.

Грань называется *допустимой* для некоторого сегмента, если все контактные вершины этого сегмента принадлежат этой грани. Если обозначить через $\Gamma(S)$ множество допустимых граней для сегмента S, то в примере имеем $\Gamma(S_1) = \{\Gamma_1\}$, $\Gamma(S_2) = \{\Gamma_2, \Gamma_4\}$, $\Gamma(S_3) = \{\Gamma_3, \Gamma_4\}$.

Распознавание планарности и построение плоской укладки [4]

Найти в графе какой-нибудь простой цикл и уложить его.
Пока есть не уложенные ребра, повторять
    найти грани уложенного подграфа, сегменты и допустимые грани для каждого сегмента;
    если для некоторого сегмента нет допустимых граней, то граф не планарен, стоп;
    если есть сегменты, имеющие только одну допустимую грань, то пусть S – такой сегмент, иначе выбрать любой сегмент S;
    в сегменте S найти путь P, соединяющий две контактные вершины этого сегмента и не содержащий других контактных вершин;
    уложить путь P в одну из допустимых граней для сегмента S.

В применении к рассмотренному примеру в качестве S должен быть выбран сегмент S_1, а в качестве пути P можно взять, например, путь (3, 1, 7, 8). Он должен быть уложен в грань Γ_1, после чего она разобьется на две новых грани.

Задачи

3.1. Сколько ребер в лесе с n вершинами и k компонентами связности?

3.2. Сколько ребер в графе с n вершинами, если в нем имеется единственный цикл?

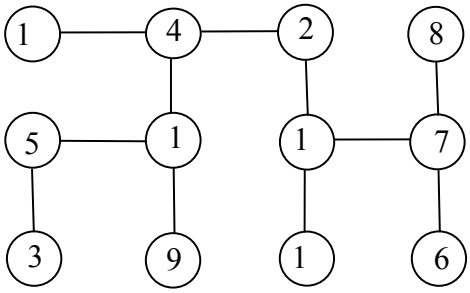
3.3. Перечислите все абстрактные деревья с 6 (7) вершинами.

3.4. В дереве имеется 40 вершин степени 4, все остальные вершины – листья. Сколько листьев в этом дереве?

3.5. В дереве имеется ровно три листа a, b, c, причем $d(a,b) = 37$, $d(a,c) = 44$, $d(b,c) = 21$. Сколько всего вершин в этом дереве?

3.6. Дерево имеет две центральные вершины, а его радиус равен 6. Чему равен диаметр этого дерева?

3.7. Постройте код Прюфера для дерева, изображенного на рисунке.



3.8. Восстановите дерево по коду Прюфера $P(T) = (4, 1, 6, 2, 2, 2, 7, 6)$.

3.9. Сколько имеется помеченных корневых деревьев с n вершинами?

3.10. Сколько различных (помеченных) каркасов у графа K_5?

3.11. Сколько попарно неизоморфных каркасов у графа K_5?

3.12. С помощью теоремы Кирхгофа найдите число каркасов у графа $K_{2,3}$.

3.13. Найдите два неизоморфных дерева с одинаковыми наборами степеней вершин.

3.14. Найдите дерево с наименьшим числом вершин, не имеющее нетривиальных автоморфизмов.

3.15. Постройте канонический код дерева из задачи 3.8, взяв в качестве корня вершину 2.

3. 16. Восстановите дерево по каноническому коду $c(T) = 000111001001011100101010 11$.

3.17. Какие из следующих графов являются двудольными: 1) $\overline{C_5}$; 2) $\overline{2C_4}$; 3) $\overline{K_5 + K_6}$; 4) $\overline{2K_2 \circ 2K_2}$?

3.18. Сколько различных абстрактных двудольных графов можно получить, добавляя одно ребро к графу а) P_7; б) P_8; в) C_{12}?

3.19. Какое наименьшее число ребер нужно удалить из графа K_8, чтобы получился двудольный граф?

3.20. Двудольный граф имеет k компонент связности. Каким числом способов его можно разбить на две доли?

3.21. При каких значениях k граф Q_k является двудольным?

3.22. Какие из следующих графов планарны: 1) $\overline{C_5 + K_1}$; 2) $\overline{K_3 + K_4}$; 3) $\overline{3K_2}$; 4) $\overline{K_1 \circ \overline{K_5}}$?

3.23. Какое наибольшее число граней может быть у плоского графа с 5 вершинами?

3.24. Какое наименьшее число ребер нужно удалить из графа K_6, чтобы получился

планарный граф?

3.25. Какое наименьшее количество новых ребер нужно добавить к графу C_6, чтобы получился непланарный граф?

3.26. При каких значениях *k* граф Q_k является планарным?

3.27. Примените алгоритм распознавания планарности а) к графу из задачи 1.16; б) к графу $\overline{P_6}$; в) к графу $\overline{C_7}$.

3.28. Плоский граф называется *плоской триангуляцией*, если граница каждой грани состоит из трех ребер. Докажите, что каждый плоский граф является подграфом плоской триангуляции. Сколько ребер у плоской триангуляции с *n* вершинами?

3.29*. Граф называется *расщепляемым*, если множество его вершин можно разбить на два подмножества, одно из которых порождает пустой, другое – полный подграф. Докажите, что граф расщепляемый тогда и только тогда, когда в нем нет порожденного подграфа, изоморфного какому-нибудь из графов $2K_2$, C_4, C_5.

3.30*. Докажите, что почти все графы не двудольные.

3.31*. Докажите, что почти все графы не планарные.

3.32*. Покажите, что алгоритм построения кода Прюфера можно реализовать так, что время его работы будет $O(n)$.

3.33*. Покажите, что алгоритм восстановления дерева по коду Прюфера можно реализовать так, что время его работы будет $O(n)$.

4. Методы обхода графа

Решение многих задач на графах основывается на полном обходе графа. Такой обход можно выполнить многими способами, но наибольшее распространение получили две стратегии – поиск в ширину и поиск в глубину. Оба эти метода можно рассматривать как реализации общего плана обхода, состоящего в следующем.

Обход начинается в заранее выбранной стартовой вершине и состоит в систематическом исследовании ребер и посещении вершин. Какие именно действия выполняются при этом, зависит от конкретной задачи, для решения которой и выполняется обход. Но в любом случае тот факт, что данная вершина посещена, запоминается. Вершину, которая еще не посещена, будем называть *новой*. В результате посещения вершина становится *открытой* и остается такой, пока не будут исследованы все инцидентные ей ребра. После этого она превращается в *закрытую*.

Очередной шаг обхода начинается с выбора какой-либо вершины x из множества открытых, она становится *активной*. Если среди инцидентных ей ребер имеются неисследованные, то выбирается такое ребро (x, y). Если вершина y новая, то она посещается, ребро (x, y) при этом классифицируется как *прямое*. Если же y не новая, то ребро (x, y) считается *обратным* (ведущим в уже посещенную вершину).

Если все ребра, инцидентные активной вершине, уже исследованы, она становится закрытой. Эти действия повторяются до тех пор, пока множество открытых вершин не станет пустым. Если при этом еще остались новые вершины, то выбирается и посещается одна из таких вершин и процесс повторяется.

По окончании обхода прямые ребра образуют каркас графа. В основе большинства применений поиска в ширину и поиска в глубину лежат свойства этого каркаса.

Основное различие между поиском в ширину и поиском в глубину состоит в том, как выбирается активная вершина.

Поиск в ширину

При поиске в ширину в качестве активной вершины выбирается та из открытых, которая была посещена раньше других. Для реализации такого правила выбора удобно использовать очередь для хранения множества открытых вершин.

В приводимом описании процедуры поиска в ширину (BFS – Breadth First Search) a – стартовая вершина, Q – очередь открытых вершин.

Procedure BFS(a)

1 посетить a;
2 **while** $Q \neq \varnothing$ **do**
3 $x \Leftarrow Q$;
4 **for** $y \in N(x)$ **do**
5 **if** y новая **then** посетить y

Процедура посещения, кроме действий, связанных с решением конкретной задачи, должна включать две обязательных операции: вершину нужно пометить как посещенную (т.е. не новую) и поместить в очередь Q.

Процедура BFS обеспечивает посещение всех вершин из компоненты связности, содержащей вершину a. Полный обход графа с множеством вершин V может быть осуществлен следующим образом.

Поиск в ширину

6 пометить все вершины как новые;
7 **for** $x \in V$ **do if** x новая **then** BFS(x)

Если граф задан списками смежности, то внутренний цикл в строке 4 процедуры BFS повторяется $\sum_{x \in V} \deg(x) = 2m$ раз. Кроме того, цикл в строке 7 повторяется n раз. Поэтому общая трудоемкость поиска в ширину оценивается как $O(m+n)$. Если же граф задан матрицей смежности, то для сканирования окрестности вершины (строка 4) необходимо полностью просмотреть соответствующую строку этой матрицы и общая трудоемкость будет $O(n^2)$.

Рассмотрим примеры применения поиска в ширину для решения конкретных задач.

Компоненты связности. Рассмотрим задачу выявления компонент связности графа. Ответ нужно получить в виде таблицы, в которой для каждой вершины x должен быть указан номер компоненты $comp(x)$, которой эта вершина принадлежит. Для решения этой задачи достаточно ввести переменную c со значением, равным текущему номеру компоненты, и каждый раз при посещении новой вершины x присваивать значение $comp(x) = c$. Значение c первоначально устанавливается равным 0 и модифицируется при каждом вызове процедуры BFS.

Кратчайшие пути и расстояния. Допустим, поиск в ширину выполняется на связном графе. Каркас, образованный прямыми ребрами, можно рассматривать как корневое дерево с корнем в стартовой вершине. Оно называется *BFS-деревом*. В процессе обхода легко построить таблицу отцов F, задающую это дерево: достаточно при посещении вершины y при активной вершине x присвоить значение $F(y) = x$. BFS-дерево с данным корнем не единственно, оно зависит от того, в каком порядке рассматриваются ребра, инцидентные активной вершине. Но все BFS-деревья обладают одним свойством, на котором основаны важнейшие применения поиска в ширину. Корневой каркас связного графа называется *деревом кратчайших путей*, если путь от любой вершины до корня в этом каркасе является кратчайшим путем между этими вершинами во всем графе.

Теорема о BFS-дереве. *Любое BFS-дерево является деревом кратчайших путей.*

Таким образом, однократным выполнением поиска в ширину можно найти кратчайшие пути и расстояния от данной вершины до всех остальных.

Центр дерева. Центр дерева можно найти следующим образом. Выберем в данном дереве произвольную вершину a. Выполнив поиск в ширину из вершины a, найдем наиболее удаленную от нее вершину b. Выполнив второй раз поиск в ширину со стартом в вершине b, найдем наиболее удаленную от нее вершину c. Центр пути, соединяющего b и c, является центром всего дерева.

Поиск в глубину

При поиске в глубину в качестве активной выбирается та из открытых вершин, которая была посещена последней. Для реализации такого правила выбора наиболее удобной структурой хранения множества открытых вершин является стек: открываемые вершины складываются в стек в том порядке, в каком они посещаются, а в качестве активной выбирается последняя вершина.

Обозначим стек для открытых вершин через S, а верхний элемент стека – через $top(S)$. Тогда процедура обхода одной компоненты связности методом поиска в глубину со стартовой вершиной a может быть записана следующим образом (DFS – от Depth First Search).

Procedure DFS(a)

1 посетить a;
2 **while** $S \neq \varnothing$ **do**
3 $x := top(S)$;
4 **if** имеются неисследованные ребра, инцидентные вершине x

5 **then** выбрать такое ребро (x, y);
6 **if** y новая **then** посетить y
7 **else** удалить x из S

При посещении вершина помечается как посещенная и помещается в стек.

Алгоритм обхода всего графа – тот же, что и в случае поиска в ширину. Остаются верными и оценки трудоемкости: $O(m+n)$, если граф представлен списками смежности, $O(n^2)$, если матрицей смежности.

Поиск в глубину можно применить для нахождения компонент связности графа или для построения каркаса точно таким же образом, как поиск в ширину. Для связного графа каркас с корнем в стартовой вершине, получаемый поиском в глубину, называется *DFS-деревом*.

Если в некотором связном графе выбран корневой каркас, то все ребра, не принадлежащие каркасу, можно разделить на две категории: ребро назовем *продольным*, если одна из его вершин является предком другой, в противном случае назовем его *поперечным*.

Теорема о DFS-дереве. *Относительно DFS-дерева все обратные ребра являются продольными.*

На этом свойстве основаны многие применения поиска в глубину. Рассмотрим некоторые из них.

Шарнир. Допустим, требуется выяснить, является ли шарниром некоторая вершина x данного графа (предположим, что он связный). Для решения этой задачи достаточно выполнить поиск в глубину со стартом в вершине x и построением DFS-дерева. Если теперь вершину x удалить из графа, то, ввиду отсутствия поперечных ребер, он распадется на столько компонент связности, сколько сыновей у вершины x в DFS-дереве. Значит, вершина x является шарниром тогда и только тогда, когда она имеет в DFS-дереве с корнем x степень 2 или больше.

Все шарниры. Однократным поиском в глубину можно найти все шарниры графа. Будем нумеровать вершины при обходе графа в том порядке, в котором они посещаются. Номер, полученный вершиной x, обозначим через $Dn(x)$, он называется *глубинным номером*. Введем на множестве вершин еще одну функцию L, связанную с DFS-деревом: значением $L(x)$ является наименьший из глубинных номеров вершин, смежных с потомками вершины x. Если вершина y является сыном вершины x, то $L(y) \le Dn(x)$ (так как вершина y является потомком самой себя и смежна с вершиной x).

Теорема о шарнирах. *Корень DFS-дерева является шарниром графа тогда и только тогда, когда его степень в этом дереве больше 1. Вершина x, отличная от корня, является шарниром тогда и только тогда, когда у нее в DFS-дереве имеется такой сын y, что* $L(y) = Dn(x)$.

Функцию L можно определить рекурсивно – если мы знаем ее значения для всех сыновей вершины x и глубинные номера всех вершин, смежных с x и не являющихся ее сыновьями, то $L(x)$ есть минимум из всех этих величин, то есть

$$L(x) = \min\left(\min_{y \in A} L(y), \quad \min_{y \in B} Dn(y) \right),$$

где A обозначает множество всех сыновей вершины x, а B – множество всех остальных вершин, смежных с x. Ниже приведена основанная на этом равенстве процедура вычисления функции L и выявления шарниров для одной компоненты связности. Вначале каждой вершине присвоен нулевой глубинный номер и это нулевое значение используется затем как признак того, что вершина новая. Переменная c хранит текущий глубинный номер, вначале $c = 0$. Предполагается, что окрестность каждой вершины реализована в виде списка и запись $y \Leftarrow N(x)$ означает, что в качестве y берется очередная вершина из списка $N(x)$ и при этом вершина y из списка удаляется.

Procedure Шарниры(a)

```
1     посетить(a);
2     while S ≠ ∅ do
3           x := top(S);
4           if N(x) ≠ ∅
5           then   y ⇐ N(x);
6                  if   Dn(y) = 0
7                  then посетить( y)
8                  else  L(x) := min(L(x), Dn(y))
9           else  y ⇐ S;
10                 if  y ≠ a
11                 then   x := top(S);
12                        if  x ≠ a  then  L(x) := min(L(x), L(y));
13                                         if  L(y) = Dn(x)  then пометить x как шарнир
```

Procedure посетить(x)

```
1     x ⇒ S;
2     c := c + 1;
3     Dn(x) := c;
4     L(x) := c
```

Этот алгоритм находит все шарниры, кроме стартовой вершины a (если она является шарниром). Из теоремы видно, что эта вершина должна обрабатываться особо – нужно просто отслеживать ее степень в DFS-дереве. Это не включено в описание алгоритма, чтобы не загромождать его.

Блоки. *Блок* графа – это его максимальный связный подграф, не имеющий собственных шарниров (т.е. некоторые шарниры графа могут принадлежать блоку, но своих шарниров у блока нет). Каждое ребро графа принадлежит в точности одному блоку, а вершина может принадлежать нескольким блокам, но только в том случае, когда она – шарнир. Строение связного графа G, состоящего из нескольких блоков, описывается *BC-деревом*, которое строится следующим образом. В этом дереве вершины двух типов – одни соответствуют блокам графа G (вершины-блоки), другие – его шарнирам (вершины-шарниры). Вершина-блок соединяется в BC-дереве ребром с вершиной-шарниром, если соответствующий шарнир принадлежит соответствующему блоку.

Описанный выше алгоритм выявления шарниров нетрудно приспособить для отыскания всех блоков графа. Достаточно завести еще один стек и складывать в него все посещаемые вершины. При обнаружении шарнира x (строка 13) достаточно извлечь из стека все вершины, содержащиеся в его верхней части вплоть до вершины y и добавить к ним еще вершину x (не удаляя ее, в отличие от других, из стека). Это множество вершин образует очередной блок.

Задачи

4.1. Используя метод поиска в ширину, найдите компоненты связности графа, заданного матрицей смежности:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

4.2. Методом поиска в ширину постройте дерево кратчайших путей из вершины 1 для графа, заданного списками смежности:

1: 2,9; 2: 1, 3, 7, 8; 3: 2, 7; 4: 5, 9; 5: 4, 6, 7, 9, 10; 6: 5, 7, 10; 7: 2, 3, 5, 6, 8; 8: 2, 7, 9; 9: 1, 4. 5, 8, 10; 10: 5, 6, 9

4.3. Лес задан списками смежности. Какие из следующих оценок времени работы верны для поиска в ширину?

1) $O(n)$; 2) $O(m)$; 3) $O(m+n)$; 4) $O(n^2)$; 5) $O(mn)$.

4.4. Лес задан матрицей смежности. Какие из следующих оценок времени работы верны для поиска в ширину?

1) $O(n)$; 2) $O(m)$; 3) $O(m+n)$; 4) $O(n^2)$; 5) $O(mn)$.

4.5. Какова будет высота BFS-дерева, если поиск в ширину применяется к графу

1) K_n; 2) $K_{p,q}$, $p,q>1$; 3) Q_k; 4) C_n; 5) $P_k \times P_l$, старт в вершине степени 2?

4.6. Пусть h –высота BFS-дерева для графа G. Может быть $h > \mathrm{diam}(G)$? $h < \mathrm{rad}(G)$? Всегда ли можно выбрать стартовую вершину так, чтобы было $h = \mathrm{diam}(G)$? $h = \mathrm{rad}(G)$?

4.7. Ребро (x, y) является обратным ребром относительно BFS-дерева с корнем a. Какие из следующих соотношений могут выполняться? А если граф двудольный?

1) $d(a,x) = d(a,y)$; 2) $|d(a,x) - d(a,y)| = 1$; 3) $|d(a,x) - d(a,y)| = 2$; 4) $|d(a,x) - d(a,y)| = 3$.

4.8. Два человека имеют восьмилитровый кувшин, наполненный вином, и два пустых кувшина:
а) трехлитровый и двухлитровый;
б) пятилитровый и шестилитровый.
Они могут переливать вино из одного кувшина в другой, пока либо первый не опустеет, либо второй не наполнится. Могут они разделить вино поровну с помощью таких действий? Какое наименьшее число переливаний потребуется?

4.9. Используя алгоритм поиска в глубину, найдите компоненты связности графа, заданного списками смежности:

1: 2, 4, 6;	3: 5, 9;	5: 3, 9;	7: 8, 10;	9: 3, 5;	11: 8, 12;
2: 1;	4: 1;	6: 1;	8: 7, 10, 11;	10: 7, 8, 12;	12: 10, 11.

4.10. Постройте DFS-дерево с корнем в вершине 1 для графа, заданного списками смежности:

1: 5, 6, 7, 8;	3: 2, 4;	5: 1, 6, 10;	7: 1, 2, 4, 8;	9: 8;
2: 3, 4, 7, 8;	4: 2, 3, 7:	6: 1, 5;	8: 1, 2, 7, 9;	10: 5.

4.11. Планарный граф задан списками смежности. Какие из следующих оценок времени работы верны для поиска в глубину?
1) $O(n)$; 2) $O(m)$; 3) $O(m+n)$; 4) $O(n^2)$; 5) $O(mn)$.

4.12. Планарный граф задан матрицей смежности. Какие из следующих оценок времени работы верны для поиска в глубину?
1) $O(n)$; 2) $O(m)$; 3) $O(m+n)$; 4) $O(n^2)$; 5) $O(mn)$.

4.13. Дерево задано списками смежности. Какие из следующих оценок времени работы верны для поиска в глубину?
1) $O(n)$; 2) $O(m)$; 3) $O(m+n)$; 4) $O(n^2)$; 5) $O(mn)$.

4.14. Сколько различных DFS-деревьев можно построить для графа 1) K_n; 2) P_n; 3) C_n?

4.15. Пусть h –высота DFS-дерева для графа G. Может быть а) $h > \mathrm{diam}(G)$?
б) $h < \mathrm{rad}(G)$?

4.16. Какой может быть максимальная высота DFS-дерева, если поиск в глубину применяется к графу 1) $P_k \times P_l$; 2) Q_k; 3) $K_{p,q}$, $p > q$?

4.17. Ребро (x, y) является обратным ребром относительно DFS-дерева с корнем a.
Какие из следующих соотношений могут выполняться (d обозначает расстояние между вершинами в дереве)? А если граф двудольный?
1) $d(a,x) = d(a,y)$; 3) $|d(a,x) - d(a,y)| = 2$;
2) $|d(a,x) - d(a,y)| = 1$; 4) $|d(a,x) - d(a,y)| = 3$.

4.18. Постройте DFS-дерево и таблицы функций Dn и L для графа, заданного списками смежности:

1: 3, 4	3: 1, 6	5: 2, 7, 11	7: 5, 6, 10, 11	9: 6, 8	11: 5,7,12
2: 5	4: 1, 6	6: 3, 4, 7, 8, 9	8: 6, 9	10: 7	12: 11.

Найдите все шарниры и блоки этого графа, постройте BC-дерево.

4.19. Каждый блок графа состоит из трех вершин. Сколько вершин и ребер в этом графе, если его BC-дерево представляет собой а) путь P_{21}; б) путь P_{100}; в) граф $K_{1,20}$?

4.20*. Разработайте алгоритм, который за время $O(n)$ проверяет, является ли данный граф деревом.

4.21*. Разработайте алгоритм, проверяющий, является ли данный граф двудольным.

4.22*. Разработайте алгоритм, который находит кратчайший цикл, проходящий через данную вершину.

4.23*. Докажите, что описанный выше алгоритм нахождения центра дерева с помощью двойного обхода в ширину действительно находит центр любого дерева. Приведите пример графа, для которого это не так.

4.24*. Доработайте алгоритм выявления шарниров так, чтобы он правильно обрабатывал корень DFS-дерева (т.е. решал, является ли эта вершина шарниром).

4.25*. Разработайте алгоритм, определяющий, является ли данное ребро перешейком графа.

4.26*. Разработайте алгоритм выявления всех перешейков графа.

4.27*. (Задача об одностороннем движении) Докажите, что ребра обыкновенного графа можно ориентировать так, что получится сильно связный орграф, тогда и только тогда, когда в нем нет перешейков. Разработайте алгоритм построения такой ориентации.

5. Циклы

Эйлеровы циклы

Эйлеровым циклом (*путем*) называется цикл (путь), проходящий через все ребра графа. Граф, в котором имеется эйлеров цикл, называют *эйлеровым графом*.

Теорема об эйлеровом цикле. *Связный граф эйлеров тогда и только тогда, когда в нем степени всех вершин четны.*

Следствие (об эйлеровом пути). *Эйлеров путь в связном графе существует тогда и только тогда, когда в нем имеется не более двух вершин нечетной степени.*

Ниже приводится алгоритм построения эйлерова цикла. Предполагается, что условия существования уже проверены – граф связен и степени четны. В алгоритме строится путь, начинающийся в произвольно выбранной стартовой вершине *a*, при этом каждый раз для дальнейшего продвижения выбирается любое еще не пройденное ребро. Вершины пути накапливаются в стеке *S* (они могут повторяться). Когда наступает тупик – все ребра, инцидентные последней вершине пути, уже пройдены, производится возвращение вдоль пройденного пути, пока не встретится вершина, которой инцидентно не пройденное ребро. При возвращении вершины перекладываются из стека *S* в другой

стек C. Затем возобновляется движение вперед по не пройденным ребрам, пока снова не наступает тупик, и т. д. Процесс заканчивается, когда стек S оказывается пустым. В этот момент в стеке C находится последовательность вершин эйлерова цикла.

Построение эйлерова цикла

```
1   выбрать произвольно вершину a;
2   a ⇒ S;
3   while S ≠ ∅ do
4       x := top(S);
5       if имеется не пройденное ребро (x, y)
6           then пометить ребро (x, y) как пройденное;
7                y ⇒ S;
8           else переместить вершину x из S в C
```

Трудоемкость этого алгоритма оценивается как $O(m)$. Этот вывод справедлив лишь при определенных предположениях о том, как задан граф. Можно, например, использовать две структуры:

- массив ребер, в котором в позиции, соответствующей ребру, помещается запись, указывающая две вершины этого ребра и пометку о том, пройдено ли это ребро;
- списки инцидентности, в которых для каждой вершины перечисляются инцидентные ей ребра.

В ориентированном графе эйлеров цикл – это ориентированный цикл, проходящий через все ребра.

Теорема об эйлеровом цикле в орграфе. *Эйлеров цикл в сильно связном орграфе существует тогда и только тогда, когда в нем для каждой вершины x выполняется равенство* $\deg^+(x) = \deg^-(x)$.

Одно из применений эйлеровых циклов в орграфе – построение так называемых последовательностей де Брейна. Слово $a_1a_2\ldots a_n$ в алфавите $\{0,1\}$ является последовательностью де Брейна порядка k, если в слове $a_1a_2\ldots a_na_1\ldots a_{k-1}$ среди отрезков длины k каждое слово длины k встречается ровно один раз. Иначе говоря, если последовательность де Брейна свернуть в кольцо и двигать вдоль этого кольца окно, в котором видны k последовательных букв, то в течение одного оборота каждое слово длины k появится в окне один раз. Ясно, что $n = 2^k$. Пример последовательности де Брейна порядка 3: 00010111.

Для построения последовательностей де Брейна можно использовать граф де Брейна. Вершинами этого графа являются всевозможные слова длины $k-1$, из каждой вершины $x_1x_2\ldots x_{k-1}$ выходят ровно два ребра: в вершины $x_2x_3\ldots x_{k-1}0$ и $x_2x_3\ldots x_{k-1}1$. Первому ребру приписывается буква 0, второму – буква 1. Очевидно, входящих ребер тоже будет два: из вершин $0x_1\ldots x_{k-2}$ и $1x_1\ldots x_{k-2}$. Очевидно также, что этот граф сильно связен – от любого слова можно перейти к любому другому, добавляя буквы в конце. Значит, в нем имеется эйлеров цикл. Двигаясь вдоль такого цикла и выписывая буквы, приписанные ребрам, получим последовательность де Брейна. На рисунке 6 показан граф де Брейна для $k = 3$.

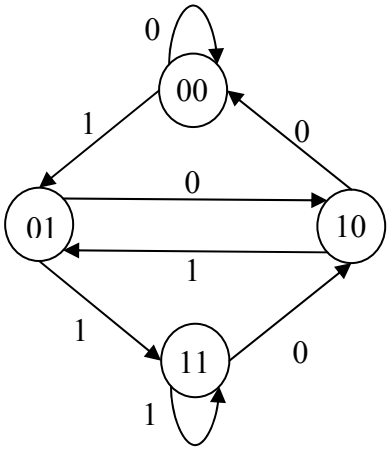


Рис. 6.

Гамильтоновы циклы

Гамильтоновым циклом (*путем*) называют простой цикл (путь), содержащий все вершины графа.

Внешне определение гамильтонова цикла похоже на определение эйлерова цикла. Однако есть кардинальное различие в сложности решения соответствующих задач. Мы видели, что имеется достаточно простой критерий существования эйлерова цикла и эффективный алгоритм его построения. Для гамильтоновых же циклов (и путей) неизвестно никаких просто проверяемых необходимых и достаточных условий их существования, а все известные алгоритмы требуют для некоторых графов перебора большого числа вариантов. Такие задачи называют задачами переборного типа или неподдающимися задачами. Очень многие известные задачи относятся к разряду неподдающихся, среди них немало задач на графах. Существует математическая теория сложности алгоритмов и задач, в которой под эффективным алгоритмом понимают алгоритм, время работы которого ограничено сверху полиномом от длины записи входных данных. Выводы этой теории делают весьма правдоподобным предположение о том, что для многих неподдающихся задач, в том числе и для задачи о гамильтоновом цикле, не существует эффективных алгоритмов.

Перебор вариантов в задаче о гамильтоновом цикле (построить гамильтонов цикл или убедиться, что его не существует) можно организовать с помощью дерева путей. Это дерево представляет всевозможные простые пути в данном графе, начинающиеся в некоторой вершине a. Его вершины соответствуют вершинам графа, при этом каждая вершина графа может быть представлена в дереве несколько раз. Оно может быть построено следующим образом. Выберем в графе произвольно вершину a и объявим ее корнем дерева. Вершины, смежные с a, добавим к дереву в качестве сыновей вершины a. Для каждой вершины b, добавленной к дереву, рассмотрим все смежные с ней вершины и те из них, которые не лежат на пути (в дереве) из a в b, добавим к дереву в качестве сыновей вершины b. Процесс построения дерева заканчивается, когда к нему уже невозможно добавить новую вершину. Очевидно, что каждому пути в дереве, начинающемся в корне, соответствует точно такой же (простой) путь в графе, и обратно, каждому простому пути в графе с началом в вершине a соответствует такой же путь в дереве. Каждой вершине, находящейся в дереве на расстоянии $n-1$ от корня, соответствует гамильтонов путь в графе с началом в вершине a, а если последняя вершина этого пути смежна с a, то получаем гамильтонов цикл. Если высота дерева путей не

превосходит $n-2$, то в графе нет гамильтоновых путей, начинающихся в *a*, и нет гамильтоновых циклов. На рисунке 7 показаны граф и его дерево путей из вершины 1.

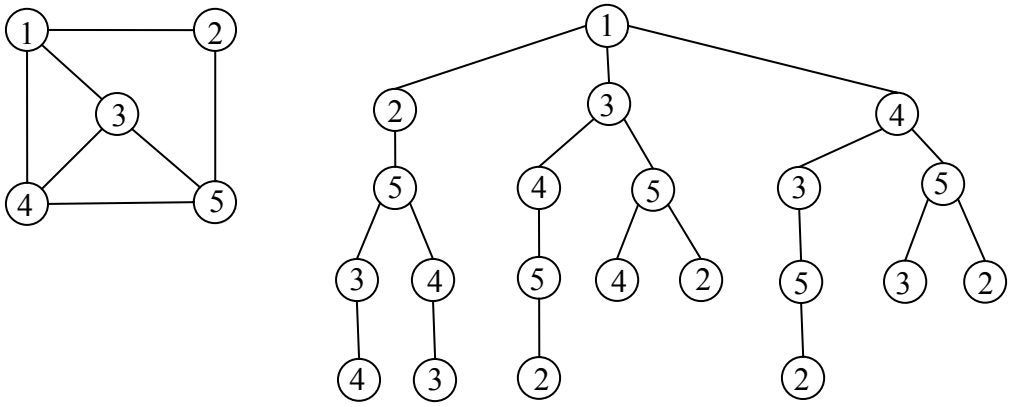


Рис. 7.

Дерево путей можно построить с помощью процедуры типа поиска в ширину: вершины исследуются и добавляются к дереву в порядке возрастания их расстояний (в дереве) от корня. Отличие от обычного поиска в ширину состоит в том, что вершина графа может добавляться к дереву несколько раз.

Дерево путей может быть очень большим. Для полного графа K_m в этом дереве будет $(n-1)!$ листьев. Однако, если задача состоит в том, чтобы найти один гамильтонов путь или цикл (если такой существует), то совсем не обязательно строить это дерево целиком. Можно организовать обход этого дерева с помощью, скажем, поиска в глубину, без его явного построения. Для полного графа такой алгоритм даст ответ очень быстро, но в некоторых случаях время его работы тоже растет с факториальной скоростью. Например, для графа $K_{n-1}+K_1$, если в качестве стартовой выбрана не изолированная вершина, будут рассмотрены все $(n-2)!$ простых путей длины $n-2$ в большой компоненте.

Кодом Грея порядка n называется расположение всех 2^n двоичных слов длины n в последовательность, в которой любые два соседних слова различаются ровно в одной букве. Двоичные слова длины *n* можно рассматривать как вершины графа Q_n – ребра этого графа соединяют как раз пары слов, различающихся в одной букве. Значит, код Грея есть не что иное, как гамильтонов путь в графе Q_n. Существование такого пути при любом *n* легко доказать с помощью индукции, используя равенство $Q_n = Q_{n-1} \times K_2$.

В ориентированном графе гамильтонов цикл (путь) – это ориентированный цикл (путь), проходящий через каждую вершину точно один раз. Задачи о гамильтоновых циклах и путях для орграфов в общем случае тоже неподдающиеся. В то же время имеется важный класс графов, для которых эти задачи имеют простое решение. Это так называемые турниры.

Орграф называется *турниром*, если для каждых двух вершин в нем имеется единственное ребро, соединяющее эти вершины.

Теорема о гамильтоновых путях в турнирах. *В любом турнире имеется гамильтонов путь.*

Для нахождения гамильтонова пути в турнире можно использовать следующее легко доказываемое свойство: если $x_1, x_2, \ldots, x_k$ – ориентированный простой путь в турнире, y – вершина, не принадлежащая этому пути, то хотя бы одна из последовательностей $y, x_1, x_2, \ldots, x_k$, $x_1, y, x_2, \ldots, x_k$, …, $x_1, x_2, \ldots, x_k, y$ тоже является ориентированным простым путем.

Аналогично обстоит дело с гамильтоновыми циклами в турнирах (см. задачу 5.25).

Пространство циклов

Циклы – очень важная часть структуры графа, с ними приходится иметь дело при решении многих задач. В графе может быть очень много циклов (см. задачу 5.15), поэтому полезно иметь средства для компактного описания множества всех циклов данного графа и манипулирования ими. Одно из наиболее удобных средств такого рода предоставляет аппарат линейной алгебры.

Зафиксируем некоторый граф $G = (V, E)$. В этом разделе слово «подграф» будет означать «остовный подграф графа G». Занумеруем ребра графа G: $E = \{e_1, e_2, \ldots, e_m\}$. Подграф H можно задать характеристическим вектором $\alpha(H) = (x_1, x_2, \ldots, x_m)$ его множества ребер, здесь $x_i = 1$, если ребро e_i принадлежит подграфу, $x_i = 0$, если не принадлежит. Тем самым устанавливается взаимно однозначное соответствие между. Тем самым устанавливается взаимно однозначное соответствие между подграфами и двоичными m-мерными векторами. В частности, вектор, у которого все компоненты единицы, соответствует самому графу G, а вектор, у которого все компоненты нули – пустому графу с множеством вершин V, последний будем обозначать буквой O.

Пусть H_1 и H_2 – два подграфа. Вектор $\alpha(H_1) \oplus \alpha(H_2)$, где $\oplus$ означает векторную сумму по модулю 2, задает подграф, который будем обозначать $H_1 \oplus H_2$ и называть *суммой по модулю 2* графов H_1 и H_2. Множество ребер графа $H_1 \oplus H_2$ является симметрической разностью множеств ребер графов H_1 и H_2. Следующие свойства введенной операции очевидны или легко проверяются.

1) Коммутативность: $H_1 \oplus H_2 = H_2 \oplus H_1$ для любых H_1 и H_2.

2) Ассоциативность: $H_1 \oplus (H_2 \oplus H_3) = (H_1 \oplus H_2) \oplus H_3$ для любых H_1, H_2, H_3.

3) $H \oplus O = H$ для любого H.

4) $H \oplus H = O$ для любого H.

Благодаря свойству ассоциативности мы можем образовывать выражения вида $H_1 \oplus H_2 \oplus \ldots \oplus H_k$, не используя скобок для указания порядка действий. Легко понять, что ребро принадлежит графу $H_1 \oplus H_2 \oplus \ldots \oplus H_k$ тогда и только тогда, когда оно принадлежит нечетному количеству из графов $H_1, H_2, \ldots, H_k$.

Определим еще операцию умножения подграфа H на элементы множества $\{0,1\}$: $0 \cdot H = O$, $1 \cdot H = H$. Теперь можно образовывать линейные комбинации подграфов вида $a_1 H_1 \oplus a_2 H_2 \oplus \ldots \oplus a_k H_k$ с коэффициентами $a_i \in \{0,1\}$. Каждая такая комбинация тоже является подграфом. Это позволяет рассматривать множество всех подграфов как линейное векторное пространство с операцией сложения векторов по модулю 2 и умножения их на числа 0, 1. Оно называется *пространством подграфов* графа G.

Далее в этом разделе *циклом графа G* будем называть его остовный подграф, у которого одна компонента связности является простым циклом, а остальные –

изолированными вершинами. Остовный подграф, у которого степени всех вершин четны, называется *квазициклом*. Таким образом, любой цикл является квазициклом и граф O – квазицикл.

Теорема о квазициклах. *Множество всех квазициклов данного графа замкнуто относительно сложения по модулю 2.*

Из этой теоремы следует, что множество всех квазициклов графа G является подпространством пространства подграфов, оно называется *пространством циклов* графа.

Компактное представление линейного векторного пространства дает его базис. Базис пространства циклов называют просто *базисом циклов.* Построить базис циклов можно следующим образом. Выберем в графе G какой-нибудь каркас T. Допустим, имеется s ребер графа, не принадлежащих каркасу. Если добавить к T одно из этих ребер, то в полученном графе образуется единственный цикл. Проделав это с каждым ребром, не принадлежащим каркасу, получим семейство из s циклов, они называются *фундаментальными циклами* относительно каркаса T.

Теорема о фундаментальных циклах. *Множество всех фундаментальных циклов относительно любого каркаса T графа G образует базис пространства циклов этого графа.*

Из этой теоремы следует, что размерность пространства циклов графа равна числу ребер, не входящих в его каркас. Так как каркас содержит $n-k$ ребер, где k – число компонент связности графа, то эта размерность равна $\nu(G)=m-n+k$. Это число называют *цикломатическим числом* графа.

Каркас графа можно построить многими способам. Для построения базиса циклов графа особенно удобен поиск в глубину благодаря основному свойству DFS-дерева – каждое обратное ребро относительно этого дерева является продольным. Это означает, что из двух вершин такого ребра одна является предком другой в DFS-дереве. Каждое такое ребро в процессе поиска в глубину встретится дважды – один раз, когда активной вершиной будет предок, другой раз, когда ею будет потомок. В этом последнем случае искомый фундаментальный цикл состоит из рассматриваемого обратного ребра и участка пути в DFS-дереве, соединяющего эти две вершины. Но этот путь так или иначе запоминается в процессе обхода в глубину, так как он необходим для последующего возвращения. Если, например, для хранения открытых вершин используется стек, то вершины этого пути находятся в верхней части стека. В любом случае этот путь легко доступен и цикл находится без труда.

Хотя сам поиск в глубину выполняется за линейное от числа вершин и ребер время, решающее влияние на трудоемкость алгоритма оказывает необходимость запоминать встречающиеся циклы. Подсчитаем суммарную длину фундаментальных циклов, полученных с помощью поиска в глубину для полного графа с n вершинами. DFS-дерево в этом случае является простым путем, относительно него будет $n-2$ цикла длины 3, $n-3$ цикла длины 4, …, 1 цикл длины n. Сумма длин всех фундаментальных циклов будет равна

$$\sum_{i=1}^{n-2} i(n+1-i)=\frac{n^3+3n^2-16n+12}{6}.$$

Таким образом, на некоторых графах число операций этого алгоритма будет величиной порядка n^3.

С пространством циклов тесно связано *пространство разрезов* графа. Пусть $A \subseteq V$, $\overline{A} = V - A$. *Разрез* графа G, определяемый множеством A, – это остовный подграф, ребрами которого являются все ребра графа, соединяющие вершины из A с вершинами из $\overline{A}$.

Теорема о разрезах. *Множество всех разрезов данного графа замкнуто относительно сложения по модулю 2.*

Разрез, определяемый одноэлементным множеством A, называется *элементарным*. Базис пространства разрезов связного графа можно получить, если взять все его элементарные разрезы, кроме одного (любого). Для несвязного графа это нужно проделать для каждой компоненты связности. Таким образом, размерность пространства разрезов графа с k компонентами равна $n-k$.

Пусть $\alpha(H_1) = (x_1,\ldots,x_m)$ и $\alpha(H_2) = (y_1,\ldots,y_m)$ – характеристические векторы двух остовных подграфов графа G. Будем говорить, что эти подграфы *ортогональны*, если $x_1y_1 \oplus x_2y_2 \oplus \ldots \oplus x_my_m = 0$. Это равносильно тому, что подграфы имеют четное число общих ребер.

Теорема о циклах и разрезах. *Любой цикл данного графа ортогонален любому его разрезу.*

Так как сумма размерностей пространств циклов и разрезов равна размерности пространства подграфов, то из этой теоремы следует, что пространства циклов и разрезов являются ортогональными дополнениями друг друга. Это можно использовать для построения базиса циклов по матрице инцидентности. Продемонстрируем это на примере. Пусть требуется найти базис циклов для графа, показанного на рисунке 8. Построим его

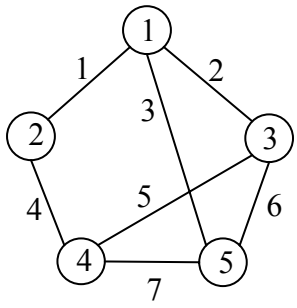


Рис. 8

матрицу инцидентности и удалим из нее одну (любую) строчку (в случае несвязного графа удаляется по одной строке из каждой компоненте связности):

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

Строки этой матрицы – характеристические векторы элементарных разрезов, образующих базис разрезов. Так как пространство циклов является ортогональным дополнением пространства разрезов, то остается найти фундаментальную систему решений системы линейных однородных уравнений с этой матрицей (над полем из двух элементов). Для

этого с помощью операций над строками преобразуем матрицу так, чтобы в первых столбцах образовалась единичная подматрица:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

Удаляем первые четыре столбца, оставшуюся матрицу транспонируем и приписываем кт ней справа единичную подматрицу:

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Строки полученной матрицы представляют базис циклов.

Задачи

5.1. Сколько существует абстрактных эйлеровых графов с 5 вершинами?

5.2. Граф $K_{3,5}$ нужно превратить в эйлеров, изменяя (удаляя и добавляя) ребра. Каково наименьшее число изменений, если разрешается а) только удалять ребра; б) только добавлять ребра; в) и удалять, и добавлять?

5.3. Проследите работу алгоритма построения эйлерова цикла на графе $K_{4,4}$. Вершины одной доли имеют номера 1-4, другой – номера 5-8, старт в вершине1. Всякий раз, когда имеется несколько непройденных ребер, по которым можно продолжить движение, выбирается то из них, которое ведет в вершину с наименьшим номером.

5.4. Алгоритм построения эйлерова цикла применяется к графу P_n. Каков будет ответ (содержимое стека *C* в конце работы алгоритма)?

5. 5. Что нужно изменить в алгоритме построения эйлерова цикла, чтобы получился алгоритм построения эйлерова пути в графе с двумя вершинами нечетной степени?

5.6. С помощью графа де Брейна найдите последовательность де Брейна 1) порядка 4 для двухбуквенного алфавита; 2) порядка 2 для четрехбуквенного алфавита.

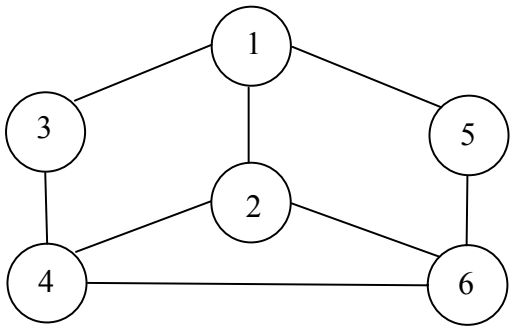
5.7. Сколько существует абстрактных графов с 5 вершинами, имеющих гамильтонов цикл?

5.8. В каких из следующих графов имеется гамильтонов цикл?
1) $K_{3,3}$; 2) $K_{3,4}$; 3) $P_3 \times P_3$; 4) $P_3 \times P_4$; 5) $P_4 \times P_4$.

5.9. При каких *p* и *q* в графе $K_{p,q}$ имеется а) гамильтонов цикл? Б) гамильтонов путь?

5.10. При каких n существует гамильтонов цикл в графе а) $P_3 \times P_n$; б) $P_4 \times P_n$?

5.11. Постройте дерево путей для графа, изображенного на рисунке а) с корнем в вершине 1; б) с корнем в вершине 2.



5.12. Сколько листьев будет в дереве путей для графа $K_{4,4}$?

5.13. Найдите гамильтонов путь в графе Q_4.

5.14. Найдите гамильтонов путь в турнире, заданном матрицей смежности:

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

5.15. Сколько в полном графе K_n имеется подграфов, изоморфных а) C_3; б) C_4; в) C_k, $k \le n$?

5.16. Операция сложения графов по модулю 2 может быть распространена на графы с разными множествами вершин следующим образом: если $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$, то $G_1 \oplus G_2 = (V_1 \cup V_2, E_1 \otimes E_2)$. Можно ли с помощью этой операции, а также удаления и добавления изолированных вершин и переименования вершин получить а) C_4 из C_3; б) C_5 из C_3; в) C_3 из C_4; г) C_3 из C_5?

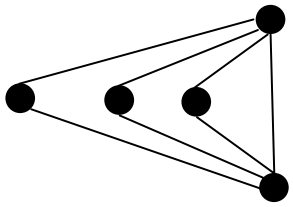
5.17. Связный граф имеет 10 вершин, 4 из них имеют степень 6, остальные – степень 3. Чему равно цикломатическое число этого графа?

5.18. Постройте фундаментальные циклы для графа $K_3 \times K_3$ относительно каркаса, полученного с помощью а) поиска в глубину; б) поиска в ширину. Выразите границу внешней грани как сумму базисных циклов.

5.19. Какова будет суммарная длина фундаментальных циклов, построенных с помощью поиска в ширину для графа а) K_n; б) $K_{n,n}$?

5.20. Сколько различных разрезов имеется а) у связного графа с n вершинами;
б) у графа с k компонентами и n вершинами?

5.21. Найдите базис циклов по матрице инцидентности для графа, показанного на рисунке.



5.22*. Модифицируйте алгоритм построения эйлерова цикла так, чтобы не требовалась предварительная проверка четности степеней. Существование вершины нечетной степени должно обнаруживаться по ходу работы алгоритма.

5.23*. Докажите, что при любом $n \geq 2$ в графе Q_n существует гамильтонов цикл.

5.24*. Разработайте алгоритм построения дерева путей и поиска гамильтонова цикла.

5.25*. Докажите, что в любом сильно связном турнире имеется гамильтонов цикл.

5.26.* Разработайте алгоритм построения базиса циклов на основе поиска в ширину.

6. Независимые множества, клики, вершинные покрытия

Рассмотрим три задачи, тесно связанные между собой.

Независимым множеством вершин графа называется любое множество попарно не смежных вершин, т.е. множество вершин, порождающее пустой подграф. Независимое множество называется *наибольшим*, если оно содержит наибольшее количество вершин. Число вершин в наибольшем независимом множестве графа G обозначается через $\alpha(G)$ и называется *числом независимости* графа. Задача о независимом множестве состоит в нахождении наибольшего независимого множества.

Кликой графа называется множество вершин, порождающее полный подграф, т.е. множество вершин, каждые две из которых смежны. Число вершин в клике наибольшего размера называется *кликовым числом* графа и обозначается через $\omega(G)$. Очевидно, задача о независимом множестве преобразуется в задачу о клике и наоборот простым переходом от данного графа G к дополнительному графу $\overline{G}$, так что $\alpha(G) = \omega(\overline{G})$.

Вершинное покрытие графа – это такое множество вершин, что каждое ребро графа инцидентно хотя бы одной из этих вершин. Наименьшее число вершин в вершинном покрытии графа G обозначается через $\beta(G)$ и называется *числом вершинного покрытия* графа.

В графе на рисунке 9 наибольшим независимым множеством является множество {1,3,4,7}, наибольшей кликой – множество {2,3,5,6}, наименьшим вершинным покрытием – множество {2,5,6}.

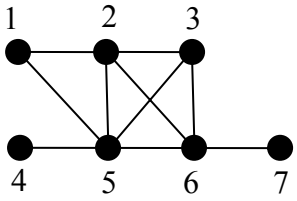


Рис. 9.

Между задачами о независимом множестве и о вершинном покрытии имеется простая связь.

Теорема о независимом множестве и вершинном покрытии. *Пусть G – граф с множеством вершин V. Множество* $U \subseteq V$ *является вершинным покрытием графа G тогда и только тогда, когда* $\overline{U} = V - U$ *– независимое множество.*

Из этой теоремы следует, что $\alpha(G) + \beta(G) = n$ для любого графа *G* с *n* вершинами.

Итак, все три задачи тесно связаны друг с другом и алгоритм для решения одной из них легко преобразуется в алгоритм решения другой. В то же время эти задачи относятся к категории неподдающихся, так что все известные алгоритмы получения их точного решения имеют переборный характер и требуют во многих случаях много времени даже для не очень больших графов. Для того, чтобы можно было решить задачу за реальное время, приходится ослаблять требования, например, искать не точное, а достаточно хорошее или приемлемое для практики решение (например, не обязательно наибольшее, но достаточно большое независимое множество). Алгоритмы, реализующие такой подход, основываются на разнообразных интуитивных идеях о том, как найти хорошее решение. Такие идеи называют *эвристиками*, а основанные на них алгоритмы – *эвристическими алгоритмами*. Иногда удается оценить точность приближения, тогда говорят о приближенном алгоритме. Рассмотрим некоторые точные, эвристические и приближенные алгоритмы для задач о независимом множестве и вершинном покрытии.

Пусть *G* – граф, в котором требуется найти наибольшее независимое множество. Выберем в нем произвольную вершину *a*, не являющуюся изолированной и рассмотрим две возможности – эта вершина принадлежит искомому множеству *X* или не принадлежит ему. Если не принадлежит, то *X* является независимым множеством графа, получаемого удалением вершины *a* из графа *G*. Обозначим этот граф через G_1. Если же *a* принадлежит *X*, то никакая вершина, смежная с *a*, не принадлежит *X*. В этом случае множество *X* является независимым множеством графа, получающегося удалением из *G* всех вершин, смежных с *a*. Обозначим этот граф через G_2. В графе G_1 вершин меньше, чем в графе *G*, а так как вершина *a* не изолированная, то в графе G_2 их тоже меньше. Таким образом, задача о независимом множестве для графа *G* свелась к решению той же задачи для двух графов меньшего размера. Это приводит к рекуррентному соотношению для числа независимости:

$$\alpha(G) = \max\{\alpha(G_1), \alpha(G_2)\}$$

и к рекурсивному алгоритму для нахождения наибольшего независимого множества графа *G*: найдем наибольшее независимое множество X_1 графа G_1, затем наибольшее независимое множество X_2 графа G_2 и выберем большее из этих двух множеств. Процесс решения можно изобразить в виде бинарного дерева (его называют *деревом подзадач, деревом вариантов, деревом решений*). Корнем дерева служит граф *G*, его двумя

сыновьями – графы G_1 и G_2. Чтобы не путать вершины дерева и вершины графа, вершины дерева будем называть узлами. Узел, не являющийся листом, называется внутренним узлом. Каждому внутреннему узлу дерева соответствует некоторый граф H и некоторая вершина этого графа x. Вершину x можно выбирать произвольно, но она не должна быть изолированной вершиной графа H. Внутренний узел имеет двух сыновей – левого и правого. Левому сыну соответствует подграф графа H, получаемый удалением вершины x, а правому – подграф, получаемый удалением всех вершин, смежных с x. Листьям соответствуют подграфы, не имеющие ребер, то есть подграфы, у которых все вершины изолированные. Множества вершин этих подграфов – это независимые множества исходного графа.

Дерево решений может быть очень большим. Например, для графа mK_2 в этом дереве будет 2^m листьев. Для нахождения наибольшего независимого множества не обязательно строить все дерево полностью, а достаточно обойти его в том или ином порядке, запоминая на каждом шаге только небольшую часть информации об устройстве этого дерева. Можно, например, применить поиск в глубину для обхода дерева: сначала пройти от корня до некоторого листа, затем вернуться к предку этого листа и искать следующий лист, и т.д.

Рассмотрим две простые эвристики для задачи о независимом множестве.

Одна из эвристических идей состоит в том, чтобы рассмотреть только один путь от корня до листа в дереве решений в надежде, что этому листу соответствует достаточно большое независимое множество. Для выбора этого единственного пути могут применяться разные соображения. В дереве решений, описанном выше, у каждого внутреннего узла имеются два сына. Одному из них соответствует подграф, получающийся удалением некоторой произвольно выбранной вершины a, а другому – подграф, получающийся удалением окрестности этой вершины. Чтобы вместо дерева получился один путь, достаточно каждый раз выполнять какую-нибудь одну из этих двух операций. Рассмотрим оба варианта.

Допустим, мы решили каждый раз удалять выбранную вершину. Эти удаления производятся до тех пор, пока не останется граф без ребер, т.е. независимое множество. Оно и принимается в качестве решения задачи. Для полного описания алгоритма необходимо еще сформулировать правило выбора активной вершины a. Мы хотим получить граф без ребер, в котором было бы как можно больше вершин. Чем меньше вершин будет удалено, тем больше их останется. Значит, цель – как можно быстрее удалить все ребра. Кажется, мы будем двигаться в нужном направлении, если на каждом шаге будем удалять наибольшее возможное на этом шаге число ребер. Это означает, что в качестве активной вершины всегда нужно выбирать вершину наибольшей степени.

Другой вариант – каждый раз удалять окрестность активной вершины a. Это повторяется до тех пор, пока оставшиеся вершины не будут образовывать независимого множества. Удаление окрестности вершины a равносильно тому, что сама эта вершина включается в независимое множество, которое будет получено в качестве ответа. Так как мы хотим получить в итоге как можно большее независимое множество, следует стараться удалять на каждом шаге как можно меньше вершин. Это означает, что в качестве активной вершины всегда нужно выбирать вершину наименьшей степени.

Имеется немало графов, для которых каждая из этих эвристик дает близкое к оптимальному, а иногда и оптимальное решение. Но, как это обычно бывает с эвристическими алгоритмами, можно найти примеры графов, для которых найденные решения будут весьма далеки от оптимальных. Рассмотрим граф G_t, у которого множество вершин V состоит из трех частей: $V = A \cup B_1 \cup B_2$, причем $|A| = |B_1| = |B_2| = t$, A является независимым множеством, каждое из множеств B_1, B_2 – кликой, и каждая

вершина из множества A смежна с каждой вершиной из множества $B_1 \cup B_2$. Этот граф можно представить формулой $G_t = (2K_t) \circ O_t$. Степень каждой вершины из множества A в этом графе равна $2t$, а степень каждой вершины из множества $B_1 \cup B_2$ равна $2t-1$. Первый алгоритм, выбирающий вершину наибольшей степени, будет удалять вершины из множества A до тех пор, пока не удалит их все. После этого останется граф, состоящий из двух клик, и в конечном итоге будет получено независимое множество из двух вершин. Второй алгоритм на первом шаге возьмет в качестве активной одну из вершин множества $B_1 \cup B_2$ и удалит всю ее окрестность. В результате получится граф, состоящий из этой вершины и клики, а после второго шага получится независимое множество, состоящее опять из двух вершин. Итак, при применении к этому графу любой из двух эвристик получается независимое множество из двух вершин. В то же время в графе имеется независимое множество A мощности t.

Рассмотрим другой способ организации перебора для получения точного решения Он использует алгебраические преобразования. Для его описания удобнее рассмотреть задачу о вершинном покрытии. Пусть $(x_1, x_2, \ldots, x_n)$ – характеристический вектор искомого множества *X*. Это множество должно быть вершинным покрытием, т.е. хотя бы одна из двух вершин каждого ребра должна принадлежать множеству *X*. Это означает, что для каждого ребра (i, j) дизъюнкция $x_i \vee x_j$ должна быть равна 1. Значит, должно выполняться равенство

$$\prod_{(i,j)\in E} (x_i \vee x_j) = 1,$$

где *E* – множество ребер графа, а П обозначает конъюнкцию. Формула в левой части равенства – конъюнктивная нормальная форма. Если раскрыть скобки то получится дизъюнктивная нормальная форма, каждое слагаемое которой представляет некоторое вершинное покрытие графа – слагаемое $x_{i_1} x_{i_2} \ldots x_{i_k}$ представляет вершинное покрытие, состоящее из вершин с номерами $i_1, i_2, \ldots, i_k$. Остается выбрать слагаемое с наименьшим числом сомножителей. К сожалению, при раскрытии скобок могут получаться очень длинные формулы, даже при применении известных упрощающих преобразований вроде закона поглощения.

Рассмотрим еще один простой приближенный алгоритм для задачи о вершинном покрытии.

Работа алгоритма начинается с создания пустого множества *X* и состоит в выполнении однотипных шагов, в результате каждого из которых к множеству *X* добавляются новые вершины. Допустим, перед очередным шагом имеется некоторое множество вершин *X*. Если оно покрывает все ребра (т.е. каждое ребро инцидентно одной из этих вершин), то процесс заканчивается и множество *X* принимается в качестве искомого вершинного покрытия. В противном случае выбирается какое-нибудь непокрытое ребро (a, b), и вершины *a* и *b* добавляются к множеству *X*.

Для полного описания алгоритма нужно бы еще сформулировать правило выбора ребра (a, b). Однако для оценки степени приближения, которая будет сейчас получена, это не имеет значения. Можно считать, что какое-то правило выбрано.

Обозначим через $\beta'(G)$ мощность вершинного покрытия, которое получится при применении этого алгоритма к графу *G*, и докажем, что $\beta'(G) \le 2\beta(G)$. Иначе говоря, полученное с помощью этого алгоритма решение не более чем в два раза отличается от оптимального.

Действительно, допустим, что до окончания работы алгоритм выполняет k шагов, добавляя к множеству X вершины ребер $(a_1, b_1), \ldots, (a_k, b_k)$. Тогда $\beta'(G) = 2k$. Никакие два из этих k ребер не имеют общей вершины. Значит, чтобы покрыть все эти ребра, нужно не меньше k вершин. Следовательно, $\beta(G) \geq k$ и $\beta'(G) \leq 2\beta(G)$.

Задачи

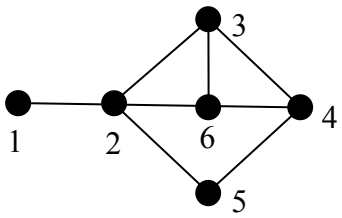
6.1. Сколько имеется абстрактных графов с $\alpha(G) = 3$, имеющих гамильтонов цикл
а) с 5 вершинами; б) с 6 вершинами?

6.2. Найдите 1) $\alpha((C_7 \circ P_7) + Q_3)$; 2) $\alpha(C_3 \times C_5)$; 3) $\beta(P_3 \times P_4)$.

6.3. Найдите $\alpha(Q_n)$, $\beta(Q_n)$, $\omega(Q_n)$.

6.4. Сколько наибольших независимых множеств имеется у графа а) C_8; б) C_9; в) P_8; г) P_9?

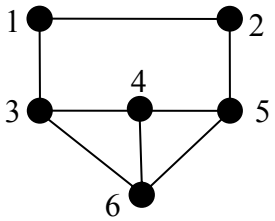
6.5. Найдите наибольшее независимое множество с помощью дерева решений в графе, показанном на рисунке.



6.7. Сколько листьев будет в дереве вариантов для независимых множеств, построенном для графа $3K_p$?

6.8. Рассмотрим две эвристики для задачи о независимом множестве:
1) выбирать вершину наибольшей степени и удалять ее, пока не останется независимое множество;
2) выбирать вершину наименьшей степени и удалять ее окрестность, пока не останется независимое множество.
Какая из них всегда дает правильный ответ а) для графа $K_{p,q}$; б) для графа P_n; в) для графа $(K_1 + K_2) \circ O_3$?

6.9. Найдите наименьшее вершинное покрытие алгебраическим методом в графе, показанном на рисунке. Примените к нему также приближенный алгоритм.



6.10. Даны графы G_1, G_2 и G_3, с 9 вершинами каждый. Известно, что $\alpha(G_1) = 2$,

$\alpha(G_2) = 3$, $\alpha(G_3) = 4$. Найдите

1) $\alpha(G_1 + G_2 + G_3)$;
2) $\alpha(G_1 \circ G_2 \circ G_3)$;
3) $\alpha((G_1 + G_2) \circ G_3)$;
4) $\beta(G_1 + G_2 + G_3)$;
5) $\beta((G_1 \circ G_2) + G_3)$.

6.11. Какие из следующих равенств выполняются для любых графов G_1 и G_2:

1) $\alpha(G_1 \times G_2) = \max(\alpha(G_1), \alpha(G_2))$;
2) $\varpi(G_1 \times G_2) = \max(\varpi(G_1), \varpi(G_2))$;
3) $\beta(G_1 \times G_2) = \max(\beta(G_1), \beta(G_2))$?

6.12*. Докажите, что а) для любого графа G с 6 вершинами $\alpha(G) \geq 3$ или $\omega(G) \geq 3$;
б) для любого графа G с 10 вершинами $\alpha(G) \geq 4$ или $\omega(G) \geq 3$.

6.13*. Докажите, что вторая эвристика из задачи 6.8, если ее применить к дереву, всегда дает наибольшее независимое множество. Верно ли это для первой эвристики из той же задачи?

6.14*. На прямой задано конечное множество интервалов, требуется найти наибольшее подмножество попарно не пересекающихся интервалов. Покажите, что задача сводится к нахождению наибольшего независимого множества в графе. Разработайте алгоритм, решающий эту задачу за время $O(n)$, где n – число интервалов.

7. Паросочетания

Паросочетания и реберные покрытия

Паросочетанием в графе называется множество ребер, попарно не имеющих общих вершин. Задача о паросочетании состоит в том, чтобы в данном графе найти паросочетание с наибольшим числом ребер. Это число для графа G будем обозначать через $\pi(G)$. *Реберным покрытием* графа называется такое множество ребер, что всякая вершина графа инцидентна хотя бы одному из этих ребер. Наименьшее число ребер в реберном покрытии графа G обозначим через $\rho(G)$. Заметим, что реберное покрытие существует только для графов без изолированных вершин.

Между задачами о паросочетании и о реберном покрытии имеется тесная связь, подобно связи между задачами о независимом множестве (паросочетание называют иногда независимым множеством ребер) и о вершинном покрытии. Количественно эта связь выражается следующим образом.

Теорема о паросочетаниях и реберных покрытиях. *Для любого графа G с n вершинами, не имеющего изолированных вершин, справедливо равенство $\pi(G) + \rho(G) = n$.*

Из решения одной задачи легко получить решение другой. Пусть M – наибольшее паросочетание в графе G, не имеющем изолированных вершин. Для каждой вершины, не принадлежащей никакому ребру паросочетания, выберем какое-нибудь ребро,

инцидентное этой вершине и добавим все выбранные ребра к множеству M. Полученное множество ребер будет наименьшим реберным покрытием. Обратно, пусть C – наименьшее реберное покрытие графа G. Тогда в подграфе, образованном ребрами этого покрытия, каждая компонента связности является звездой (звезда – полный двудольный граф, у которого одна доля состоит из одной вершины). Выбрав по одному ребру из каждой компоненты, получим наибольшее паросочетание.

Несмотря на такое сходство между "вершинными" и "реберными" вариантами независимых множеств и покрытий, имеется кардинальное различие в сложности соответствующих экстремальных задач. "Вершинные" задачи относятся кразряду неподдающихся, для реберных же известны полиномиальные алгоритмы.

Метод увеличивающих путей

Пусть G – граф, M – некоторое паросочетание в нем. Ребра паросочетания будем называть *сильными,* остальные ребра графа – *слабыми*. Вершину назовем *свободной,* если она не принадлежит ребру паросочетания. *Чередующимся путем* относительно данного паросочетания называется простой путь, в котором чередуются сильные и слабые ребра (т.е. за сильным ребром следует слабое, за слабым – сильное). Чередующийся путь называется *увеличивающим путем*, если он соединяет две свободные вершины. Если имеется увеличивающий путь относительно данного паросочетания, то можно построить большее паросочетание – нужно вдоль этого пути превратить слабые ребра в сильные, а сильные в слабые. Эту операцию назовем *инверсией*. В результате инверсии мощность паросочетания увеличивается на 1.

Теорема об увеличивающем пути. *Паросочетание является наибольшим тогда и только тогда, когда относительно него нет увеличивающих путей.*

Для решения задачи о паросочетании остается научиться находить увеличивающие пути или убеждаться, что таких путей нет. Это можно сделать, перебирая чередующиеся пути (подобно перебору путей при поиске гамильтонова цикла). Дело осложняется тем, что чередующихся путей может быть много (см. задачу 7.4). Оказывается, нет необходимости перебирать их все. Известны эффективные алгоритмы, которые ищут увеличивающие пути для произвольных графов. Рассмотрим простой алгоритм, решающий эту задачу для двудольных графов.

Паросочетания в двудольных графах

Пусть G – двудольный граф с долями A и B, M – паросочетание в G. Всякий увеличивающий путь, если такой имеется, соединяет вершину из множества A с вершиной из B. Поэтому при поиске увеличивающего пути достаточно рассматривать чередующиеся пути, начинающиеся в свободных вершинах из доли A.

Зафиксируем некоторую свободную вершину $a \in A$. Дерево T с корнем в вершине a назовем *деревом достижимости* для вершины a, если

1) T является подграфом графа G;

2) каждый путь в дереве T, начинающийся в корне, является чередующимся путем;

3) T – максимальное дерево со свойствами 1) и 2).

Дерево достижимости определено не однозначно, но любое такое дерево в двудольном графе обладает следующим свойством.

Теорема о дереве достижимости. *Пусть G – двудольный граф с заданным паросочетанием, T – дерево достижимости для вершины a. Вершина x принадлежит дереву T тогда и только тогда, когда в графе существует чередующийся путь, соединяющий вершины a и x.*

Для построения дерева достижимости можно использовать слегка модифицированный поиск в ширину из вершины a. Отличие от стандартного поиска в ширину состоит в том, что для вершин из доли A (они находятся на четном расстоянии от вершины a) исследуются инцидентные им слабые ребра, а для вершин из доли B – сильные. Если в дереве появляется отличная от a свободная вершина b, это означает, что найден увеличивающий путь – это путь между a и b в дереве. Выполнив инверсию, получим большее паросочетание. После этого, если в доле A еще имеются свободные вершины, возобновляется поиск увеличивающего пути (т.е. строится новое дерево достижимости).

Если дерево построено и в нем нет других свободных вершин, кроме корня, то нужно выбрать другую свободную вершину в доле A (если такие еще есть) и строить дерево достижимости для нее. Если деревья достижимости для всех свободных вершин из A построены, а увеличивающий путь не найден, то имеющееся паросочетание является наибольшим.

Допустим, дерево достижимости для вершины a построено и выяснилось, что нет увеличивающих путей, начинающихся в a. Затем было построено дерево достижимости с другим корнем, найден увеличивающий путь и построено увеличенное паросочетание. Нужно ли строить дерево достижимости для вершины a относительно нового паросочетания? Оказывается, нет. Назовем свободную вершину *бесполезной* для данного паросочетания, если нет увеличивающих путей, начинающихся в этой вершине. Пусть M' – паросочетание, полученное из паросочетания M инверсией относительно некоторого увеличивающего пути.

Утверждение. Если вершина a бесполезна для паросочетания M, то она бесполезна и для паросочетания M'.

Таким образом, каждая вершина может выступать в качестве корня дерева достижимости не более одного раза, т.е. всего будет построено не более n деревьев. Построение одного дерева выполняется за время $O(m)$ и общее время работы алгоритма оценивается как $O(mn)$.

Независимые множества в двудольных графах

Описанный метод построения наибольшего паросочетания называют методом чередующихся путей. Он может применяться и для решения других задач. Рассмотрим задачу о независимом множестве. Она трудна в общем случае, но для двудольных графов допускает достаточно быстрое решение с помощью метода чередующихся путей.

Пусть G – двудольный граф с долями A и B и требуется найти в нем наибольшее независимое множество. Найдем сначала наибольшее паросочетание M в этом графе. Пусть A_0 и B_0 соответственно множества свободных вершин в долях A и B. Найдем все вершины из A, достижимые чередующимися путями из A_0, пусть A_1 – множество всех таких вершин. Аналогично, пусть B_1 – множество все вершин из B, достижимых чередующимися путями из B_0. Обозначим через A_2 множество всех вершин из A, не

достижимых чередующимися путями ни из A_0, ни из B_0 (см. рисунок 10). Вершины из $A_0 \cup A_1$ не смежны с вершинами из $B_0 \cup B_1$, так как иначе образовался бы увеличивающий путь. Вершины из A_2 тоже не смежны с вершинами из $B_0 \cup B_1$, иначе они были бы достижимы из B_0. Значит, $A_0 \cup A_1 \cup A_2 \cup B_0 \cup B_1$ – независимое множество. Оно содержит все свободные вершины и по одной вершине из каждого ребра паросочетания. Отсюда следует, что это независимое множество – наибольшее. Паросочетание *M* состоит из $\pi(G)$ ребер, а свободных вершин имеется ровно $n - 2\pi(G)$. Следовательно, $\alpha(G) = n - \pi(G)$. Так как $\alpha(G) + \beta(G) = n$, то справедливо следующее утверждение.

Теорема о числе вершинного покрытия двудольного графа. *Для любого двудольного графа G выполняется равенство* $\beta(G) = \pi(G)$.

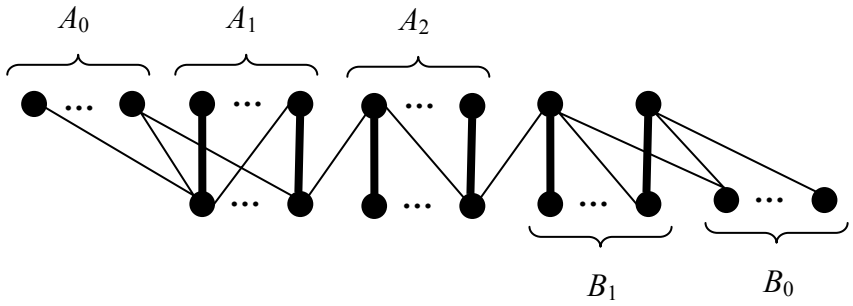


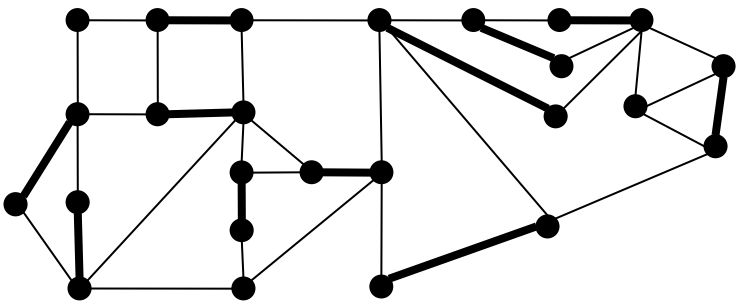
Рис. 10.

Задачи

7.1. Найдите наибольшее паросочетание и наименьшее реберное покрытие для графов P_6, P_7, $K_{3,7}$.

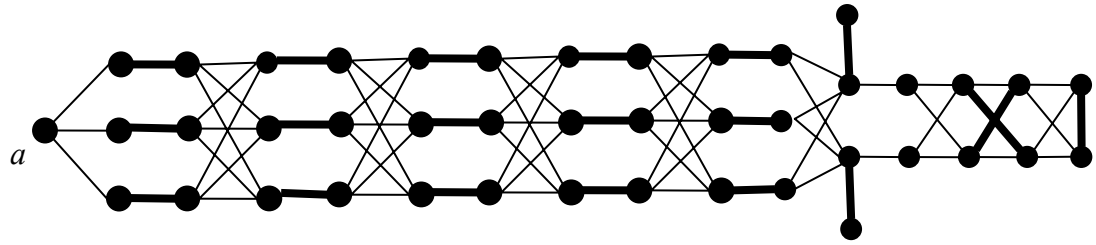
7.2. Сколько наибольших паросочетаний имеется в графе 1) P_5; 2) P_6; 3) C_5; 4) C_6; 5) K_4; 6) K_5?

7.3. Является ли показанное на рисунке паросочетание наибольшим?

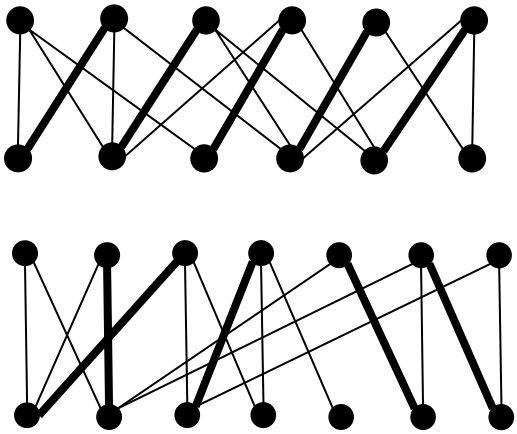


7.4. Сколько имеется максимальных (не имеющих продолжения) чередующихся путей относительно показанного на рисунке паросочетания, начинающихся в вершине *a*? Есть ли среди них увеличивающий путь? Есть ли вообще для данного паросочетания

увеличивающий путь?



7.5. Проверьте, являются ли данные паросочетания наибольшими. Найдите в этих графах наименьшие реберные покрытия и наибольшие независимые множества.



7.6*. Докажите утверждение о бесполезных вершинах.

8. Раскраски

Раскраска вершин

Раскраской вершин графа называется назначение цветов его вершинам. Раскраска называется *правильной*, если любые две смежные вершины имеют разные цвета. Задача о раскраске состоит в нахождении правильной раскраски данного графа G в наименьшее число цветов. Это число называется *хроматическим числом* графа и обозначается $\chi(G)$.

В правильной раскраске полного графа K_n все вершины должны иметь разные цвета, поэтому $\chi(K_n) = n$. Если в каком-нибудь графе имеется полный подграф с k вершинами, то для раскраски этого подграфа необходимо k цветов. Отсюда следует, что для любого графа выполняется неравенство

$$\chi(G) \geq \omega(G).$$

Однако хроматическое число может быть и строго больше кликового числа. Например, $\omega(C_5) = 2$, а $\chi(C_5) = 3$.

Раскраску можно также рассматривать как разбиение множества вершин на независимые множества. Каждая из частей разбиения представляет собой множество вершин одного цвета, эти множества называют цветными классами. Каждый цветной класс содержит не более чем $\alpha(G)$ вершин. Поэтому произведение числа классов (т.е. цветов) на $\alpha(G)$ не превосходит числа всех вершин графа. Отсюда следует неравенство

$$\chi(G) \geq \frac{n}{\alpha(G)}.$$

Тот же граф C_5 служит примером, когда это неравенство строгое.

Очевидно, $\chi(G) = 1$ тогда и только тогда, когда G – пустой граф. Нетрудно охарактеризовать и графы с хроматическим числом 2 (точнее, не больше 2). По определению, это такие графы, у которых множество вершин можно разбить на два независимых множества. Но это совпадает с определением двудольного графа. Поэтому двудольные графы называют еще *бихроматическими*. Согласно теореме Кёнига, граф является бихроматическим тогда и только тогда, когда в нем нет циклов нечетной длины.

Для графов с хроматическим числом 3 такого простого описания не найдено. Не известны и простые алгоритмы, проверяющие, можно ли данный граф раскрасить в 3 цвета. Задача такой проверки (вообще, задача проверки возможности раскрасить граф в k цветов при любом фиксированном $k \geq 3$) относится к неподдающимся.

Рассмотрим алгоритм решения задачи о раскраске, похожий на описанный выше алгоритм для задачи о независимом множестве. Сходство заключается в том, что задача для данного графа сводится к той же задаче для двух других графов. Поэтому снова возникает дерево решений, обход которого позволяет найти решение. Но есть и одно существенное различие, состоящее в том, что теперь два новых графа не будут подграфами исходного графа.

Выберем в данном графе G две несмежные вершины x и y и построим два новых графа: G_1, получающийся добавлением ребра (x, y) к графу G, и G_2, получающийся из G слиянием вершин x и y. Операция слияния состоит в удалении вершин x и y и добавлении новой вершины z и ребер, соединяющих ее с каждой вершиной, с которой была смежна хотя бы одна из вершин x, y.

Если в правильной раскраске графа G вершины x и y имеют разные цвета, то она будет правильной и для графа G_1. Если же цвета вершин x и y в раскраске графа G одинаковы, то граф G_2 можно раскрасить в то же число цветов: новая вершина z окрашивается в тот цвет, в который окрашены вершины x и y, а все остальные вершины сохраняют те цвета, которые они имели в графе G. И наоборот, раскраска каждого из графов G_1, G_2, очевидно, дает раскраску графа G в то же число цветов. Поэтому

$$\chi(G) = \min\{\chi(G_1), \chi(G_2)\},$$

что дает возможность рекурсивного нахождения раскраски графа в минимальное число цветов. Заметим, что граф G_1 имеет столько же вершин, сколько исходный граф, но у него больше ребер. Поэтому рекурсия в конечном счете приводит к полным графам, для которых задача о раскраске решается тривиально.

Для задачи о раскраске вершин придумано немало эвристик. Одна из простейших называется *последовательной раскраской*. Вершины графа как-нибудь линейно упорядочиваются и раскрашиваются в этом порядке в цвета, которыми являются натуральные числа. Очередная вершина красится в наименьший цвет, который еще не использован ни для одной из смежных с ней вершин. Интересно, что для любого графа существует упорядочение, при котором последовательная раскраска дает оптимальный результат.

Обозначим через $\Delta(G)$ наибольшую степень вершины в графе G. При окрашивании очередной вершины в алгоритме последовательной раскраски для смежных с ней вершин использовано не более $\Delta(G)$ цветов, значит, хотя бы один из цветов $1,2,\ldots,\Delta(G)+1$ свободен и может быть применен. Отсюда следует еще одно неравенство для хроматического числа:

$$\chi(G) \le \Delta(G) + 1,$$

причем алгоритм последовательной раскраски в любом случае использует не более $\Delta(G) + 1$ цветов.

Раскраска ребер

Наряду с задачей о раскраске вершин известна задача о *раскраске ребер* графа, когда цвета назначаются ребрам. Раскраска ребер называется правильной, если любые два ребра, имеющие общую вершину, окрашены в разные цвета. Минимальное число цветов, необходимое для правильной раскраски ребер графа G, называется *хроматическим индексом* графа и обозначается через $\chi'(G)$.

В правильной раскраске ребер множество ребер одного цвета образует паросочетание. Поэтому такую раскраску можно трактовать как разбиение множества ребер графа на паросочетания.

Обозначим через $\Delta(G)$ максимальную степень вершины в графе. При правильной реберной раскраске все ребра, инцидентные одной вершине, должны иметь разные цвета. Отсюда следует, что для любого графа выполняется неравенство $\chi'(G) \ge \Delta(G)$. Для некоторых графов имеет место строгое неравенство, например, $\Delta(C_3) = 2$, а $\chi'(C_3) = 3$. Следующая теорема показывает, что $\chi'(G)$ может отличаться от $\Delta(G)$ не более чем на 1.

Теорема Визинга [4]. *Для любого графа G справедливо неравенство* $\chi'(G) \le \Delta(G) + 1$.

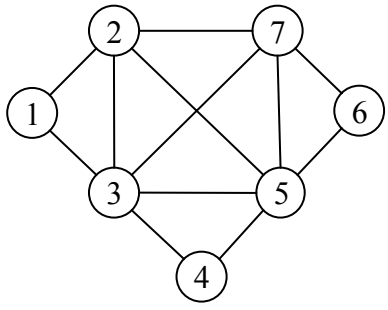
Задачи

8.1. Найдите хроматическое число графов $K_{p,q}$, P_n, C_n, Q_n.

8.2. Найдите хроматическое число графа 1) $C_4 + C_7$; 2) $C_4 \circ C_7$; 3) $\overline{C_7}$.

8.3. Примените алгоритм раскрашивания, основанный на дереве решений, к графу из задачи 6.9.

8.4. Примените алгоритм последовательной раскраски к графу, изображенному на рисунке, если вершины упорядочиваются а) по возрастанию номеров; б) по убыванию номеров.



8.5. Для каких из следующих графов алгоритм последовательной раскраски дает точный результат при любом упорядочении вершин: P_3, P_4, C_4, C_6, $K_{p,q}$?

8.6. Верно ли, что алгоритм последовательной раскраски, примененный к двудольному графу, всегда дает оптимальную раскраску?

8.7. Найдите хроматический индекс графов C_6, C_7, $K_{3,3}$, $K_{4,4}$, K_4, K_5.

9. Оптимальные каркасы и пути

В задаче об оптимальном каркасе (она известна также как задача о кратчайшем остовном дереве) задан граф $G=(V,E)$, каждому ребру которого приписано число, называемое весом ребра. Вес ребра (x,y) будем обозначать через $w(x,y)$. Вес множества $X \subseteq E$ определяется как сумма весов составляющих его ребер. Требуется в графе *G* найти каркас минимального веса. Для решения этой задачи известно несколько алгоритмов. Рассмотрим два из них.

Алгоритм Прима

Будем предполагать, что граф *G* связен, так что решением задачи всегда будет дерево. В алгоритме Прима (R. Prim) на каждом шаге рассматривается частичное решение задачи, представляющее собой дерево. Вначале это дерево состоит из единственной вершины, в качестве которой может быть выбрана любая вершина графа. Затем к дереву последовательно добавляются ребра и вершины, пока не получится остовное дерево, т.е. каркас. Для того чтобы из текущего дерева при добавлении нового ребра опять получилось дерево, это новое ребро должно соединять вершину дерева с вершиной, еще не принадлежащей дереву. Такие ребра будем называть *подходящими* относительно рассматриваемого дерева. В алгоритме Прима применяется следующее правило выбора: на каждом шаге из всех подходящих ребер выбирается ребро наименьшего веса. Это ребро вместе с одной новой вершиной добавляется к дереву. Если обозначить через U и F множества вершин и ребер строящегося дерева, то алгоритм Прима можно представить следующим образом. Через $\overline{U}$ обозначаем дополнение множества *U* до множества *V*.

Алгоритм Прима.

1 $U := \{a\}$, где a – произвольная вершина графа;
2 $F := \varnothing$;
3 **while** $U \neq V$ **do**
4 найти ребро наименьшего веса (x,y) среди всех ребер, у которых $x \in U$, $y \in \overline{U}$;
5 добавить к *F* ребро (x,y), а к *U* вершину *y*.

Теорема об алгоритме Прима. *Подграф* (U,F), *построенный с помощью алгоритма Прима, является каркасом наименьшего веса для данного графа.*

Оценим время работы алгоритма Прима. Цикл **while** в строке 3 повторяется $n-1$ раз. Внутри этого цикла есть еще скрытый цикл в строке 4, где ищется ребро наименьшего веса среди подходящих ребер. Допустим, что этот поиск производится самым бесхитростным образом, т.е. просматриваются все пары вершин (x,y) с $x \in U$, $y \in \overline{U}$. Если $|U| = k$, то имеется $k(n-k)$ таких пар. Всего получаем

$$\sum_{k=1}^{n-1} k(n-k) = n\sum_{k=1}^{n-1} k - \sum_{k=1}^{n-1} k^2 = \frac{n^2(n-1)}{2} - \frac{(n-1)n(2n-1)}{6} = \frac{n^3-n}{6}$$

пар, которые нужно рассмотреть. Таким образом, трудоемкость алгоритма будет $O(n^3)$.

Небольшое усовершенствование позволяет на порядок ускорить этот алгоритм. Допустим, что для каждой вершины y из множества $\overline{U}$ известна такая вершина $f(y) \in U$, что ребро $(f(y), y)$ имеет наименьший вес среди всех ребер вида (x, y), где $x \in U$. Тогда при $|U| = k$ для поиска подходящего ребра наименьшего веса достаточно будет сравнить $n-k$ ребер вида $(f(y), y)$, а общее число анализируемых ребер будет равно

$$\sum_{k=1}^{n-1} (n-k) = \frac{n(n-1)}{2}.$$

В этом случае, однако, необходимы дополнительные действия для обновления таблицы значений функции f при добавлении новой вершины к дереву, т.е. при переносе одной вершины из множества $\overline{U}$ в множество U. Сначала, когда множество U состоит из единственной вершины a, полагаем $f(x) = a$ для всех $x \in \overline{U}$. В дальнейшем эти значения могут меняться. Допустим, на некотором шаге к дереву присоединяется вершина y. Тогда для каждой вершины $z \in \overline{U}$ либо сохраняется старое значение $f(z)$, либо устанавливается новое $f(z) = y$, в зависимости от того, какое из ребер $(f(z), z)$ и (y, z) имеет меньший вес. По окончании работы алгоритма массив f будет массивом отцов построенного дерева относительно корня a. Поэтому отпадает необходимость отдельно запоминать добавляемые к дереву ребра. Алгоритм теперь можно записать следующим образом.

Алгоритм Прима усовершенствованный.

1 $U := \{a\}$, где a – произвольная вершина графа;
2 **for** $y \in \overline{U}$ **do** $f(y) := a$ **;**
3 **while** $U \neq V$ **do**
4 найти вершину $y \in \overline{U}$ с наименьшим значением $w(f(y), y)$;
5 добавить к U вершину y;
6 **for** $z \in \overline{U}$ **do** **if** $w(f(z), z) > w(y, z)$ **then** $f(z) := y$.

При $|U| = k$ цикл в строке 6 повторяется $n-k$ раз. Таким образом, дополнительное время, необходимое для обслуживания массива f, тоже оценивается сверху квадратичной функцией от n и общая оценка трудоемкости усовершенствованного алгоритма Прима будет $O(n^2)$.

Алгоритм Краскала

Другой алгоритм для задачи об оптимальном каркасе известен как алгоритм Краскала (J. Kruskal). В нем тоже на каждом шаге рассматривается частичное решение. Отличие от алгоритма Прима состоит в том, что в алгоритме Краскала частичное решение всегда представляет собой остовный лес F графа G, т.е. лес, состоящий из всех вершин графа G и некоторых его ребер. Вначале F не содержит ни одного ребра, т.е. состоит из

изолированных вершин. Затем к нему последовательно добавляются ребра, пока не будет построен каркас графа *G*. Пусть *F* – лес, построенный к очередному шагу. Новое ребро можно добавить к этому лесу так, чтобы он остался лесом, только если оно соединяет вершины из разных компонент связности леса *F*. Теперь именно такие ребра будем называть подходящими. Для выбора добавляемого ребра применяется тот же принцип, что и в алгоритме Прима – из всех подходящих ребер выбирается ребро наименьшего веса. Для того чтобы облегчить поиск этого ребра, вначале все ребра графа упорядочиваются по возрастанию весов: $w(e_1) \le \ldots \le w(e_m)$. Теперь последовательность ребер $e_1,\ldots,e_m$ достаточно просмотреть один раз и для очередного рассматриваемого ребра нужно определить, является ли оно подходящим относительно построенного к этому моменту леса *F*. Подходящие ребра добавляются к F, остальные просто пропускаются.

Теорема об алгоритме Краскала. *Лес F, построенный с помощью алгоритма Краскала, является каркасом наименьшего веса для данного графа.*

Скорость работы алгоритма Краскала зависит от того, как организовано хранение информации о текущем лесе и как выполняется проверка подходящих ребер. Применение специальных структур данных, которые мы не будем здесь рассматривать, позволяет реализовать этот алгоритм с оценкой времени работы $O(mf(n))$, где $f(n)$ – очень медленно растущая функция. В частности, $f(n) < 5$ для $n < 2^{65536}$.

Кратчайшие пути

Ранее уже была рассмотрена задача о кратчайших путях, для решения которой можно применять метод поиска в ширину. В той задаче под длиной пути понималось число ребер в нем, т.е. все ребра считались имеющими одну и ту же единичную длину. Теперь рассмотрим более общую задачу, в которой длины ребер могут быть различными. Предполагаем, что задан связный граф $G = (V, E)$ и для каждого его ребра (x, y) указана длина этого ребра $l(x, y)$ – неотрицательное число. Длина пути есть сумма длин его ребер.

Рассмотрим задачу отыскания путей наименьшей длины от заданной вершины *a* до всех остальных вершин графа. Возможно, более естественной постановкой задачи кажется поиск кратчайшего пути между двумя заданными вершинами. Однако в настоящее время не известно никакого способа решения этой задачи, существенно лучшего, чем поиск кратчайших путей от начальной вершины до всех остальных. Наиболее известным способом решения этой задачи является описываемый далее алгоритм Дейкстры (E. Dijkstra).

Решение задачи о кратчайших путях из заданной вершины удобно представлять в виде *дерева кратчайших путей.* Это дерево с корнем *a*, являющееся каркасом графа *G*, в котором путь от любой вершины до корня является кратчайшим путем между этими вершинами во всем графе.

Алгоритм Дейкстры очень похож на алгоритм Прима. В нем процесс построения дерева тоже начинается с одной вершины, только в алгоритме Прима это могла быть любая вершина графа, а теперь это должна быть вершина *a*. В дальнейшем на каждом шаге к дереву присоединяется одно новое ребро (и одна вершина). Это ребро выбирается из подходящих ребер, причем понятие подходящего ребра здесь такое же – это ребро, соединяющее вершину дерева с вершиной, ему не принадлежащей. И правило выбора добавляемого ребра такое же – среди подходящих ребер выбирается ребро наименьшего веса. Разница в том, что в алгоритме Прима веса ребер заданы изначально, а в алгоритме Дейкстры они вычисляются.

Пусть $T = (U, F)$ – частичное дерево кратчайших путей, построенное к некоторому моменту. Обозначим через $L(x)$ длину пути между вершинами *x* и *a* в дереве *T*. В качестве веса подходящего ребра (x, y) принимается величина $L(x) + l(x, y)$.

Теорема об алгоритме Дейкстры. *Дерево, построенное с помощью алгоритма Дейкстры, является деревом кратчайших путей.*

Алгоритм Дейкстры для ускорения работы модифицируется так же, как алгоритм Прима. Распространим определение функции L на множество $\overline{U}$: для $y \in \overline{U}$ положим $L(y)$ равным наименьшему значению величины $L(z) + l(z, y)$ по всем $z \in U$. Через $f(y)$ обозначим то значение *z*, на котором этот минимум достигается. Таким образом, $L(y)$ – это длина кратчайшего пути из *a* в *y*, проходящего только через вершины множества *U* (кроме вершины *y*), а $f(y)$ – предпоследняя вершина этого пути. Теперь алгоритм Дейкстры можно записать следующим образом.

Алгоритм Дейкстры.

7 $U := \{a\}$; $L(a) := 0$;
8 **for** $y \in \overline{U}$ **do** $\{ f(y) := a$; $L(y) := l(a, y) \}$;
9 **while** $U \neq V$ **do**
10 найти вершину $y \in \overline{U}$ с наименьшим значением $L(y)$;
11 добавить к *U* вершину *y*;
12 **for** $z \in \overline{U}$ **do** **if** $L(y) + l(y, z) < L(z)$ **then** $\{ f(z) := y$; $L(z) := L(y) + l(y, z) \}$.

Задачи

9.1. Найдите оптимальный каркас по матрице весов ребер с помощью алгоритма Прима (прочерк означает отсутствие ребра).

$$\begin{pmatrix} - & 2 & 4 & 2 & - & - & - \\ 2 & - & 2 & 3 & 3 & - & 1 \\ 4 & 2 & - & 3 & - & 3 & 2 \\ 2 & 3 & 3 & - & - & 1 & 3 \\ - & 3 & - & - & - & - & 4 \\ - & - & 3 & 1 & - & - & 4 \\ - & 1 & 2 & 3 & 4 & 4 & - \end{pmatrix} \qquad \begin{pmatrix} - & 3 & 5 & 3 & 4 & - & 2 \\ 3 & - & 1 & - & 3 & 4 & 3 \\ 5 & 1 & - & 2 & 1 & 3 & - \\ 3 & - & 2 & - & - & 4 & 4 \\ 4 & 3 & 1 & - & - & 2 & 2 \\ - & 4 & 3 & 4 & 2 & - & 5 \\ 2 & 3 & - & 4 & 2 & 5 & - \end{pmatrix}$$

9.2. Найдите оптимальный каркас с помощью алгоритма Краскала.

$$\begin{pmatrix} - & 4 & 3 & 2 & 3 & 2 & - & 1 \\ 4 & - & 1 & - & 1 & 5 & 3 & 4 \\ 3 & 1 & - & 3 & 2 & 3 & 5 & - \\ 2 & - & 3 & - & 4 & 2 & - & 2 \\ 3 & 1 & 2 & 4 & - & - & 3 & 4 \\ 2 & 5 & 3 & 2 & - & - & 2 & 3 \\ - & 3 & 5 & - & 3 & 2 & - & 2 \\ 1 & 4 & - & 2 & 4 & 3 & 2 & - \end{pmatrix}$$

9.3. Какие из следующих утверждений верны для любого взвешенного графа?

1) Если в графе имеется единственное ребро наименьшего веса, то оно принадлежит каждому оптимальному каркасу.
2) Если в графе имеются точно два ребра наименьшего веса, то они оба принадлежат каждому оптимальному каркасу.
3) Если в графе имеются точно три ребра наименьшего веса, то все они принадлежат каждому оптимальному каркасу.
4) Каждое ребро минимального веса принадлежит какому-нибудь оптимальному каркасу.

9.4. Вершины полного графа K_{pq} размещаются в в целочисленных точках прямоугольника $[1,\ p\ ]\times[1,\ q\ ]$. Вес каждого ребра равен евклидовой длине отрезка, соединяющего вершины этого ребра.

1) Чему равен вес оптимального каркаса для этого графа?
2) Каков будет вес оптимального каркаса, если из графа удалить все ребра длины 1?
3) Каков будет вес оптимального каркаса, если из графа удалить все ребра с длинами $1, 2$ и $\sqrt{2}$, а $p = q = 8$?
4) Каков будет вес оптимального каркаса, если из графа удалить все ребра с длинами 1 и $\sqrt{2}$, а $p = q = 8$?

9.5. По матрице длин ребер графа с помощью алгоритма Дейкстры найдите

1) кратчайшие пути от вершины 7 до всех остальных вершин
2) кратчайший путь между вершинами 1 и 4;

$$\begin{pmatrix} 0 & 3 & - & - & - & - & 2 \\ 3 & 0 & 2 & - & 6 & - & - \\ - & 2 & 0 & 2 & - & 1 & - \\ - & - & 2 & 0 & 5 & 5 & - \\ - & 6 & - & 5 & 0 & 1 & - \\ - & - & 1 & 5 & 1 & 0 & 1 \\ 2 & - & - & - & - & 1 & 0 \end{pmatrix}$$

9.6. Каркасы, построенные для некоторого графа с помощью алгоритмов Прима, Краскала и Дейкстры, имеют соответственно веса a, b и c. Какое из следующих соотношений обязательно выполняются для этих чисел?

1) $a \ge c$; 2) $a = b$; 3) $b \le c$; 4) $b = c$.

10. Потоки

В этом разделе будем рассматривать ориентированные графы без петель и кратных ребер. Для вершины x множество всех входящих в нее ребер обозначается через $E^+(x)$, а множество выходящих – через $E^-(x)$. *Сетью* называется орграф, в котором

1) каждому ребру e приписано положительное число $c(e)$, называемое *пропускной способностью* ребра;
2) выделены две вершины s и t, называемые соответственно *источником* и *стоком*, при этом $E^+(s) = E^-(t) = \varnothing$.

Вершины сети, отличные от источника и стока, будем называть *внутренними*.

Пусть задана сеть N с множеством вершин V и множеством ребер E. Пусть f – функция с вещественными значениями, определенная на множестве E. Для вершины x обозначим $f^+(x) = \sum_{e \in E^+(x)} f(e)$, $f^-(x) = \sum_{e \in E^-(x)} f(e)$. Функция f называется *потоком* в сети N, если она удовлетворяет условиям:

(1) $0 \le f(e) \le c(e)$ для каждого ребра e;

(2) $f^+(x) = f^-(x)$ для каждой внутренней вершины x.

На рисунке 11 показан пример сети и потока в ней. В дроби, приписанной каждому ребру, числитель представляет пропускную способность ребра, а знаменатель – величину потока на этом ребре.

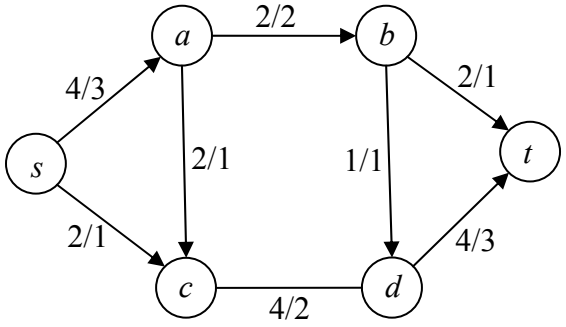


Рис. 11.

Условие (2) называется *условием сохранения потока*. Так как каждое ребро является входящим для одной вершины и выходящим для другой, то

$$\sum_{x \in V} f^+(x) = \sum_{x \in V} f^-(x).$$

Из этого равенства и условия сохранения потока следует, что

$$f^-(s) = f^+(t).$$

Эта величина обозначается через $M(f)$ и называется *величиной потока*. В примере на рисунке $M(f) = 4$. *Задача о максимальном потоке* состоит в том, чтобы для данной сети найти поток наибольшей величины.

Поток на рисунке 11 не является максимальным – можно, например, добавить по единице на ребрах пути s, a, c, d, t. Получится поток величины 5, показанный на рисунке 12 слева. Но и он не максимален. Можно увеличить поток на 1 на ребрах (s,c), (c,d),

(b,t) и уменьшить на 1 на ребре (b,d). Условие сохранения останется выполненным, а величина потока станет равной 6 (рисунок 12, справа).

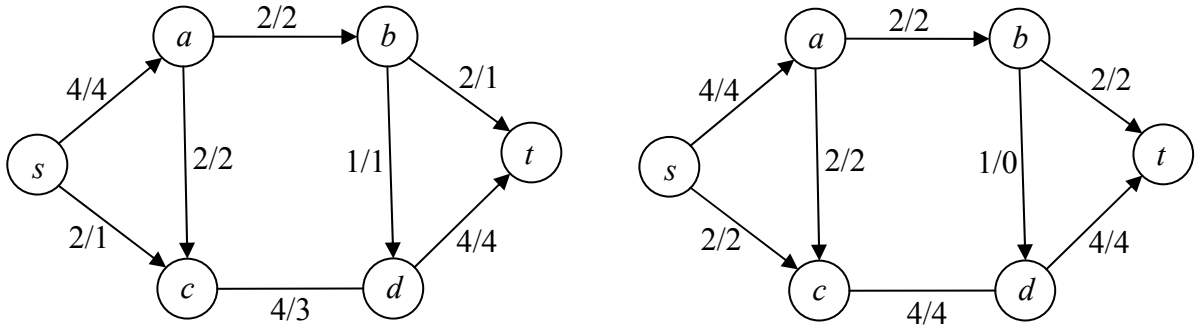


Рис. 12.

Приведенный пример иллюстрирует общий метод, на котором основаны многие алгоритмы решения задачи о максимальном потоке – метод увеличивающих путей. Он является обобщением метода увеличивающих путей для задачи о паросочетании.

Допустим, имеется сеть N и в ней поток f. Пусть $x_1, e_1, x_2, e_2, \ldots, e_{k-1}, x_k$ – неориентированный путь в сети. Ребро e_i назовем *прямым* ребром этого пути, если $e_i = (x_i, x_{i+1})$, и *обратным*, если $e_i = (x_{i+1}, x_i)$. Путь назовем *подходящим* относительно потока f, если для каждого прямого ребра выполняется неравенство $f(e_i) < c(e_i)$, а для каждого обратного – неравенство $f(e_i) > 0$. Таким образом, на каждом прямом ребре подходящего пути поток можно увеличить, а на каждом обратном – уменьшить. *Увеличивающий путь* – это подходящий путь из источника в сток.

Если имеется увеличивающий путь для потока f, то поток можно увеличить на величину δ, равную минимуму из величин «недогрузок» (разностей $c(e) - f(e)$) прямых ребер этого пути и величин потока на обратных ребрах. Нужно прибавить δ к потоку на каждом прямом ребре пути и вычесть δ из потока на каждом обратном ребре. Условие сохранения останется выполненным, а величина потока увеличится на δ.

Теорема об увеличивающем пути. *Поток максимален тогда и только тогда, когда относительно него нет увеличивающего пути.*

Пусть $X \subseteq V$, $\overline{X} = V - X$, причем $s \in X$, $t \in \overline{X}$. Множество всех ребер, у которых начальная вершина принадлежит X, а концевая – $\overline{X}$, называется *разрезом* сети. Пропускная способность разреза есть сумма пропускных способностей его ребер.

Теорема о максимальном потоке и минимальном разрезе. *Максимальная величина потока равна минимальной пропускной способности разреза данной сети.*

Метод увеличивающих путей для задачи о максимальном потоке предложили Форд и Фалкерсон в 1955 г. Известно несколько алгоритмов, реализующих этот метод, они различаются, в частности, стратегией поиска увеличивающих путей. Первый алгоритм, для которого была получена верхняя оценка трудоемкости, предложили Эдмондс и Карп в 1972 г. В этом алгоритме всегда ищется кратчайший (по числу ребер) увеличивающий путь. Удобно этот поиск вести не на исходной сети N, а на *остаточной сети* R, которая при заданном на сети N потоке f определяется следующим образом. Множество вершин,

источник и сток у остаточной сети те же, что у исходной. Пусть e – ребро исходной сети. Тогда

1) если $f(e) < c(e)$, то ребро e включается в сеть R и ему в этой сети присваивается пропускная способность $c'(e) = c(e) - f(e)$;
2) если $f(e) > 0$, то к сети R добавляется ребро противоположного направления $\overline{e}$ с пропускной способностью $c'(\overline{e}) = f(e)$.

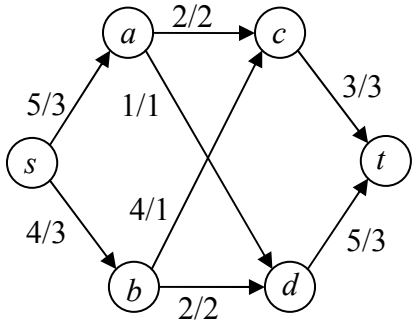
Легко видеть, что увеличивающие пути в исходной сети находятся во взаимно однозначном соответствии с ориентированными путями из источника в сток в остаточной сети. В алгоритме Эдмондса-Карпа нужно в остаточной сети искать кратчайший ориентированный путь из s в t. Это можно сделать за линейное время с помощью поиска в ширину. Если увеличивающий путь обнаружен, поток увеличивается. Для нового потока строится остаточная сеть и т.д., пока не будет построен поток, относительно которого нет увеличивающего пути (в остаточной сети нет ориентированного пути из источника в сток. Общая оценка трудоемкости алгоритма Эдмондса-Карпа $O(m^2 n)$. В настоящее время известны и более быстрые алгоритмы для задачи о максимальном потоке.

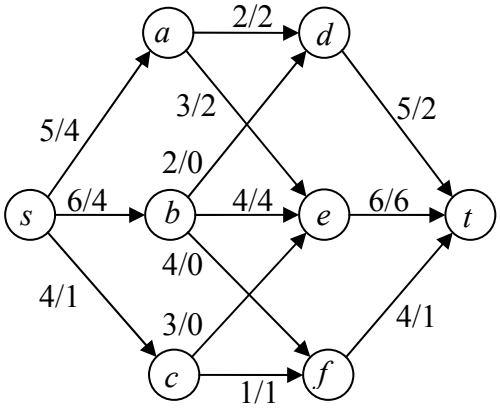
Когда максимальный поток найден, нетрудно найти и минимальный разрез в сети, т.е. разрез с минимальной пропускной способностью. Нужно найти все вершины, достижимые из источника подходящими путями. Пусть X – множество всех таких вершин, тогда множество всех ребер из X в $\overline{X}$ является минимальным разрезом.

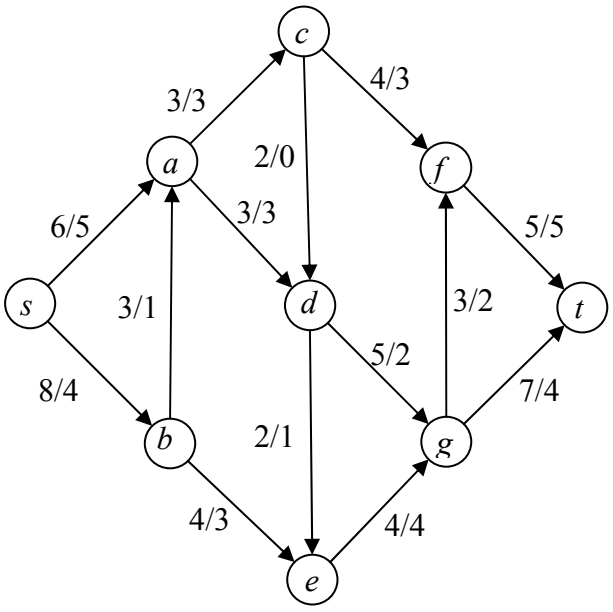
Алгоритм нахождения максимального потока можно применять для нахождения наибольшего паросочетания в двудольном графе. Пусть A и B – доли такого графа. Ориентируем все ребра графа по направлению от A к B. Добавим источник s и ребра из s ко всем вершинам доли A. Добавим сток t и ребра от всех вершин доли B к t. Припишем каждому ребру пропускную способность 1. Найдем в построенной сети целочисленный максимальный поток. Ребра между A и B, на которых поток равен 1, образуют наибольшее паросочетание.

Задачи

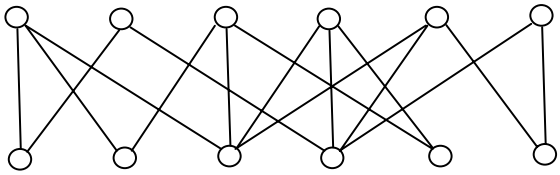
10.1. Выясните, является ли данный поток максимальным. Если да, то найдите минимальный разрез. Если нет, то найдите максимальный поток.







10.2. Найдите наибольшее паросочетание с помощью потокового алгоритма.



Литература

1. Алексеев В.Е., Таланов В.А. Графы. Модели вычислений. Алгоритмы.– Н.Новгород, ННГУ, 2005. 307 с.
2. Алексеев В.Е., Таланов В.А. Графы и алгоритмы. Структуры данных. Модели вычислений.– М.: Интернет-университет информационных технологий; БИНОМ. Лаборатория знаний, 2006. 320 с.
3. Берж К. Теория графов и ее применение.– М.: ИЛ, 1962. 319 с.
4. Емеличев В.А., Мельников О.И., Сарванов В.И., Тышкевич Р.И. Лекции по теории графов.– М.: Наука, 1990. 383 с.
5. Зыков А.А. Основы теории графов.– М.: Вузовская книга, 2004. 664 с.
6. Касьянов В.Н., Евстигнеев В.А. Графы в программировании: обработка, визуализация и применение.– СПб.: БХВ-Петербург, 2003. 1104 с.
7. Кристофидес Н. Теория графов. Алгоритмический подход . – М.: Мир, 1978. 432 с.
8. Кормен Т.Х., Лейзерсон Ч.И., Ривест Р.Л., Штайн К. Алгоритмы: построение и анализ.– М.: Вильямс, 2005. 1296 с.
9. Липский В. Комбинаторика для программистов. – М.: Мир, 1988. 213 с.
10. Оре О. Теория графов.– М.: Наука, 1980. 336 с.
11. Седжвик Р. Фундаментальные алгоритмы на C++. Часть 5. Алгоритмы на графах. – СПб.: DiaSoftЮП, 2002. 496 с.
12. Харари Ф. Теория графов.– М.: Мир, 1973. 300 с.