ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

Нижегородский государственный университет им. Н.И.Лобачевского

http://vmk.ucoz.net/

Л.П. Жильцова, Т.Г. Смирнова

Основы теории графов и теории кодирования в примерах и задачах

Учебно-методическое пособие

Рекомендовано методической комиссией факультета вычислительной математики и кибернетики для студентов ННГУ, обучающихся по специальности
080801 «Прикладная информатика (в области принятия решений)»

Нижний Новгород
2008

УДК 519.17 + 519.711.4
ББК В182
Ж − 72

Ж−72 Жильцова Л.П., Смирнова Т.Г. Основы теории графов и теории кодирования в примерах и задачах: Учебное пособие. – Нижний Новгород: Издательство Нижегородского госуниверситета, 2008. – 64 с.

Рецензент: канд. физ.-мат. наук, ст.н.сотр. НИИ ПМК **В.И. Шевченко**

В настоящем пособии рассматриваются основные понятия и алгоритмы теории графов и теории кодирования. Изучение каждой темы сопровождается необходимым теоретическим материалом и примерами решения типовых задач, а также предлагаются задачи для самостоятельного решения.

Учебно-методическое пособие предназначено для студентов второго курса факультета ВМК специальности «Прикладная информатика (в области принятия решений)», изучающих курс “Дискретная математика”.

УДК 519.17 + 519.711.4
ББК В182

Глава 1. ЭЛЕМЕНТЫ КОМБИНАТОРИКИ

Пусть A и B – множества, тогда отображение $f : A \to B$ есть:

1) ***инъекция***, если. любые два различных элемента из A отображаются в различные элементы множества B;

2) ***сюръекция***, если для любого $y \in B$ существует $x \in A$, что $f(x) = y$;

3) ***биекция* (*взаимно-однозначное отображение*)**, если оно инъективно и сюръективно.

$|A|$ – число элементов конечного множества A (***мощность*** множества).

Правило равенства: если между конечными множествами A и B существует взаимно-однозначное соответствие, то $|A| = |B|$.

Правило суммы: если A и B – непересекающиеся конечные множества, то $|A \cup B| = |A| + |B|$.

Правило произведения: для любых конечных множеств A и B имеет место равенство $|A \times B| = |A| \cdot |B|$. В более общем виде, если элемент a можно выбрать k способами, и после этого, независимо от того, какой элемент a был выбран, элемент b можно выбрать n способами, то упорядоченную пару (a,b) можно выбрать $k \cdot n$ способами.

Пусть $A = \{a_1,...,a_n\}$ − конечное множество. ***Размещением (перестановкой)*** из A по k называется упорядоченное подмножество $(a_{i_1},...,a_{i_k})$ из k элементов множества A. Число всех размещений из n элементов по k обозначается через A_n^k. При подсчете числа размещений замечаем, что первым элементом может быть любой из n элементов множества A, вторым элементом − любой из $(n\text{-}1)$ оставшихся в A элементов и т.д. Поэтому $A_n^k = n(n-1)...(n-k+1) = \dfrac{n!}{(n-k)!}$.

Перестановками элементов множества $A = \{a_1,...,a_n\}$ называются всевозможные упорядоченные множества из n элементов $a_1,...,a_n$. Перестановки из n элементов − частный случай размещений из n элементов по n, поэтому число всех перестановок из n элементов $A_n^n = n(n-1)...2 \cdot 1 = n!$

Сочетанием элементов из множества A по k называется неупорядоченное подмножество $(a_{i_1},...,a_{i_k})$ из k элементов множества A. Число всех сочетаний из n элементов по k обозначается через C_n^k или $\binom{n}{k}$. В сочетании, в отличие от

размещения, порядок следования элементов не учитывается и из одного сочетания из n элементов по k получается $k!$ размещений. Получаем $C_n^k = \binom{n}{k} = \frac{n!}{k!(n-k)!}$.

Сочетанием с повторениями элементов из множества A по k называется неупорядоченный набор $(a_{i_1},...,a_{i_k})$ из k элементов множества A, в котором элементы могут повторяться. Число всех сочетаний с повторениями из n элементов по k обозначается через $\hat{C}_n^k$. Пусть в сочетании элемент a_i встречается s_i раз, тогда каждому сочетанию поставим в соответствие набор из n-1 нулей и k единиц: $\underbrace{1...1}_{s_1}0\underbrace{1...1}_{s_2}0...0\underbrace{1...1}_{s_n}$. Это соответствие между сочетаниями и наборами из n-1 нулей и k единиц взаимно однозначно. Так как число наборов длины $n+k-1$, содержащих n-1 нулей и k единиц, имеется $\binom{n+k-1}{k}$, получаем $\hat{C}_n^k = \binom{n+k-1}{k} = \frac{(n+k-1)!}{k!\cdot(n-1)!}$.

Пример 1.1. Для $A=\{a,b,c\}$ всевозможными размещениями по 2 элемента будут пары (*a*, *b*), (*b*, *a*), (*b*, *c*), (*c*, *b*), (*a*, *c*), (*c*, *a*), сочетаниями по 2 элемента будут (*a*, *b*), (*a*, *c*), (*b*, *c*), сочетаниями с повторениями из *A* по 2 элемента − (*a*, *b*), (*a*, *c*), (*b*, *c*), (*a*, *a*), (*b*, *b*), (*c*, *c*).

Всевозможными перестановками элементов множества $A=\{a,b,c\}$ являются упорядоченные наборы (*a*,*b*,*c*), (*a*,*c*,*b*), (*b*,*a*,*c*), (*b*,*c*,*a*), (*c*,*a*,*b*), (*c*,*b*,*a*).

Разбиением множества A называется множество его непустых подмножеств $A_1,...,A_k$, если выполнены условия:

1) $A_i \cap A_j = \varnothing$ для всех $i \neq j$; 2) $A = \bigcup_{i=1}^{k} A_i$.

Число всех возможных упорядоченных разбиений множества A мощности *n* на *k* подмножеств мощностей $n_1, n_2, \ldots, n_k$ равно

$$P_n(n_1,n_2,...,n_k) = \binom{n}{n_1,n_2,...,n_k} = \frac{n!}{n_1!n_2!...n_k!}.$$

Пример 1.2. Дано множество U из 7 элементов. Каким числом способов в нем можно выбрать три подмножества A, B, C так, чтобы выполнялись условия: $|AC \cup B| = 2$, $|\overline{A}\,\overline{B}\,C| = 2$.

Изобразим подмножества A, B, C на диаграмме Венна(см. рис.1.1). Введем обозначения: $A_1 = AC \cup B$, $A_2 = \overline{A}\,\overline{B}\,C$, $A_3 = \overline{B}\,\overline{C}$.

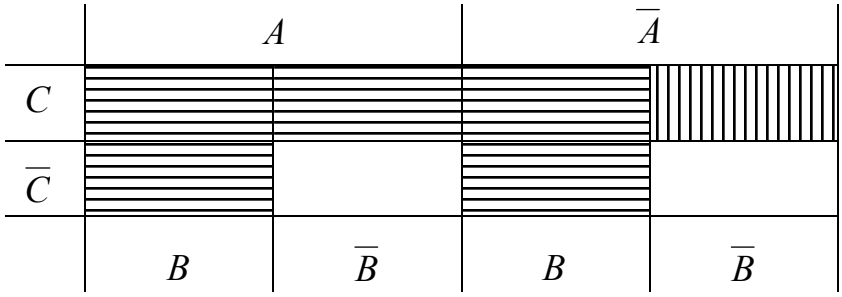


Рис.1.1

Заметим, что $|A_1| = 2$ (это подмножество размещено в пяти ячейках диаграммы Венна – горизонтальная штриховка), $|A_2| = 2$ (одна ячейка диаграммы – вертикальная штриховка) и $|A_3| = 3$ (две незаштрихованные ячейки диаграммы), т.е. подмножества A_1, A_2, A_3 задают разбиение множества U. Тогда получаем, что число способов выбора подмножеств A, B, C множества U равно $P_7(2,2,3) \cdot 5^2 \cdot 2^3 = \dfrac{7!}{2! \cdot 2! \cdot 3!} \cdot 5^2 \cdot 2^3 = 42\,000$.

Пример 1.3. Сколько различных слов можно составить, переставляя буквы слова «математика»?

Всего в слове «математика» 10 букв, причем буква «м» встречается 2 раза, буква «а» – 3 раза, буква «т» – 2 раза, буква «е» – 1 раз, буква «и» - 1 раз, буква «к» – 1 раз, тогда число всех различных слов, которые можно составить, равно $P_{10}(2,3,2,1,1,1) = \dfrac{10!}{2! \cdot 3! \cdot 2!} = 151\,200$.

Пример 1.4. Сколько слов длины 9 в алфавите $\{a_1, a_2, a_3, a_4, a_5, a_6\}$ можно составить при условии, что $n_1 = n_2 = n_3 = n_4$, где n_i обозначает число вхождений буквы a_i в слово.

Если $n_1 = n_2 = n_3 = n_4 = 0$, тогда $n_5 + n_6 = 9$, и число слов, удовлетворяющих этому условию, равно числу двоичных последовательностей длины 9.

Если $n_1 = n_2 = n_3 = n_4 = 1$, тогда $n_5 + n_6 = 5$, и, рассматривая всевозможные разложения числа 5 на упорядоченную сумму двух слагаемых, получаем, что число слов в этом случае равно $P_9(1,1,1,1,0,5) + P_9(1,1,1,1,1,4) +$

$+P_9(1,1,1,1,2,3)+P_9(1,1,1,1,3,2)+P_9(1,1,1,1,4,1)+P_9(1,1,1,1,5,0)=2\left(\frac{9!}{5!}+\frac{9!}{4!}+\frac{9!}{2!\cdot 3!}\right)=$

=96768. Наконец, если $n_1=n_2=n_3=n_4=2$, тогда $n_5+n_6=1$, и число слов равно $P_9(2,2,2,2,0,1)+P_9(2,2,2,2,1,0)=2\cdot\frac{9!}{2!\cdot 2!\cdot 2!\cdot 2!}=45\,360$. По правилу суммы получаем, что можно составить $512+96\,768+45\,360=142\,640$ слов.

Пример 1.5. Сколькими способами можно переставить буквы слова «комиссия» так, чтобы:

1) никакие две гласные не стояли рядом;
2) не менялся порядок гласных букв;
3) две буквы «с» не шли подряд?

1) Имеется $P_4(1,1,2)=\frac{4!}{2!}=12$ перестановок согласных. Для каждого размещения согласных имеем 5 мест для размещения четырех гласных слова, тогда число всех размещений гласных букв слова равно $\frac{A_5^4}{2!}=\frac{5!}{2!}=60$. По правилу произведения получаем $12\cdot 60=720$ слов.

2) Число размещений согласных букв с учетом того, что буква «с» повторяется дважды равно $\frac{A_8^4}{2!}=\frac{8!}{4!\cdot 2!}=840$. Очевидно, что на оставшиеся места гласные в указанном порядке размещаются однозначно.

3) Число различных слов, которые получаются перестановками букв слова «комиссия» равно $P_8(1,1,1,1,2,2)=\frac{8!}{2!\cdot 2!}=10\,080$. Число слов, в которых две буквы «с» идут подряд равно $P_7(1,1,1,1,1,2)=\frac{7!}{2!}=2\,520$. Получаем, $10\,080-2\,520=7\,560$ слов, в которых две буквы «с» не идут подряд.

Бином Ньютона: $(x+y)^n=\sum_{k=0}^{n}\binom{n}{k}x^k y^{n-k}$.

Формула включений и исключений для n множеств:

$$\left|\bigcup_{i=1}^{n}A_i\right|=\sum_{k=1}^{n}(-1)^{k-1}\sum_{\{i_1,\ldots,i_k\}\subseteq\{1,2,\ldots,n\}}\left|A_{i_1}\cap\ldots\cap A_{i_k}\right|.$$

Задачи

1.1. Имеется n_1 книг одного автора, n_2 – второго, n_3 – третьего. Каким числом способов можно выбрать

1) одну книгу;
2) две книги разных авторов;
3) три книги разных авторов;
4) две книги одного автора;
5) три книги одного автора;
6) одну книгу первого автора, две – второго и три – третьего?

1.2. Каким числом способов можно заполнить анкету, содержащую n вопросов, если на каждый вопрос можно ответить

1) `да` или `нет`;
2) `да`, `нет`, `не знаю`?

1.3. Сколько слов длины n можно составить, если в алфавите k букв? Сколько среди них палиндромов, т.е. слов, читающихся одинаково слева направо и справа налево?

1.4. Сколько в k-буквенном алфавите имеется слов длины n, в которых любые две соседние буквы различны?

1.5. Сколько матриц с m строками и n столбцами можно составить из элементов 0 и 1?

1.6. Сколько матриц с m попарно различными строками и n столбцами можно составить из элементов 0 и 1?

1.7. Каким числом способов можно на обычной шахматной доске разместить белую и черную ладьи так, чтобы они не атаковали друг друга?

1.8. Каким числом способов можно на шахматной доске поместить черного и белого королей так, чтобы они не атаковали друг друга?

1.9. Сколько имеется четырехзначных чисел, у которых каждая следующая цифра:

1) больше предыдущей;
2) меньше предыдущей?

1.10. В комнате n лампочек.

1) Сколько всего может быть различных способов освещения комнаты?

2) Сколько всего может быть разных способов освещения комнаты, при которых горит ровно k $(k \le n)$ лампочек?

1.11. Сколько имеется вариантов выбора трех призеров среди n участников конкурса:

1) с указанием занимаемых ими мест;

2) без указания мест?

1.12. Дано множество U из n элементов и в нем подмножество A из k элементов. Найти число подмножеств $B \subseteq U$, удовлетворяющих условию:

1) $B \subset A$;

2) $B \supset A$;

3) $A \cap B = \varnothing$;

4) $A \cap B \neq \varnothing$;

5) $|B \cap A| = 1$;

6) $|B \cap A| \geq 2$.

7) $|B \cap A| = 2$;

8) $|B \cap \overline{A}| = 2$;

9) $|B - A| = 3, |A - B| = 4$;

10) $|A \otimes B| = 1$.

1.13. Сколько делителей у числа 2048? 2310? 2880? Сколько делителей имеет число $p_1^{k_1} \ldots p_s^{k_s}$, где $p_1, \ldots, p_s$ –различные простые числа, $k_1, \ldots, k_s$ – целые неотрицательные?

1.14. Сколькими способами можно расставить восемь ладей на обычной шахматной доске так, чтобы они не угрожали друг другу, т.е. чтобы никакие две из них не стояли на одной вертикали или горизонтали?

1.15. Из колоды, содержащей 52 карты, вынули 10 карт. Сколькими способами это можно сделать? В скольких случаях среди этих карт окажется:

1) хотя бы один туз;

2) ровно один туз;

3) ровно четыре туза;

4) не менее двух тузов;

5) ровно два туза?

1.16. Имеется колода из $4n$ карт четырех мастей, по n карт каждой масти, занумерованных числами $1,2,...,n$. Каким числом способов можно выбрать пять карт так, чтобы среди них оказались:

1) пять карт одной масти с последовательными номерами;
2) четыре карты с одинаковыми номерами;
3) три карты с одним номером и две карты с другим;
4) пять карт одной масти;
5) пять карт с последовательными номерами;
6) в точности три карты из пяти с одинаковыми номерами;
7) две карты с одинаковыми, остальные с разными номерами?

1.17. Сколько имеется пятизначных десятичных чисел, у которых:

1) все цифры различны;
2) есть одинаковые цифры;
3) все цифры различны, причем последняя – не 0;
4) все цифры разные, причем первая – не 9, последняя – не 0;
5) две первых цифры различны, а две последних –одинаковы;
6) сумма цифр четна?

1.18. Каким числом способов можно разместить n различных предметов по k различным ящикам? Сколько таких размещений, если в каждый ящик укладывается не более одного предмета?

1.19. Если компьютерный пароль содержит пять символов, которые могут быть цифрой или буквой латинского алфавита, то сколько имеется паролей, начинающихся с буквы?

1.20. На книжной полке требуется расположить 4 различные книги по математике, 5 различных книг по программированию и 2 различные книги по истории. Сколькими способами это можно сделать, если:

1) нет ограничений по расстановке;
2) все книги по одному и тому же предмету должны стоять вместе;
3) все книги по одному и тому же предмету должны стоять вместе, но книги по математике и программированию не должны стоять рядом?

1.21. На книжной полке требуется расположить 4 одинаковые книги по математике, 5 одинаковых книг по программированию и 2 одинаковые книги по истории. Сколькими способами это можно сделать?

1.22. Каким числом способов из 10 человек можно выбрать три комиссии, если в первой и во второй комиссиях должно быть по 3 человека, а в третьей – 5 человек, и ни один из членов первой комиссии не должен входить во вторую и третью?

1.23. Сколько различных слов можно составить, переставляя буквы в слове “программа”?

1.24. Каким числом способов можно разместить 7 студентов в трех комнатах общежития, если в одной комнате имеется одно, в другой – два, в третьей – четыре свободных места?

1.25. Среди сотрудников фирмы семнадцать человек знают английский язык, десять – немецкий, семеро – французский. Три человека знают английский и французский, два – немецкий и французский, четверо – английский и немецкий.

1) Сколько человек работает в фирме, если каждый знает хотя бы один язык, а два человека знают все три языка?
2) Сколько сотрудников, не знающих ни одного иностранного языка, если в фирме работает тридцать человек и никто из них не знает всех трех языков?

1.26. На одной из кафедр университета работают тринадцать человек, причем каждый из них знает хотя бы один иностранный язык. Десять человек знают английский, семеро – немецкий, шестеро – французский. Пятеро знают английский и немецкий, четверо – английский и французский, трое – немецкий и французский.

1) Сколько человек знают все три языка?
2) Сколько человек знают ровно два языка?
3) Сколько человек знают только английский язык?

1.27. Из 100 опрошенных студентов 50 программируют на алгоритмическом языке Си++, 53 − на Паскале, 42 − на Бейсике, 15 студентов могут программировать на Си++ и на Бейсике, 20 студентов − на Паскале и Бейсике, 25 − на Си++ и Паскале, а 5 студентов программируют на всех трех языках.

1) Сколько студентов не могут программировать ни на одном из перечисленных языков?
2) Сколько студентов программируют хотя бы на одном из перечисленных языков?
3) Сколько студентов программируют только на Паскале?

4) Сколько студентов не программируют ни на Си++, ни на Паскале?

1.28. Дано множество U из n элементов. Каким числом способов в нем можно выбрать три подмножества A, B, C так, чтобы выполнялись заданные условия:

1) $n=7,\ |(A-B)\cup C|=6,\ |C-(A\cap B)|=2;$
2) $n=6,\ |A\cup B|=5,\ |A-(B\cap C)|=1;$
3) $n=7,\ |(A\cap B)-C|=4,\ |C\cap(A\cup B)|=1;$
4) $n=6,\ |(A-B)\cup C|=4,\ |A\cap B\cap C|=1;$
5) $n=8,\ |A\cup B\cup C|=4,\ |A\cap B|=1.$

1.29. Рассматриваются слова в алфавите $\{a_1,a_2,\ldots,a_q\}$. Через n_i обозначается число вхождений буквы a_i в слово. Требуется подсчитать число слов длины n, удовлетворяющих данным условиям:

1) $q=5,\ n=8,\ n_1+n_2+n_3=2;$
2) $q=4,\ n=8,\ n_2=n_1+2;$
3) $q=3,\ n=9,\ n_1+n_2<n_3;$
4) $q=3,\ n=9,\ 2n_1\le n_2+n_3;$
5) $q=5,\ n=7,\ n_1+n_2+n_3=2,\ n_4\ge 3.$

Глава 2. ТЕОРИЯ ГРАФОВ

2.1. Основные понятия теории графов

Граф $G=(V,E)$ задается непустым множеством вершин V и множеством ребер E, состоящим из пар элементов V. Если рассматриваются неупорядоченные пары, граф называется ***неориентированным***, если упорядоченные – ***ориентированным***. Если ребро (a,b) принадлежит графу, то вершины a и b называют ***смежными***. Ребро вида (a,a) называется ***петлей***. Неориентированный граф без петель называют ***обыкновенным***. Во всех задачах этого раздела, где термин “граф” употребляется без уточнения, имеются в виду обыкновенные графы.

Графы $G_1=(V_1,E_1)$ и $G_2=(V_2,E_2)$ называют ***изоморфными***, если существует биекция $f: V_1 \to V_2$, такая, что для любых $a,b \in V_1$ $(a,b) \in E_1$ тогда и только тогда, когда $(f(a),f(b)) \in E_2$. Эта биекция f называется ***изоморфизмом*** графа G_1 на граф G_2. Если графы G_1 и G_2 изоморфны, то пишем, что $G_1 \cong G_2$. Отношение изоморфизма графов есть отношение эквивалентности, так как оно рефлексивно, симметрично и транзитивно. Следовательно, множество всех графов разбивается на классы так, что графы из одного класса попарно изоморфны, а графы из разных классов не изоморфны.

Степень вершины a – количество смежных с ней вершин, обозначается через $d(a)$. ***Набор степеней*** графа – упорядоченная по неубыванию последовательность степеней вершин.

Теорема 2.1.1. *Пусть в графе* $G=(V,E)$ $|V|=n, |E|=m$, *тогда* $\sum_{a \in V} d(a) = 2m$.

Дополнительный граф к графу $G=(V,E)$ – такой граф $\overline{G}=(V',E')$, у которого $V'=V$, а $(a,b) \in E'$ тогда и только тогда, когда $(a,b) \notin E$.

Подграф графа $G=(V,E)$ – такой граф $G'=(V',E')$, что $V' \subseteq V$, $E' \subseteq E$.

Остовный подграф графа $G=(V,E)$ – такой граф $G'=(V',E')$, у которого $V'=V$, $E' \subseteq E$, т.е. остовный подграф получается из исходного графа удалением только ребер без удаления вершин.

Порожденный подграф графа $G=(V,E)$ – такой граф $G'=(V',E')$, у которого $V'\subseteq V$, $E'=\{(a,b)\in E \mid a,b\in V'\}$, т.е. порожденный подграф получается из исходного графа удалением вершин и всех ребер, инцидентных удаленным вершинам.

Двудольный граф – граф, множество вершин которого можно разбить на две части (доли) так, что концы каждого ребра лежат в разных частях.

K_n – ***полный граф*** с n вершинами, т.е. граф, в котором любые две вершины смежны. O_n – ***пустой граф*** с n вершинами, т.е. граф, в котором никакие две вершины не смежны.

$K_{p,q}$ – ***полный двудольный граф***. В нем множество вершин можно разбить на две части V_1 и V_2, причем $|V_1|=p$, $|V_2|=q$, и две вершины смежны тогда и только тогда, когда они принадлежат разным долям.

Q_n –***n-мерный куб.*** Вершинами этого графа являются все двоичные наборы длины n и две вершины смежны тогда и только тогда, когда соответствующие наборы отличаются ровно в одной позиции.

Задачи

2.1.1. Построить граф пересечений граней куба. Написать матрицу смежности полученного графа.

2.1.2. В графе n вершин и m ребер. Сколько у него

1) остовных подграфов;
2) порожденных подграфов?

2.1.3. Определить число графов с n вершинами, в которых допускаются ребра следующих типов:

1) неориентированные и петли;
2) ориентированные и петли;
3) ориентированные, но не петли.

2.1.4. Выяснить, существуют ли графы с набором степеней:

1) (0,2,2,3,3); 2) (2,2,2,3,3);
3) (2,2,3,3,3); 4) (0,1,2,3,4).

2.1.5. Определить число ребер в каждом из графов K_n, $K_{p,q}$, Q_n.

2.1.6. При каких n существуют графы с n вершинами, каждая из которых имеет степень 3? степень 4?

2.1.7. Графы, изображенные на рис. 2.1, разбить на классы попарно неизоморфных графов.

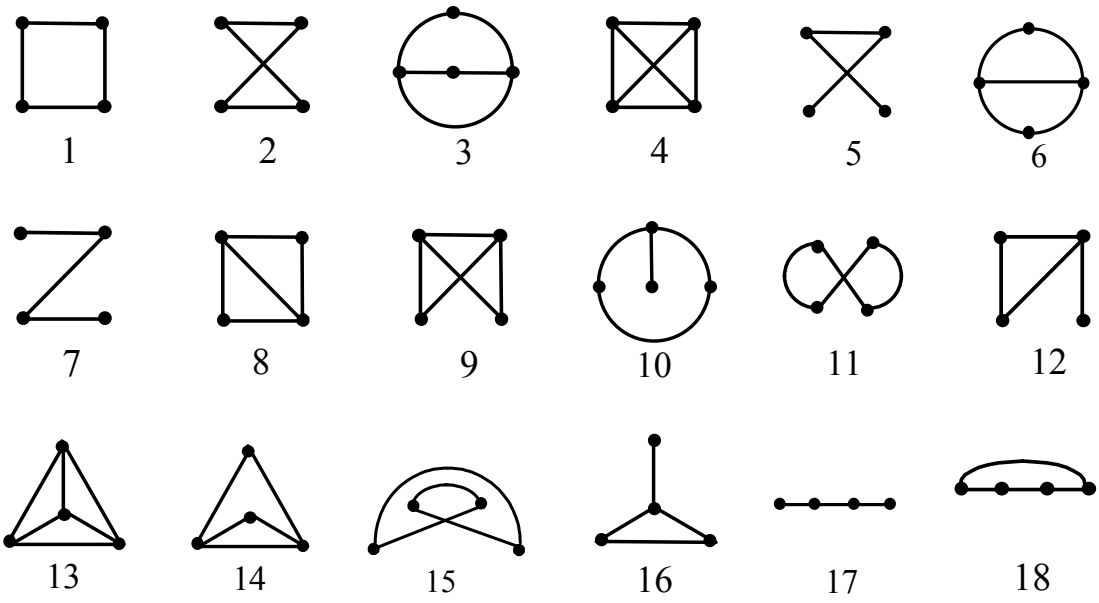


Рис. 2.1

2.1.8. Перечислить все попарно неизоморфные графы:

1) с 4 вершинами;
2) с 6 вершинами и 3 ребрами;
3) с 6 вершинами и 13 ребрами.

2.1.9. Графы, изображенные на рис. 2.2, разбить на классы попарно неизоморфных графов.

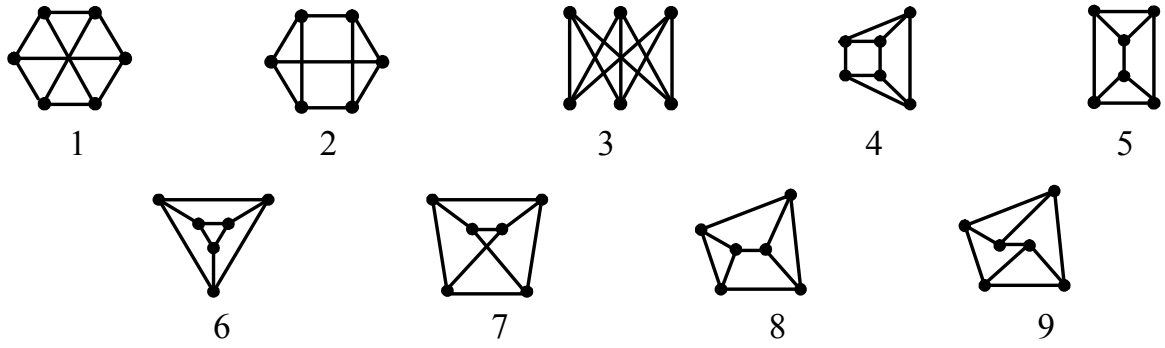


Рис. 2.2

2.1.10. Графы, изображенные на рис. 2.3, разбить на классы попарно неизоморфных графов.

Рис. 2.3

2.1.11. Графы, изображенные на рис. 2.4, разбить на классы попарно неизоморфных графов.

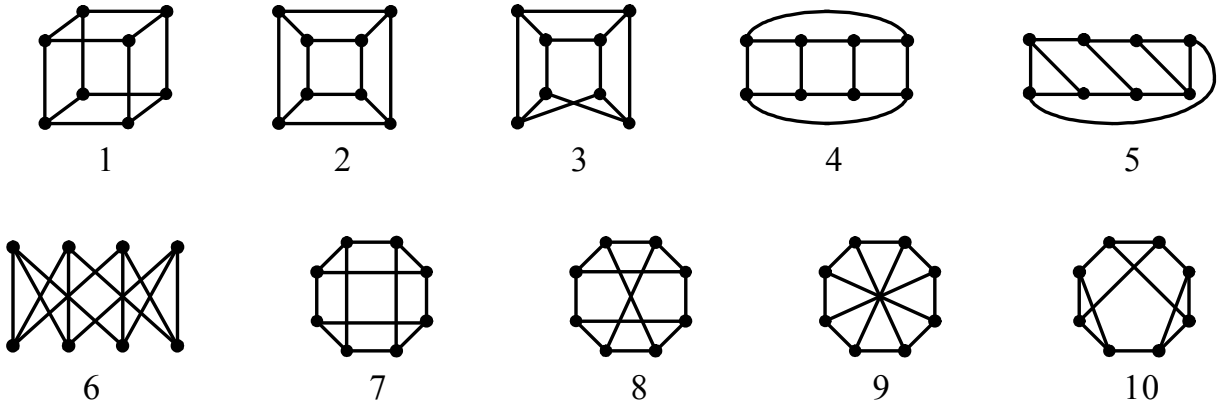


Рис. 2.4

2.1.12. Граф G называется самодополнительным, если $G \cong \overline{G}$. Найти все самодополнительные графы с числом вершин, не превосходящим 6.

2.1.13. Доказать, что в каждом графе с не менее чем двумя вершинами найдутся две вершины с одинаковыми степенями.

2.1.14. Какие из графов, изображенных на рис. 2.4, являются двудольными?

2.1.15. Каково наибольшее число ребер в двудольном графе с n вершинами?

2.2. Пути, циклы, обходы графа. Метрические характеристики

Последовательность вершин $a_1, a_2, ..., a_k$, такая, что $(a_i, a_{i+1}) \in E$ для всех $i = 1, ..., k-1$, называется ***маршрутом,*** соединяющим вершины a_1 и a_k. ***Длина маршрута*** – число ребер $k-1$.

Путь − это маршрут, в котором все ребра различны. ***Простой путь*** – путь, в котором все вершины различны.

Цикл – это замкнутый путь, в котором $a_1 = a_k$ и все ребра различны. ***Простой цикл*** – цикл, в котором вершины $a_1,...,a_{k-1}$ различны.

Теорема 2.2.1. *В любом маршруте, соединяющем две различные вершины, содержится простой путь, соединяющий те же вершины. В любом цикле, проходящем через некоторое ребро, содержится простой цикл, проходящий через это ребро.*

P_n − простой путь с n вершинами.

C_n − простой цикл с n вершинами.

Теорема 2.2.2. *Если в графе степень каждой вершины не меньше 2, то в нем есть цикл.*

Связный граф – такой граф, в котором для любых двух вершин имеется маршрут, соединяющий эти вершины.

Компонента связности графа – связный подграф, не содержащийся в большем связном подграфе.

Перешеек – ребро, при удалении которого увеличивается число компонент связности.

Расстояние между вершинами связного графа – длина кратчайшего простого пути, соединяющего эти вершины. ***Эксцентриситет*** вершины – расстояние от этой вершины до наиболее удаленной от нее.

Диаметр графа – максимальный среди всех эксцентриситетов вершин.

Радиус графа – минимальный среди всех эксцентриситетов вершин.

Центральная вершина – вершина, эксцентриситет которой равен радиусу графа. ***Центр*** графа – множество всех центральных вершин.

Эйлеров цикл – цикл, проходящий через все ребра графа. Граф, который имеет эйлеров цикл, называется ***эйлеровым графом***.

Имеется достаточно простой критерий существования эйлерова цикла и эффективный алгоритм его построения.

Теорема 2.2.3. *Связный граф является эйлеровым тогда и только тогда, когда степени всех его вершин четны.*

Эйлеров цикл в эйлеровом графе G можно получить при помощи ***алгоритма Флёри***. Он основан на следующих двух правилах.

1) Выберем произвольную вершину u графа G, и произвольному ребру (u, v) припишем номер 1. Затем вычеркнем это ребро и перейдем в вершину v.

2) Пусть w − вершина, в которую мы пришли в результате выполнения предыдущего шага, и k − номер, присвоенный некоторому ребру на этом шаге. Выбираем любое ребро, инцидентное вершине w, причем перешеек выбираем только в том случае, когда нет других ребер. Выбранному ребру присваиваем номер $k+1$ и вычеркиваем его.

Алгоритм Флёри заканчивает работу, когда все ребра графа вычеркнуты, т. е. занумерованы. Номер, присвоенный ребру, указывает, каким по счету это ребро проходится в эйлеровом цикле.

Теорема 2.2.4. *Почти каждый граф не является эйлеровым.*

Гамильтонов цикл – простой цикл, проходящий через все вершины графа. Граф, который имеет гамильтонов цикл, называется ***гамильтоновым графом***.

В отличие от задачи распознавания эйлеровости графа, задача распознавания гамильтоновости графа до сих пор не имеет никаких просто проверяемых необходимых и достаточных условий их существования, а все известные алгоритмы включают в себя перебор большого числа вариантов.

Теорема 2.2.5. *Почти все графы гамильтоновы.*

Задачи

2.2.1. Определить число простых путей и простых циклов длины k в графах K_n и $K_{p,q}$.

2.2.2. В графе Q_n найти число простых путей длины n, соединяющих вершины (0,0,...,0) и (1,1,...,1).

2.2.3. Какое наименьшее число ребер может быть в связном графе с n вершинами?

2.2.4. Могут ли графы G и $\overline{G}$ оба быть несвязными?

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

2.2.6. Какое наибольшее число ребер может быть в несвязном графе с n вершинами?

2.2.7. Найти радиус и диаметр каждого из графов $P_n, C_n, Q_n, K_{p,q}$.

2.2.8. Найти радиус, диаметр, центр графа, заданного матрицей смежности:

1) $\begin{pmatrix} 0&1&0&1&0&0&0&1\\ 1&0&1&0&0&0&1&1\\ 0&1&0&1&0&1&0&0\\ 1&0&1&0&1&0&0&0\\ 0&0&0&1&0&0&1&0\\ 0&0&1&0&0&0&1&0\\ 0&1&0&0&1&1&0&1\\ 1&1&0&0&0&0&1&0 \end{pmatrix}$; 2) $\begin{pmatrix} 0&1&1&0&1&0&0&1\\ 1&0&1&0&0&0&1&1\\ 1&1&0&1&1&0&1&1\\ 0&0&1&0&1&0&0&0\\ 1&0&1&1&0&1&1&1\\ 0&0&0&0&1&0&0&1\\ 0&1&1&0&1&0&0&1\\ 1&1&1&0&1&1&1&0 \end{pmatrix}$; 3) $\begin{pmatrix} 0&1&0&0&0&1&1&1\\ 1&0&1&1&1&1&1&0\\ 0&1&0&1&1&1&0&0\\ 0&1&1&0&1&1&0&0\\ 0&1&1&1&0&1&0&0\\ 1&1&1&1&1&0&1&0\\ 1&1&0&0&0&1&0&1\\ 1&0&0&0&0&0&1&0 \end{pmatrix}$.

Определить, является ли граф эйлеровым. В случае положительного ответа, при помощи алгоритма Флёри получить в нем эйлеров цикл.

2.2.9. Построить граф, центр которого:

1) состоит ровно из одной вершины;
2) состоит из трех вершин и не совпадает с множеством всех вершин;
3) совпадает с множеством всех вершин.

2.2.10. Найти радиус, диаметр, центр графа, заданного матрицей смежности:

1) $\begin{pmatrix} 0&0&1&1&0&1&0&0\\ 0&0&1&1&1&0&0&0\\ 1&1&0&0&0&0&0&1\\ 1&1&0&0&0&0&1&0\\ 0&1&0&0&0&0&1&0\\ 1&0&0&0&0&0&0&1\\ 0&0&0&1&1&0&0&0\\ 0&0&1&0&0&1&0&0 \end{pmatrix}$; 2) $\begin{pmatrix} 0&1&1&0&0&0&0&0\\ 1&0&1&1&0&0&1&0\\ 1&1&0&1&0&0&0&0\\ 0&1&1&0&1&0&0&0\\ 0&0&0&1&0&1&1&0\\ 0&0&0&0&1&0&1&1\\ 0&1&0&0&1&1&0&1\\ 0&0&0&0&0&1&1&0 \end{pmatrix}$; 3) $\begin{pmatrix} 0&1&0&0&0&0&0&1\\ 1&0&1&0&0&0&1&0\\ 0&1&0&1&0&1&0&0\\ 0&0&1&0&1&0&0&0\\ 0&0&0&1&0&1&0&1\\ 0&0&1&0&1&0&1&0\\ 0&1&0&0&0&1&0&1\\ 1&0&0&0&1&0&1&0 \end{pmatrix}$.

Определить, является ли граф гамильтоновым. В случае положительного ответа построить гамильтонов цикл графа.

2.2.11. При каких p и q в графе $K_{p,q}$ есть эйлеров цикл? Эйлеров путь? Гамильтонов цикл? При каких n в графе Q_n есть эйлеров цикл?

2.2.12. Доказать, что в графе Q_n при любом $n \geq 2$ имеется гамильтонов цикл.

2.2.13. Найти граф с шестью вершинами, который имеет эйлеров цикл, но не имеет гамильтонова цикла.

2.2.14. Найти граф с шестью вершинами, который имеет гамильтонов цикл, но не имеет эйлерова цикла.

2.3. Деревья

Дерево – связный граф, не имеющий циклов. ***Лес*** – граф, не имеющий циклов. ***Лист*** в дереве – вершина степени 1.

Теорема 2.3.1. *Пусть граф G имеет n вершин и m ребер, тогда следующие условия эквивалентны:*

1) *G − дерево;*
2) *G не содержит циклов и $m = n - 1$;*
3) *G − связный граф и $m = n - 1$;*
4) *G − связный граф, но при удалении любого ребра получаем несвязный граф.*

Корневое дерево − дерево с выделенной вершиной, которая называется корнем дерева.

Между деревьями порядка n ($n \geq 3$) и последовательностями длины $n-2$ из элементов множества $\{1, 2,..., n\}$ можно задать биекцию p, которую назовем соответствием Прюфера.

Пусть T − дерево с n вершинами. Будем считать, что его вершинами являются натуральные числа 1, 2,…, n. Пусть a_1 — наименьший лист в T, а b_1 — смежная с ним вершина. Удалив из T вершину a_1 и ребро $e_1 = (a_1, b_1)$, получим дерево T_1, к которому также применим описанную процедуру. Повторяем ее до тех пор, пока после удаления вершины a_{n-2} и ребра $e_{n-2} = (a_{n-2}, b_{n-2})$ не получим дерево T_{n-2}, состоящее из одного ребра $e_{n-1} = (a_{n-1}, b_{n-1})$. Дереву T ставим в соответствие упорядоченный набор чисел $p(T) = (b_1,..., b_{n-2})$, который называется ***кодом Прюфера.***

Пусть $V = \{1, 2,..., n\}$, $n \geq 3$. Опишем процедуру восстановления по коду Прюфера $p(T) = (b_1,..., b_{n-2})$, где $b_i \in V$ для всех $i = 1,..., n-2$, дерева T, вершинами которого являются элементы множества V. Находим наименьший элемент a_1 множества V, не содержащийся среди элементов последовательности $p(T)$, и восстанавливаем ребро (a_1, b_1) дерева. Далее удаляем a_1 из V и первую компоненту b_1 из последовательности $p(T)$. Продолжаем процедуру для оставшихся чисел, пока не будут удалены все компоненты последовательности $p(T)$. Два оставшихся элемента множества V − есть последнее ребро дерева T.

Теорема 2.3.2 (***Теорема Кэли***). *Число различных деревьев порядка n ($n \geq 3$) равно n^{n-2}.*

Задачи

2.3.1. Перечислить все попарно неизоморфные деревья с числом вершин, не превышающим 6.

2.3.2. Найти два неизоморфных дерева с одинаковыми наборами степеней.

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

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

2.3.5. В дереве с n вершинами степень каждой вершины равна 1 или k. Сколько листьев в таком дереве?

2.3.6. Найти число корневых деревьев с множеством вершин $(1,...,n)$.

2.3.7. Построить код Прюфера для деревьев, изображенных на рис.2.5.

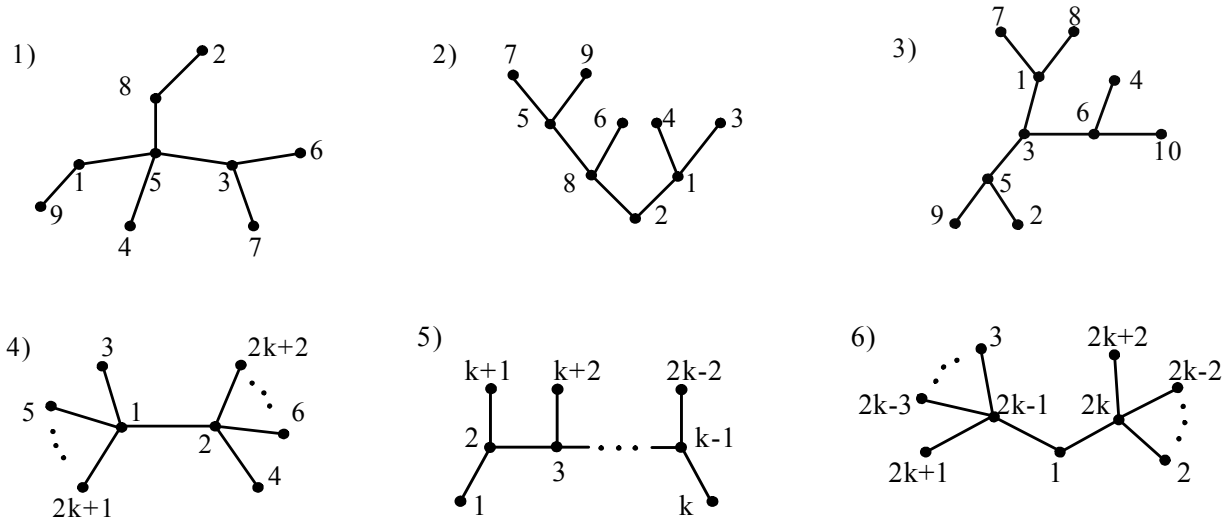


Рис. 2.5

2.3.8. По заданному коду Прюфера $p(T)$ восстановить дерево. Найти центральные вершины восстановленного дерева.

1) $P(T) = (4,1,6,2,2,2,7)$;

2) $P(T) = (4,2,3,4,2,3,1,1)$;

3) $P(T) = (5,2,5,3,5,4,9,6)$;

4) $P(T) = (\underbrace{2,2,...,2}_{n-2})$;

5) $P(T) = (1,2,..., n-2)$;

6) $P(T) = (3,3,4,5,..., n-2, n-2)$.

2.3.9. Найти все графы, которые являются деревьями вместе со своими дополнениями.

2.4. Планарные графы

Плоский граф – граф, вершинами которого являются точки плоскости, а ребрам соответствуют непрерывные линии, соединяющие смежные вершины, причем эти линии пересекаются только в концевых точках, т. е. в вершинах. ***Планарный граф*** – граф, изоморфный плоскому.

Гранью плоского графа называется множество всех таких точек плоскости данного графа, которые можно соединить друг с другом простой кривой, не пересекающей ребра исходного плоского графа.

Пример 2.4.1. Граф, изображенный на рис. 2.6 (1), не является плоским, так как его ребра пересекаются, а граф, изображенный на рис. 2.6 (2), является плоским. Так как графы, изображенные на рис. 2.6, изоморфны, заключаем, что граф K_4 на рис. 2.6 (1) планарный, но не плоский, а граф на рис. 2.6 (2) является и планарным, и плоским.

1 2

Рис. 2.6

Теорема 2.4.1 (*Формула Эйлера*). *Для связного плоского графа* $G = (V, E)$ *с* n *вершинами и* m *ребрами имеет место формула* $n + f = m + 2$, *где* f *− общее число граней.*

Пример 2.4.2. На рис. 2.7 изображен связный плоский граф с четырьмя гранями. Всякий плоский граф имеет одну, и притом единственную, неограниченную грань (на рисунке это грань 4). Такая грань называется ***внешней***, а остальные грани − ***внутренними***. Получаем для графа, что $n = 4,\ f = 4,\ m = 6$, и формула Эйлера для него имеет место.

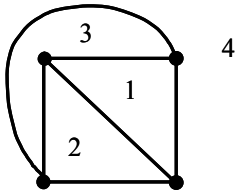
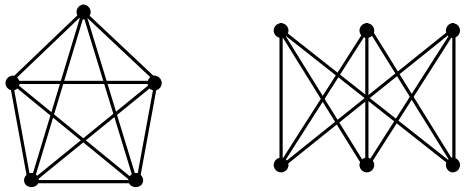


Рис. 2.7

Следствие 2.4.1. *Во всяком связном планарном графе с* n $(n \geq 3)$ *вершинами и* m *ребрами имеет место неравенство* $m \leq 3 \cdot (n-2)$.

Следствие 2.4.2. *Во всяком связном планарном графе с* n $(n \geq 3)$ *вершинами и* m *ребрами, не содержащем циклов длины три, имеет место неравенство* $m \leq 2 \cdot (n-2)$.

Пример 2.4.3. Полный граф K_5 (см. рис. 2.8) имеет пять вершин и десять ребер, поэтому для него не выполняется неравенство из следствия 2.4.1, значит, K_5 не является планарным графом.



K_5 $K_{3,3}$

Рис. 2.8

Пример 2.4.4. Полный двудольный граф $K_{3,3}$ (см. рис. 2.8) имеет шесть вершин и девять ребер. Он не содержит циклов нечетной длины и для него не выполняется неравенство из следствия 2.4.2, значит, $K_{3,3}$ не является планарным графом.

Следствие 2.4.3. *Во всяком связном планарном графе с* n $(n \geq 3)$ *вершинами и* m *ребрами, не содержащем циклов длины меньше* k $(k \geq 3)$*, имеет место неравенство* $m \leq \dfrac{k}{k-2}(n-2)$.

Операция подразбиения ребра (a,b) в графе $G=(V,E)$ состоит в удалении ребра (a,b) и добавлении двух новых ребер $(a,c),(c,b)$, где c − новая вершина. Графы G и G' называются ***гомеоморфными***, если оба они могут быть получены из одного и того же графа подразбиением его ребер.

Теорема 2.4.2 (*Критерий планарности Понтрягина-Куратовского*). *Граф планарен тогда и только тогда , когда он не содержит подграфов, гомеоморфных графам* K_5 *или* $K_{3,3}$.

Операция стягивания ребра (a,b) в графе $G=(V,E)$ состоит в отождествлении (слиянии) смежных вершин a и b. Граф G называется ***стягиваемым к графу*** G', если G' получается из G в результате некоторой последовательности стягиваний ребер.

Теорема 2.4.3 (*Критерий планарности Вагнера*). *Граф планарен тогда и только тогда, когда он не содержит подграфов, стягиваемых к графам* K_5 *или* $K_{3,3}$.

Пример 2.4.5. Пользуясь критерием Понтрягина-Куратовского, покажем, что граф Петерсена, изображенный на рис. 2.9 (1), не является планарным. Удалив из него ребра (7,10) и (3,4), замечаем, что получившийся подграф (рис. 2.9 (2)) гомеоморфен графу $K_{3,3}$ (рис. 2.9 (3)).

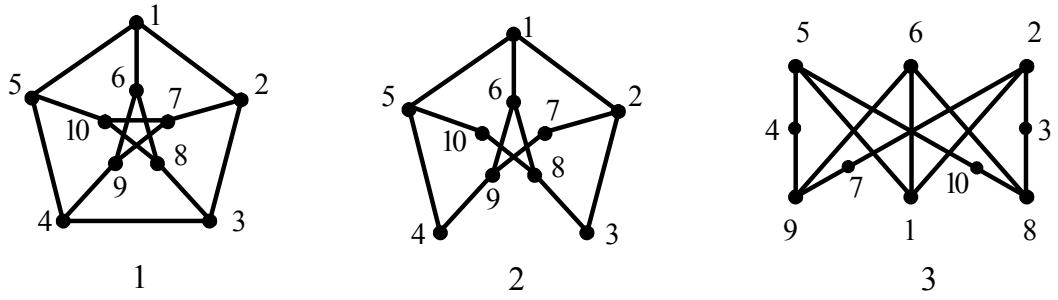


Рис. 2.9

Применим теперь к графу Петерсена критерий Вагнера. Стянем ребра (1,6), (2,7), (3,8), (4,9) и (5,10), тогда граф Петерсена очевидным образом стягивается к K_5, и, значит, не является планарным.

Можно доказать непланарность графа Петерсена с помощью следствия 2.4.3 из теоремы Эйлера. Заметим, что граф Петерсена не содержит циклов длины три и четыре, т. е. он не содержит циклов длины меньше $k = 5$. В нем $n = 10$ и $m = 15$. Подставляя имеющиеся данные в неравенство $m \le \frac{k}{k-2}(n-2)$, заключаем, что оно не выполняется.

Задачи

2.4.1. Какие из графов, изображенных на рис. 2.10, планарны?

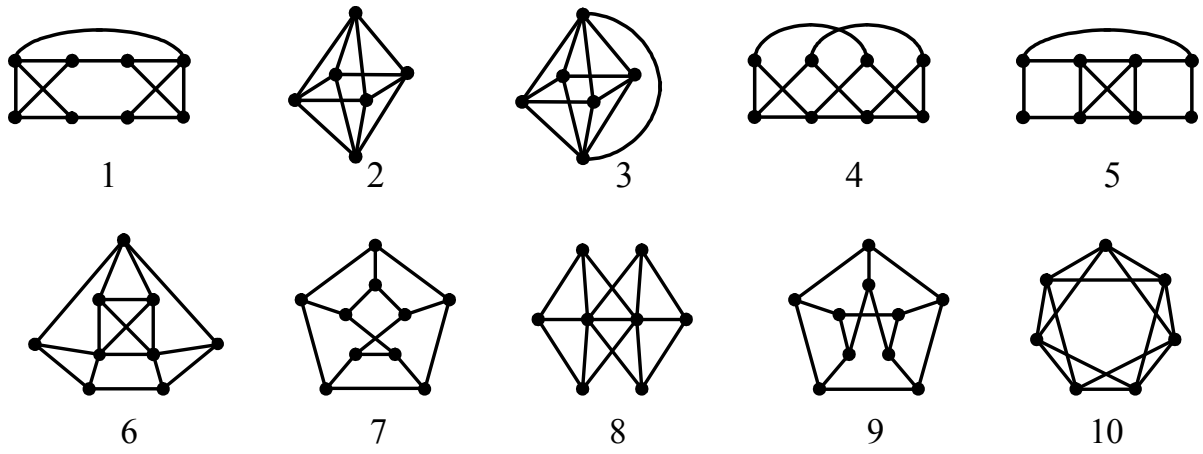


Рис. 2.10

2.4.2. Какое наименьшее количество ребер нужно удалить из графа G, чтобы получить планарный граф, если:

1) $G = K_6$;
2) $G = Q_4$;
3) G - граф Петерсена (рис. 2.9 (1))?

2.4.3. Построить граф с 6 вершинами и 12 ребрами, который содержит одновременно подграфы, гомеоморфные K_5 и $K_{3,3}$.

2.4.4. Выяснить, существует ли планарный граф, у которого:

1) 7 вершин и 16 ребер;
2) 8 вершин и 17 ребер.

2.4.5. Какое наибольшее число граней может быть у плоского пятивершинного графа, не имеющего петель и кратных ребер? Изобразить этот граф.

2.4.6. При каких n граф Q_n планарен?

2.5. Паросочетания

Паросочетание в графе − множество его ребер, в котором каждая пара ребер не имеет общей вершины.

Паросочетание называется ***максимальным***, если оно не содержится в паросочетании с большим числом ребер.

Паросочетание называется ***наибольшим***, если оно содержит наибольшее число ребер среди всех паросочетаний. Число ребер в наибольшем паросочетании графа G называется ***числом паросотания*** и обозначается через $\alpha_1(G)$.

Пример 2.5.1. На рис. 2.11 показан двудольный граф G. Множество $M_1 = \{(v_1,2), (v_2,1), (v_3,3), (v_4,5), (v_5,4)\}$ является в графе G и максимальным, и наибольшим паросочетанием, поэтому $\alpha_1(G)=5$.

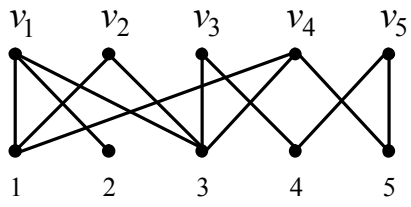


Рис. 2.11

Множество $M_2 = \{(v_1,1), (v_2,3), (v_3,3), (v_5,5)\}$ паросочетанием не является, так как в нем два ребра имеют общую вершину.

Множество $M_3 = \{(v_1,1), (v_2,3), (v_3,4), (v_5,5)\}$

является максимальным паросочетанием, но не является наибольшим.

Множество $M_4 = \{(v_1,1),(v_2,3),(v_5,5)\}$ является паросочетанием, но не является ни максимальным (оно содержится в паросочетании M_3), ни наибольшим.

Ребра паросочетания называются ***сильными***, остальные ребра − ***слабыми***.

Вершина графа называется ***свободной***, если она не принадлежит ребру паросочетания.

Чередующейся цепью относительно данного паросочетания называется простой путь, в котором чередуются сильные и слабые ребра. Чередующаяся цепь называется ***увеличивающей***, если она соединяет две свободные вершины.

Теорема 2.5.1. *Паросочетание является наибольшим тогда и только тогда, когда относительно него нет увеличивающих цепей.*

Дерево достижимости − дерево с корнем, в котором каждый путь, начинающийся в корне, является чередующимся.

Пусть $G = (V_1, V_2, E)$ − двудольный граф с долями V_1 и V_2, M − паросочетание в G. Выберем некоторую свободную вершину $a \in V_1$. Опишем процедуру построения дерева достижимости из этой вершины. Вершина x графа G называется ***четной*** или ***нечетной*** в зависимости от того, четно или нечетно расстояние между нею и вершиной a. Так как граф G двудольный, то любой путь, соединяющий вершину a с четной (нечетной) вершиной, имеет четную (нечетную) длину. Значит, в чередующемся пути, ведущем из вершины a в четную (нечетную) вершину, последнее ребро обязательно сильное (слабое).

При построении дерева достижимости используем модифицированный поиск в ширину из вершины a. Он состоит в том, что для четных вершин исследуются инцидентные им слабые ребра, а для нечетных − сильные. Если очередная рассматриваемая вершина x оказывается свободной, нет необходимости доводить построение дерева до конца. В этом случае путь между вершинами a и x в дереве является увеличивающим путем и можно его использовать для построения большего паросочетания.

Найденный увеличивающий путь начинается и заканчивается в свободных вершинах, а вдоль пути чередуются сильные и слабые ребра. Если на этом пути превратить каждое сильное ребро в слабое, а каждое слабое − в сильное, то получится новое паросочетание, в котором на одно ребро больше. Увеличение паросочетания с помощью описанной процедуры лежит в основе ***метода чередующихся (увеличивающих) цепей.***

Пример 2.5.2. Применим алгоритм чередующихся цепей для построения наибольшего паросочетания в двудольном графе, изображенном на рис. 2.12, слева. Выберем некоторое максимальное паросочетание в этом графе, например, $M = \{(v_1,1), (v_3,3), (v_5,5)\}$. Вершина v_2 является свободной относительно паросочетания M.

На рис. 2.12, справа, изображено дерево достижимости с корнем v_2. В этом дереве из корня исходят три чередующиеся цепи. Две из них, заканчивающиеся свободной вершиной 2 или 4, представляют собой увеличивающий путь. Рассмотрим увеличивающий путь: $(v_2,3), (3,v_3), (v_3,4)$. Используем его для увеличения паросочетания M путем удаления из него сильного ребра $(v_3,3)$ и добавления двух слабых ребер $(v_2,3), (v_3,4)$.

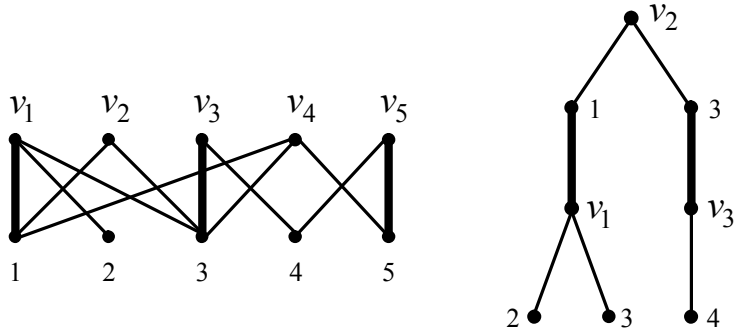


Рис. 2.12

Получили паросочетание $M' = \{(v_1,1), (v_2,3), (v_3,4), (v_5,5)\}$, которое изображено на рис. 2.13, слева. Вершина v_4 является свободной относительно паросочетания M'. На рис. 2.13, справа, изображено дерево достижимости с корнем v_4, в котором цепь $(v_4,1), (1,v_1), (v_1,2)$ является увеличивающей. Удалим сильное ребро $(v_1,1)$ из паросочетания и присоединим к нему пару слабых ребер $(v_4,1), (v_1,2)$. Получили паросочетание $M'' = \{(v_1,2), (v_2,3), (v_3,4), (v_4,1), (v_5,5)\}$, изображенное на рис. 2.14. Заметим, что в графе G относительно найденного паросочетания M'' нет свободных вершин.

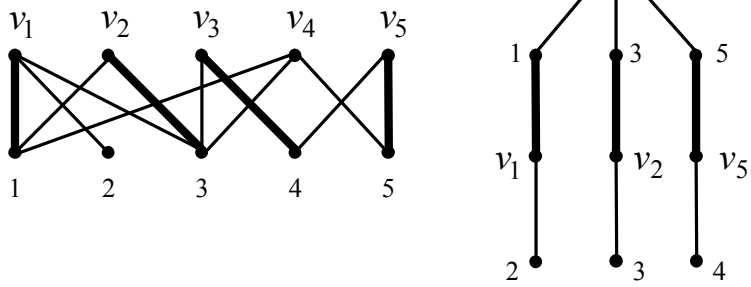


Рис. 2.13

Значит, построенное паросочетание M'' является наибольшим паросочетанием двудольного графа G.

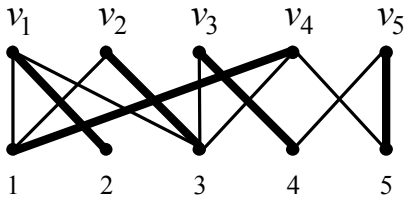


Рис. 2.14

Задачи

2.5.1. Задан двудольный граф $G=(V_1 \cup V_2, E)$, где $V_1=\{v_1,v_2,v_3,v_4,v_5,v_6,v_7\}$ и $V_2=\{1,2,3,4,5,6,7\}$, S – список смежных вершин в графе. Используя алгоритм чередующихся цепей, увеличить паросочетание π до наибольшего паросочетания:

1) S: $v_1:\{1,3,4\}, v_2:\{1,2,5\}, v_3:\{2,4,5\}, v_4:\{1,6\}, v_5:\{3,5,7\}, v_6:\{6,7\}, v_7:\{4,5\}$;
   $\pi=\{(v_1,1),(v_2,2),\ (v_3,4),(v_5,5),(v_6,6)\}$;
2) S: $v_1:\{1,2,3,4\}, v_2:\{1,3\}, v_3:\{2,6\}, v_4:\{3,4,6\}, v_5:\{3,7\}, v_6:\{6,7\}, v_7:\{4,5,7\}$;
   $\pi=\{(v_1,1),(v_2,3),\ (v_3,6),(v_4,4),(v_7,7)\}$;
3) S: $v_1:\{1,2\}, v_2:\{2,5,6\}, v_3:\{5\}, v_4:\{1,2,5\}, v_5:\{3,4,5,6\}, v_6:\{4,5,7\}, v_7:\{5,6\}$;
   $\pi=\{(v_1,1),(v_2,2),\ (v_3,5),(v_5,3),(v_6,7),(v_7,6)\}$;
4) S: $v_1:\{2,3,5\}, v_2:\{1,4,5\}, v_3:\{1,2,3,6,7\}, v_4:\{4,6\}, v_5:\{4,7\}, v_6:\{4,7\}, v_7:\{6,7\}$;
   $\pi=\{(v_1,2),(v_2,4),\ (v_3,3),(v_4,6),(v_6,7)\}$;
5) S: $v_1:\{1,2\}, v_2:\{1,2,4\}, v_3:\{1,3,4,5\}, v_4:\{2,5\}, v_5:\{3,4,5,7\}, v_6:\{5,6,7\}, v_7:\{6,7\}$;
   $\pi=\{(v_1,1),(v_2,2),\ (v_3,3),(v_5,5),(v_7,7)\}$.

2.5.2. Для двудольных графов, изображенных на рис. 2.15, найти наибольшее паросочетание, применяя метод чередующихся цепей.

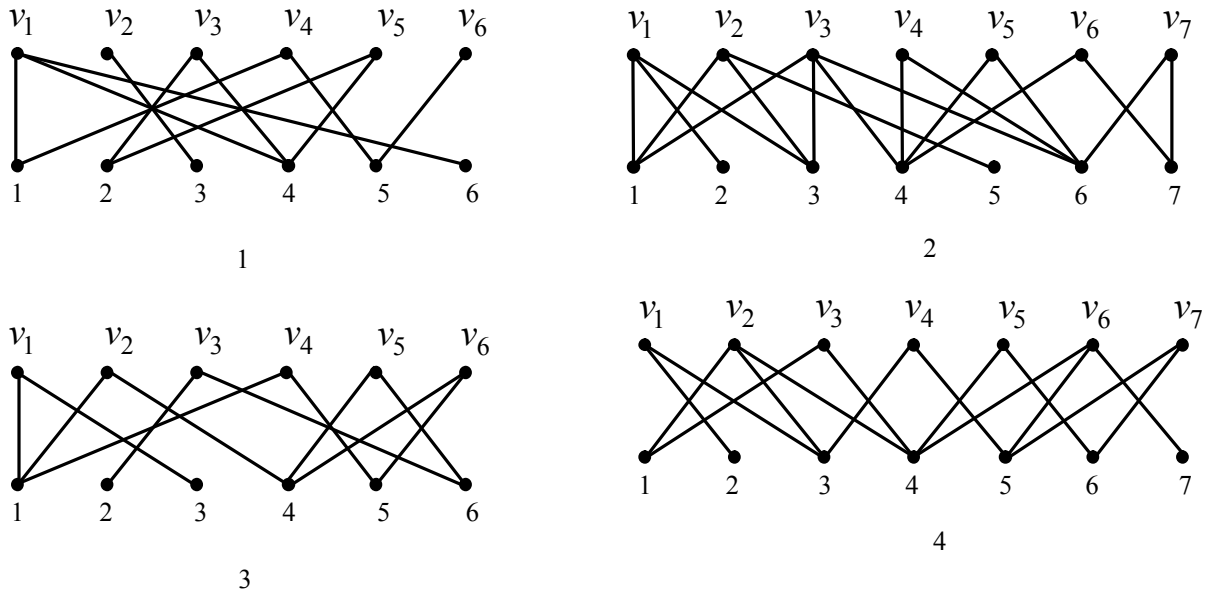


Рис. 2.15

2.6. Независимые множества. Клики графа

Множество вершин графа называется ***независимым***, если никакие две вершины из этого множества не смежны.

Независимое множество называется ***максимальным***, если оно не является собственным подмножеством другого независимого множества. Независимое множество называется ***наибольшим***, если оно содержит наибольшее число вершин. Число вершин в наибольшем независимом множестве графа G называется ***числом независимости*** графа G и обозначается $\alpha(G)$.

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

Рассмотрим алгоритм для нахождения наибольшего независимого множества графа G, в основе которого лежит метод поиска вглубину. Выберем в графе произвольную вершину a.

Пусть G_1 – подграф, получающийся удалением из графа G вершины a, а G_2 – подграф, получающийся удалением из графа G всех вершин, смежных с вершиной a.

Пусть X – какое-нибудь независимое множество графа G. Если оно не содержит вершину a, то оно является независимым множеством графа G_1. Если же $a \in X$, то никакая вершина, смежная с a, не принадлежит X. В этом случае множество X является независимым множеством графа G_2. 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 задачи можно рассматривать как исчерпывающий поиск в дереве подзадач, т.е. подграфов. Каждой вершине этого дерева соответствует

некоторый граф, сыновьям вершины – два подграфа этого графа, корню – исходный граф. Листьям дерева соответствуют подграфы, не имеющие ребер, то есть подграфы, у которых все вершины изолированные. Множества вершин этих подграфов − это независимые множества исходного графа.

Пример 2.6.1. На рис. 2.16 представлен процесс нахождения наибольшего независимого множества для графа, изображенного в корне дерева поиска.

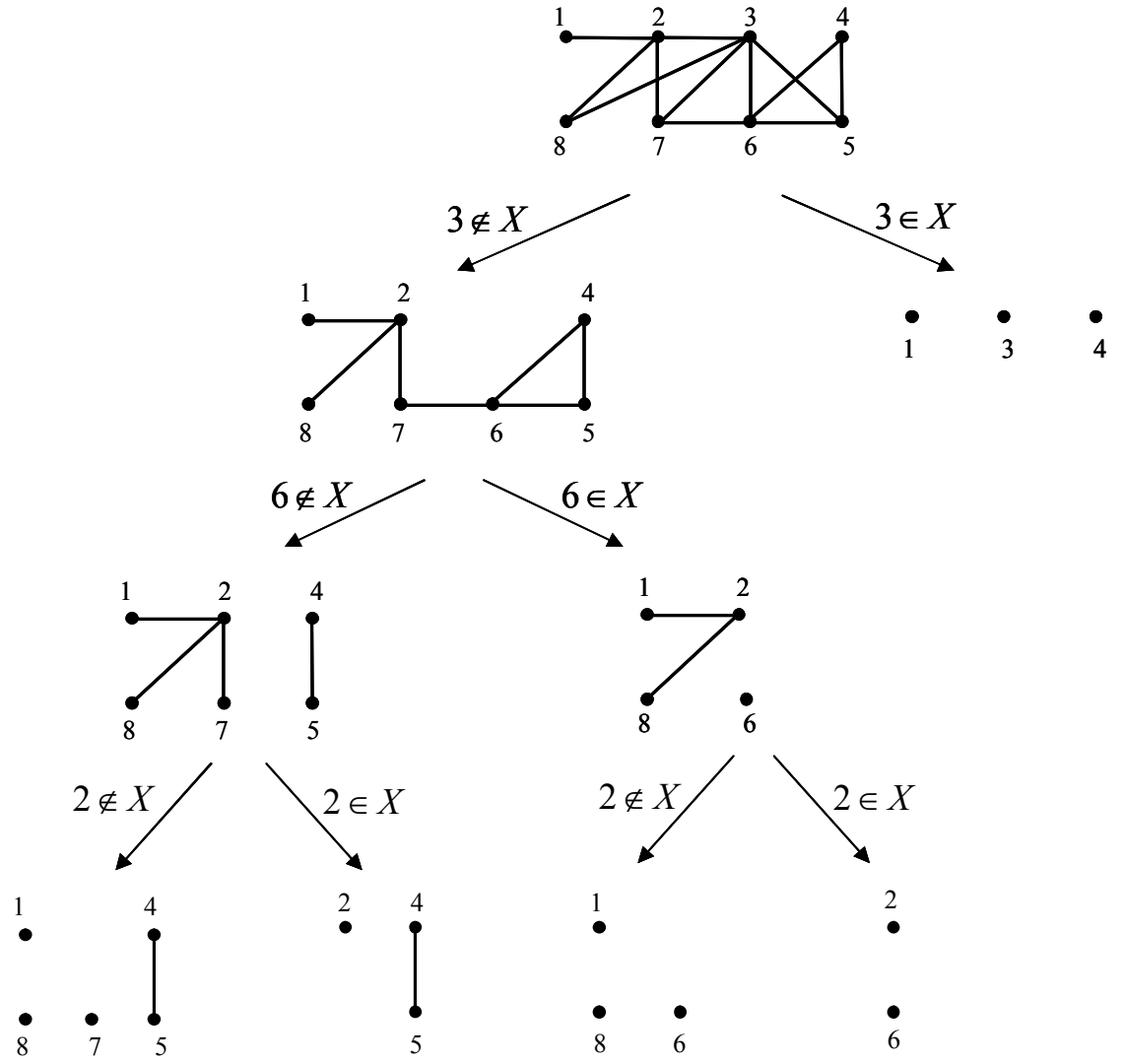


Рис. 2.16

Для нахождения наибольшего независимого множества не обязательно строить все дерево полностью. Получив граф, для которого задача нахождения наибольшего независимого множества может быть решена без дальнейшего разбиения на подграфы, можно переходить к поиску следующего листа. На рис. 2.16, просматривая все листья построенного дерева поиска, находим наибольшее независимое множество графа: {1, 4, 7, 8} или {1, 5, 7, 8}. Число независимости графа $\alpha(G) = 4$.

Задачи

2.6.1. Найти наибольшее независимое множество вершин графов, заданных матрицей смежности:

1) $\begin{pmatrix} 0&0&0&0&0&0&1&1\\ 0&0&1&0&1&0&1&0\\ 0&1&0&1&1&1&1&0\\ 0&0&1&0&0&0&0&0\\ 0&1&1&0&0&1&0&0\\ 0&0&1&0&1&0&1&0\\ 1&1&1&0&0&1&0&1\\ 1&0&0&0&0&0&1&0 \end{pmatrix}$; 2) $\begin{pmatrix} 0&1&0&0&0&1&0&1\\ 1&0&0&0&0&1&0&1\\ 0&0&0&1&0&0&0&1\\ 0&0&1&0&1&1&1&0\\ 0&0&0&1&0&1&0&0\\ 1&1&0&1&1&0&0&0\\ 0&0&0&1&0&0&0&1\\ 1&1&1&0&0&0&1&0 \end{pmatrix}$; 3) $\begin{pmatrix} 0&0&0&0&0&0&1&0\\ 0&0&1&0&0&0&1&0\\ 0&1&0&1&1&1&0&1\\ 0&0&1&0&0&0&1&0\\ 0&0&1&0&0&1&0&0\\ 0&0&1&0&1&0&1&0\\ 1&1&0&1&0&1&0&1\\ 0&0&1&0&0&0&1&0 \end{pmatrix}$.

2.6.2. Найти наибольшее независимое множество вершин графа Петерсена (рис.2.9(1)).

2.6.3. Найти число независимости графа G:

1) $G = K_n$; 2) $G = K_n - (v_1, v_2)$; 3) $G = P_n$; 4) $G = C_n$;

5) $G = K_{p,q}$.

2.7. Раскраски

Раскраской вершин (ребер) графа называется сопоставление цветов вершинам (ребрам) графа. Раскраска называется ***правильной***, если смежные вершины (ребра) окрашены в разные цвета.

Хроматическое число графа G − наименьшее число цветов, для которого существует правильная раскраска вершин графа G, обозначается как $\chi(G)$.

Хроматический индекс графа G − наименьшее число цветов, для которого существует правильная раскраска ребер графа G, обозначается как $\chi'(G)$.

Теорема 2.7.1. Для любого графа G верно неравенство $\chi(G) \geq \omega(G)$, где $\omega(G)$ − кликовое число графа G.

Обозначим через $s(G)$ наибольшую из степеней вершин графа G.

Теорема 2.7.2. Для любого графа G верно неравенство $\chi(G) \leq 1 + s(G)$.

Теорема 2.7.3. Для любого графа G справедливы неравенства $s(G) \leq \chi'(G) \leq s(G) + 1$.

Рассмотрим алгоритм раскраски графа, основанный на построении дерева поиска. Пусть x и y − две несмежные вершины графа G.

Пусть G_1 − граф, получающийся добавлением ребра (x, y) к графу G. Если в правильной раскраске графа G вершины x и y имеют разные цвета, то она будет правильной и для графа G_1.

Пусть G_2 − граф, получающийся из G слиянием вершин x и y. Если в правильной раскраске графа G цвета вершин x и y одинаковы, то эта раскраска будет правильной и для графа G_2.

Обратно, раскраска каждого из графов G_1, G_2 даст раскраску графа G в то же число цветов. Таким образом,

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

Что дает возможность рекурсивного нахождения раскраски графа в минимальное число цветов. Эта рекурсия окончательно приводит к полным графам, для которых задача о раскраске решается тривиально.

Пример 2.7.1. Применим описанный алгоритм к решению задачи о раскраске для графа C_5. На рис. 2.17 показан фрагмент дерева поиска правильных раскра-

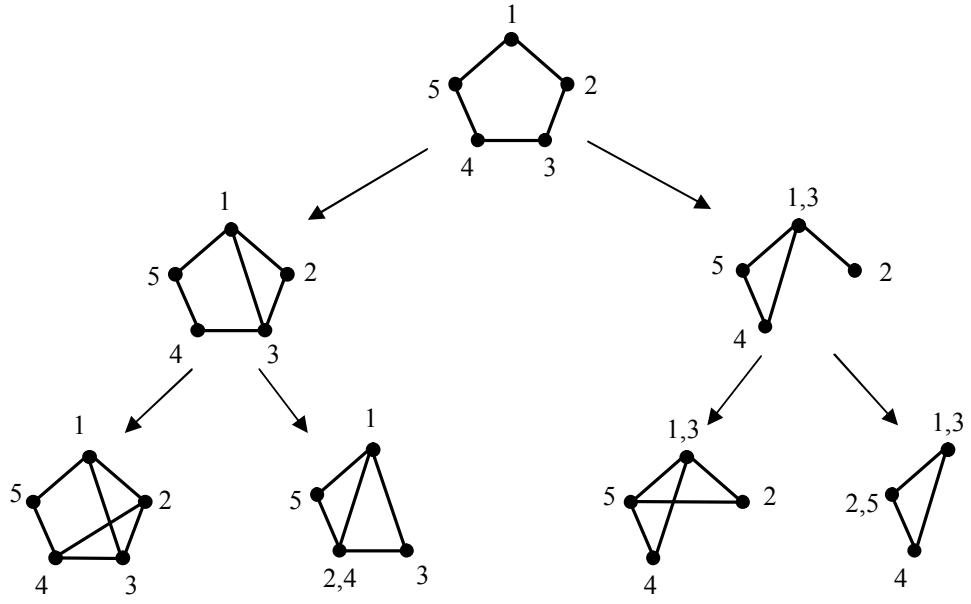


Рис. 2.17

сок графа C_5. Рассматривая подграфы, расположенные в листьях этого дерева, заключаем, что наименьшее число цветов, необходимое для раскраски C_5 равно трем. Полный граф из трех вершин, полученный в результате работы данно-

го алгоритма дает правильную раскраску C_5: вершины 1,3 красим в цвет 1, вершины 2,5 − в цвет 2, вершину 4 − в цвет 3. Получаем, $\chi(C_5)=3$.

Для произвольного графа G определим ***реберный граф*** $L(G)$. Множеством вершин графа $L(G)$ является множество ребер G, вершины x и y смежны в $L(G)$ тогда и только тогда, когда ребра x и y смежны в G.

Поскольку ребра любого графа G смежны тогда, когда смежны соответствующие вершины реберного графа $L(G)$, то $\chi'(G)$ можно определить как хроматическое число графа $L(G)$, то есть $\chi'(G)=\chi(L(G))$.

Исходя из теоремы 2.7.3, получаем, что в классе реберных графов хроматическое число может принимать только два значения − $s(G)$ и $s(G)+1$. Несмотря на это, определение величины $\chi'(G)$ является весьма трудной задачей.

Задачи

2.7.1. Найти хроматическое число и хроматический индекс графа G:
1) $G=K_n$; 2) $G=K_n-(v_1,v_2)$; 3) $G=P_n$; 4) $G=C_n$;
5) $G=K_{p,q}$.

2.7.2. Найти хроматическое число и хроматический индекс графа Петерсена (см. рис. 2.9(1)).

2.7.3. Найти хроматическое число и хроматический индекс графов, изображенных на рис. 2.17.

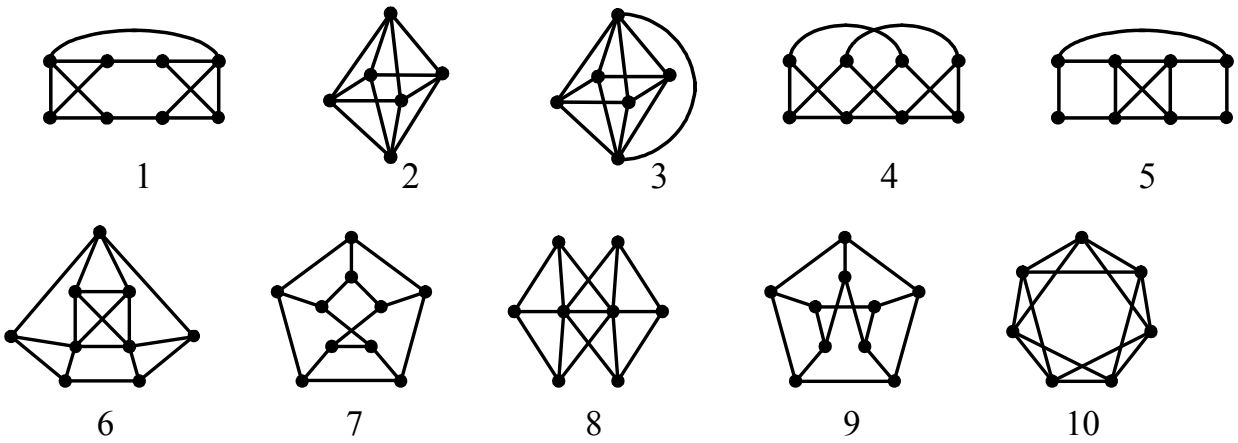


Рис. 2.17

2.8. Задача о кратчайших путях

Взвешенный граф − граф, ребрам которого приписаны некоторые (обычно неотрицательные) числа (веса).

Задача о кратчайших путях рассматривается для взвешенных ориентированных графов, дугам которых приписаны неотрицательные веса.

Всякому неориентированному графу можно сопоставить орграф, который получается из данного заменой каждого неориентированного ребра (i, j) парой ориентированных ребер (i, j) и (j, i) с весом, равным весу исходного неориентированного ребра.

Если граф не взвешен, можно считать, что все ребра имеют один и тот же вес.

В задаче рассматриваются следующие вопросы:

1) Какова длина кратчайшего пути между двумя выделенными вершинами? Что это за путь?

2) Каковы длины кратчайших путей между выделенной вершиной и всеми остальными вершинами графа? Каковы эти пути?

3) Каковы длины путей между всеми парами вершин? Что это за пути?

Обозначим через w_{ij} вес дуги (i, j). Несуществующие дуги будем считать дугами с бесконечным весом.

Весом (***длиной***) ***пути*** называется сумма весов дуг в этом пути.

Для задачи о кратчайших путях не требуется выполнение условия $w_{ij} = w_{ji}$ и выполнение неравенства треугольника $w_{ik} + w_{kj} \geq w_{ij}$.

Первая задача, нахождение длины кратчайшего пути между двумя вершинами s (начало) и f (конец) и отыскание самого пути, может быть решена с помощью алгоритма Дейкстры, в основе которого лежит метод поиска в ширину.

Алгоритм Дейкстры.

В процессе работы алгоритма вершинам графа присваиваются метки: временные или постоянные.

Вначале вершине s присваивается постоянная метка 0 (нулевое расстояние до самой себя), а каждой из остальных вершин приписывается временная метка ∞.

На каждом шаге одной вершине с временной меткой приписывается постоянная метка и поиск продолжается дальше. Метки меняются следующим образом:

1) Каждой вершине j, имеющей временную метку, присваивается новая временная метка – наименьшая из ее временной метки и чисел w_{ij} + окончательная метка вершины i, где i – вершина, которой присвоена постоянная метка на предыдущем шаге.

2) Находится наименьшая из всех временных меток, которая становится постоянной меткой самой вершины. В случае равенства выбирается любая из них.

Процесс продолжается до тех пор, пока вершина f не получит постоянную метку.

Первой вершиной с постоянной меткой будет s, которая находится на расстоянии 0 от себя. Следующей, получившей постоянную метку, будет вершина, ближайшая к s. Третьей вершиной, получившей постоянную метку, будет вторая по близости к s, и т.д. Постоянная метка каждой вершины – это кратчайшее расстояние до этой вершины от начальной вершины s.

Сам путь можно вычислить, используя в процессе работы алгоритма указатель *prev(v)* на вершину с постоянной меткой, с помощью которой произошло последнее изменение значения метки для v. Когда метка вершины v становится постоянной, *prev(v)* указывает на вершину, предшествующую вершине v в кратчайшем пути. Вершины, входящие в кратчайший путь, имеют следующий вид: *s, … , pre(pre(pre(f))), pre(pre(f)), pre(f), f.*

Пример 2.8.1. Рассмотрим взвешенный ориентированный граф, изображенный на рис. 2.18. Требуется найти кратчайшее расстояние между вершинами b и g.

Справа от рисунка изображена матрица расстояний графа.

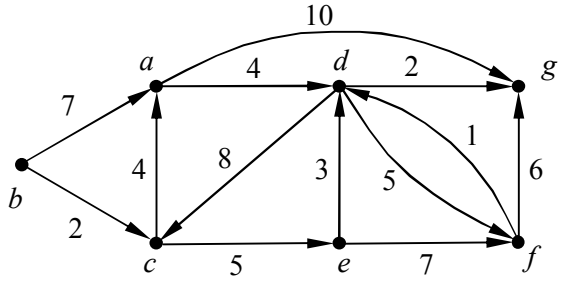


Рис. 2.18

	a	***b***	***c***	***d***	***e***	***f***	***g***
a	0	∞	∞	4	∞	∞	10
b	7	0	2	∞	∞	∞	∞
c	4	∞	0	∞	5	∞	∞
d	∞	∞	8	0	∞	5	2
e	∞	∞	∞	∞	0	7	∞
f	∞	∞	∞	1	∞	0	6
g	∞	∞	∞	∞	∞	∞	0

Работу алгоритма по шагам проиллюстрируем с помощью следующей таблицы:

a	*b*	*c*	*d*	*e*	*f*	*g*	
∞	**0**	∞	∞	∞	∞	∞	Начальная вершина *b* помечена 0.
7		2	∞	∞	∞	∞	Пересчитываются метки вершин, связанных с вершиной *b* дугой.
7		**2**	∞	∞	∞	∞	Наименьшая метка (метка вершины *c*) становится постоянной.
6			∞	7	∞	∞	Пересчитываются метки вершин, связанных с вершиной *c* дугой.
6			∞	7	∞	∞	Наименьшая метка (метка вершины *a*) становится постоянной.
			10	7	∞	16	Пересчитываются метки вершин, связанных с вершиной *a* дугой.
			10	**7**	∞	16	Наименьшая метка (метка вершины *e*) становится постоянной.
			10		14	16	Пересчитываются метки вершин, связанных с вершиной *e* дугой.
			10		14	16	Наименьшая метка (метка вершины *d*) становится постоянной.
					14	12	Пересчитываются метки вершин, связанных с вершиной *d* дугой.
					14	**12**	Вершина *g* получает постоянную метку, и построение таблицы заканчивается.

Кратчайшее расстояние от вершины *s* до вершины *g* равно 12.

Найдем кратчайший путь от *s* до *g*. Для этого просмотрим столбец для *g* снизу вверх и найдем вершину с постоянной меткой, при пересчете по которой метка вершины *g* получила значение 12. Это вершина *d*. Следовательно, *pre(g)=d*. Далее просматриваем столбец для вершины *d* и находим *pre(d)=a*. Аналогично находим *pre(a)=c* и *pre(c)=b*. Процесс нахождения предшествующей вершины пути продолжается до тех пор, пока не дойдем до начальной вершины. Таким образом, мы получили следующий кратчайший путь от *s* до *g*: $b \to c \to a \to d \to g$. Его длина равна 2 + 4 + 4 + 2 = 12.

Временная сложность задачи нахождения кратчайшего пути равна $O(n^2)$, где *n* – число вершин в графе.

Для нахождения кратчайших путей от заданной вершины до всех остальных вершин достаточно достроить таблицу до конца, пока все вершины графа не

получат постоянные метки. После этого по таблице для каждой вершины строится кратчайший путь от начальной вершины. Временная сложность этой задачи также равна $O(n^2)$.

Для того, чтобы найти для каждой пары вершин кратчайший путь, достаточно решить n предыдущих задач (n – число вершин в графе), взяв в качестве начальной каждую из вершин графа. Очевидно, временная сложность задачи равна $O(n^3)$.

Задачи

2.8.1. Взвешенный ориентированный граф задан матрицей расстояний:

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	0	3	∞	∞	2	7	∞
2	∞	0	5	∞	1	∞	8
3	∞	∞	0	1	∞	∞	∞
4	∞	3	∞	0	∞	2	∞
5	∞	∞	∞	2	0	1	4
6	∞	∞	5	∞	∞	0	9
7	2	∞	∞	5	1	∞	0

Используя алгоритм Дейкстры, по матрице расстояний найти:

1) кратчайшее расстояние и кратчайший путь между вершинами 1 и 3, между вершинами 3 и 7;

2) кратчайшие расстояния и кратчайшие пути от вершин 2 и 5 до всех остальных вершин.

2.8.2. Взвешенный неориентированный граф задан матрицей расстояний:

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	0	3	∞	∞	2	7	∞
2	3	0	5	∞	1	∞	8
3	∞	5	0	1	∞	5	∞
4	∞	∞	1	0	2	∞	3
5	2	1	∞	2	0	1	4
6	7	∞	5	∞	1	0	9
7	∞	8	∞	3	4	9	0

Используя алгоритм Дейкстры, по матрице расстояний найти:

1) кратчайшее расстояние и кратчайший путь между вершинами 4 и 3, между вершинами 3 и 6;

2) кратчайшие расстояния и кратчайшие пути от вершин 2 и 5 до всех остальных вершин.

2.8.3. Обыкновенный граф задан матрицей смежности M:

1) $\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$; 2) $\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$; 3) $\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$.

Используя алгоритм Дейкстры, по матрице M построить матрицу расстояний R, в которой элемент r_{ij} равен расстоянию между вершинами i и j.

Глава 3. ТЕОРИЯ КОДИРОВАНИЯ

3.1. Основные понятия теории кодирования

Пусть $B = \{b_1, b_2, \ldots, b_m\}$ – алфавит. Конечная последовательность символов $\beta = b_{i_1} b_{i_2} \ldots b_{i_n}$ называется ***словом***, а число n – ***длиной слова*** (длину слова β будем обозначать через $|\beta|$). Через B^* обозначается множество всех слов в алфавите B. Слова из B^* будем называть также ***сообщениями***.

Под ***двоичным кодированием*** понимается инъективное отображение $f : L \to \{0,1\}^+$, где $\{0,1\}^+$ – множество всех непустых двоичных последовательностей. Далее под кодированием будем понимать двоичное кодирование.

Отображение f ставит в соответствие слову $\beta \in B^*$ слово $\alpha \in \{0,1\}^+$, α называется ***кодом сообщения*** β.

Требование инъективности отображения f означает, что различные сообщения должны кодироваться разными двоичными последовательностями. При выполнении этого требования обеспечивается однозначность декодирования сообщений. Такое кодирование будем называть ***взаимно-однозначным.***

Пример 3.1.1. Рассмотрим схему алфавитного кодирования

$$f_1 : \begin{cases} b_1 \to 0, \\ b_2 \to 01. \end{cases}$$

Схема f_1 задает взаимно-однозначное кодирование. Достаточно заметить, что перед каждым вхождением символа 1 стоит символ 0, поэтому каждое вхождение 1 вместе с предшествующим нулем кодирует букву b_2. Символ 0, за которым не следует 1, кодирует букву b_1. Например, последовательность $\beta = 001010001$ имеет единственную расшифровку $\alpha = b_1 b_2 b_2 b_1 b_1 b_2$.

Пример 3.1.2. Рассмотрим схему

$$f_2 : \begin{cases} b_1 \to 0, \\ b_2 \to 01, \\ b_3 \to 001. \end{cases}$$

Кодирование, задаваемое схемой f_2, не является взаимно-однозначным. Последовательность $\beta = 001$ допускает две расшифровки $\alpha_1 = b_1 b_2$ и $\alpha_2 = b_3$.

В теории кодирования рассматриваются не любые инъективные отображения *f*, а те из них, которые могут быть реализованы алгоритмами.

Наиболее простым способом кодирования является побуквенное, или алфавитное, кодирование.

Алфавитное кодирование задается схемой, в которой каждой букве алфавита ставится в соответствие двоичная последовательность символов:

$$f:\begin{cases} b_1 \to v_1, \\ b_2 \to v_2, \\ \quad \cdots \\ b_m \to v_m, \end{cases} \qquad v_i \in \{0,1\}^* \text{ для всех } i = \overline{1,m}.$$

Коды v_i называются ***элементарными***, а их набор $V = (v_1, v_2, \ldots v_m)$ – ***кодом***.

Через l_i будем обозначать длину элементарного кода буквы b_i. Набор длин элементарных кодов $L = (l_1, l_2, \ldots l_m)$ называется ***спектром кода***.

3.2. Проблема распознавания взаимной однозначности кодирования

Пусть слово α имеет вид $\alpha_1\alpha_2$. Тогда α_1 называется ***префиксом*** слова α, а α_2 – ***суффиксом*** α. Если $0< |\alpha_1| <|\alpha|$, то α_1 называется ***собственным префиксом*** α, если $0< |\alpha_2| <|\alpha|$, то α_2 – ***собственный суффикс*** α (здесь через $|x|$ обозначается длина слова *x*).

Схема алфавитного кодирования f обладает ***свойством префикса***, если для любых i и j $(1 \le i, j \le m, i \ne j)$ слово v_i не является префиксом слова v_j.

Алфавитное кодирование, схема которого обладает свойством префикса, называется ***префиксным***.

Теорема 3.2.1. *Если схема обладает свойством префикса, то алфавитное кодирование является взаимно-однозначным.*

Таким образом, свойство префикса является достаточным условием взаимной однозначности.

Теорема 3.2.2. *Если алфавитное кодирование со схемой f обладает свойством взаимной однозначности, то длины элементарных кодов $l_i = |v_i| (i = \overline{1,m})$ удовлетворяют неравенству Мак-Миллана:* $\sum_{i=1}^{m} 2^{-l_i} \le 1$.

Неравенство Мак-Миллана является необходимым условием взаимной однозначности кода со схемой f, но не достаточным.

Для схемы f_2, рассмотренной в примере 3.1.2, имеем $l_1 = 1,\ l_2 = 2,\ l_3 = 3$, и неравенство Мак-Миллана выполняется: $2^{-1} + 2^{-2} + 2^{-3} = 7/8 < 1$. Однако задаваемое схемой f_2, кодирование не является взаимно-однозначным.

Теорема 3.2.3. *Если числа $l_1, l_2, \ldots l_m$ удовлетворяют неравенству Мак-Миллана, то существует алфавитное кодирование, обладающее свойством префикса и удовлетворяющее равенствам: $|v_1| = l_1, |v_2| = l_2, \ldots, |v_m| = l_m$.*

Следствие 3.2.1. *Если существует взаимно-однозначное алфавитное кодирование с заданными длинами элементарных кодов, то существует также префиксное кодирование с теми же длинами элементарных кодов.*

Неравенство Мак-Миллана в силу справедливости теоремы 3.2.3 можно рассматривать и как достаточное условие взаимной однозначности, в том смысле, что если неравенство для некоторой схемы кодирования выполняется, можно заменить эту схему схемой со свойством префикса с теми же длинами элементарных кодов.

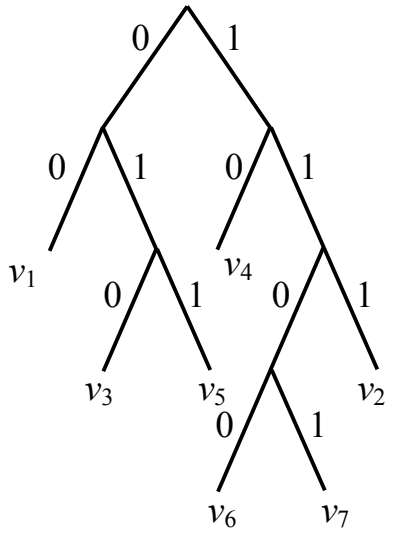
Пусть задан префиксный код $V = (v_1, v_2, \ldots v_m)$. Элементарные коды определяют ***бинарное кодовое дерево***. Из каждой вершины дерева выходит не более двух ребер в вершины следующего яруса, левое из которых помечается символом 0, а правое – символом 1. Элементарным кодам соответствуют вершины дерева, определяемые путем, идущим из корня. Если код префиксный, элементарные коды расположены в листьях дерева.

Дерево называется ***насыщенным***, если из каждой вершины, не являющейся листом, в следующий ярус выходит ровно два ребра.

Пример 3.2.1. Рассмотрим схему алфавитного кодирования f_3.

$$f_3 : \begin{cases} b_1 \to 00, \\ b_2 \to 111, \\ b_3 \to 010, \\ b_4 \to 10, \\ b_5 \to 011, \\ b_6 \to 1100, \\ b_7 \to 1101. \end{cases}$$

Заметим, что схема f_3 обладает свойством префикса. На рис. 3.1 изображено кодовое дерево этого префиксного кода.



Для непосредственной проверки взаимной однозначности необходимо в общем случае проверить бесконечное множество слов.

Проблема распознавания взаимной однозначности алфавитного кодирования решена Ал.А.Марковым (1963 г.). Алгоритм распознавания состоит в следующем.

Для кода $V=(v_1,v_2,\ldots,v_m)$ пусть S_1 − множество слов, обладающих следующим свойством. Слово β является собственным префиксом некоторого элементарного кода v_i и одновременно собственным суффиксом некоторого v_j. Положим $S=S_1\cup\{\lambda\}$ (λ - пустое слово).

Сопоставим коду *V* ориентированный граф *G*, вершинами которого являются элементы множества *S*. Вершины α и β соединяем ориентированным ребром (α,β), если существует элементарный код v_j и последовательность элементарных кодов $P=v_{i_1}v_{i_2}\ldots v_{i_k}$, такие, что $v_j=\alpha\, v_{i_1}v_{i_2}\ldots v_{i_k}\beta$. При этом *P* может быть пустой, если α и β оба непустые. Представление v_j в виде $\alpha v_{i_1}v_{i_2}\ldots v_{i_k}\beta$ будем называть разложением элементарного кода v_j.

Ребру (α,β) припишем последовательность $v_{i_1}v_{i_2}\ldots v_{i_k}$. Ребро ($\lambda$, λ) присутствует в графе тогда и только тогда, когда существует v_j и последовательность $P=v_{i_1}v_{i_2}\ldots v_{i_k}$ ($k\ge 2$), такие, что $v_j=v_{i_1}v_{i_2}\ldots v_{i_k}$.

Теорема 3.2.4. *Алфавитный код V является взаимно-однозначным тогда и только тогда, когда в графе G отсутствуют ориентированные циклы, проходящие через вершину λ.*

Пример 3.2.2. Пусть $B=(b_1,b_2,b_3)$, $V=\{1,010,101\}$. Следует выяснить, является ли код V взаимно-однозначным. Если код не взаимно-однозначный, указать пару слов, которые кодируются одинаково.

Построим множества S_1 и S: S_1={ 0, 1, 01, 10}, S={ 0, 1, 01, 10, λ}.

Выпишем все нетривиальные разложения элементарных кодов v_i.

Для v_1 нет нетривиальных разложений.

$v_2 = 010 = 0\ v_1\ 0 = 01\ \lambda\ 0 = 0\ \lambda\ 10$,

$v_3 = 101 = \lambda\ v_1\ 01 = 10\ v_1\ \lambda = 1\ \lambda\ 01 = 10\ \lambda\ 1$.

Соответствующий коду граф изображен на рис. 3.2. Граф содержит ориентированный цикл, проходящий через вершину λ, следовательно, код V не является взаимно-однозначным.

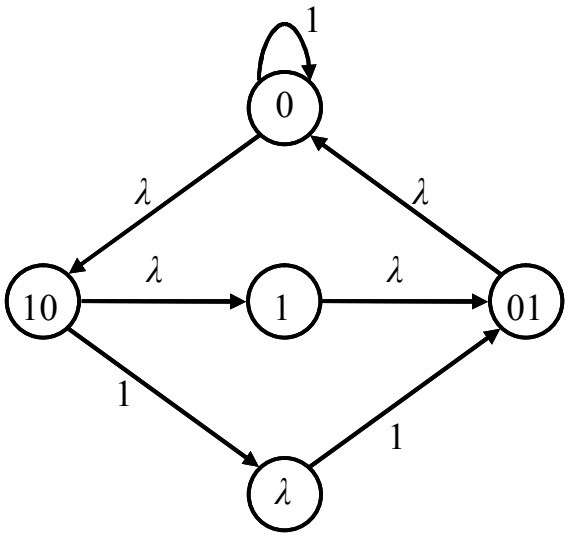


Рис. 3.2

По графу нетрудно построить двоичную последовательность, допускающую две расшифровки. Для этого достаточно, начиная с вершины λ, приписать друг к другу двоичные последовательности, соответствующие вершинам и ребрам графа, вдоль найденного цикла.

Слово γ=1010101, соответствующее циклу, допускает две расшифровки: $b_1b_2b_3$ и $b_3b_2b_1$.

Пример 3.2.3. Пусть $B=(b_1,b_2,b_3,b_4,b_5)$, $V=\{1,01,100,0100,0000\}$. Выяснить, является ли код V взаимно-однозначным. Если код не взаимно-однозначный, указать пару слов, которые кодируются одинаково.

Построим множества S_1 и S: S_1={ 0, 00, 000, 1}, S={ 0, 00, 000, 1, λ}.

Выпишем все нетривиальные разложения для элементарных кодов.

$v_2 = 01 = 0\ v_1\ \lambda = 0\ \lambda\ 1$,

$v_3 = 100 = \lambda\, v_1\, 00 = 1\, \lambda\, 00$,

$v_4 = 0100 = 0\, v_1\, 00 = \lambda\, v_2\, 00 = 0\, v_3\, \lambda$,

$v_5 = 0000 = 0\, \lambda\, 000 = 00\, \lambda\, 00 = 000\, \lambda\, 0$.

Соответствующий коду граф изображен на рис. 3.3. Граф не содержит ориентированный цикл, проходящий через вершину λ, следовательно, код V является взаимно-однозначным.

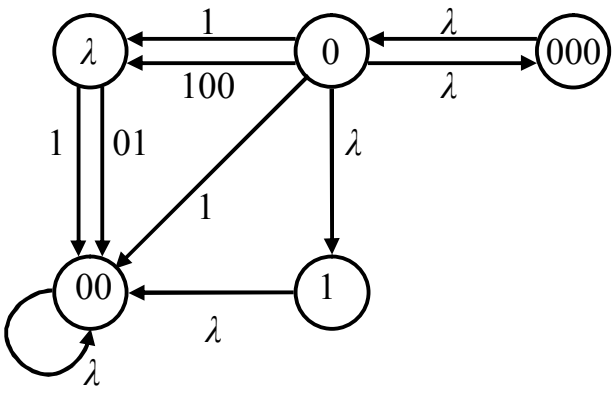


Рис. 3.3

Задачи

3.2.1. Выяснить, обладает ли код V свойством префикса:

1) $V = \{0, 10, 11, 1110\}$;
2) $V = \{01, 11, 10, 001\}$;
3) $V = \{1001, 000, 001, 0110, 010, 1000, 0111\}$;
4) $V = \{110, 011, 1011, 0100, 11011\}$.

3.2.2. Выяснить, является ли код V взаимно-однозначным. Если код не взаимно-однозначный, указать пару слов, которые кодируются одинаково:

1) $V = \{1, 100, 0001, 010, 0010\}$;
2) $V = \{01, 100, 010, 1001\}$;
3) $V = \{1,01, 10, 100, 001, 010\}$;
4) $V = \{10, 010, 0, 1001\}$;
5) $V = \{10, 010, 0101, 101\}$;
6) $V = \{10, 01, 100, 0100, 0000\}$;
7) $V = \{01, 1, 01010, 010, 1010100\}$;

8) $V = \{110, 011, 001, 0110, 110, 0\}$;

9) $V = \{0, 01, 0010001001\}$;

10) $V = \{0, 101010, 01010101\}$.

3.2.3. Доказать, что код не является взаимно-однозначным:

1) $V = \{01, 0, (10)^m 1\}$;

2) $V = \{0^i, 0^j, 0^k\}$, НОД$(i, j, k) = 1$;

3) $V = \{(01)^i 0, (10)^j 1, (01)^k\}$, НОД$(1, i+j+1, k) = 1$;

4) $V = \{0, 0^i 1, 1\varphi(0, 0^i 1)\}$, $\varphi(x,y) \in \{x, y\}^*$.

3.2.4. Для кода V найти слово минимальной длины, декодируемое неоднозначно:

1) $V = \{0, (10)^{k+1}, (01)^k\}$;

2) $V = \{010, 101, 01010, (01)^k\}$, $k = 3s+1$;

3) $V = \{0, (10)^k, (01)^m\}$;

4) $V = \{001, 011, 100, 110, (1100)^k\}$, $k=3s$;

5) $V = \{0, 10, 11, (101)^k\}$;

6) $V = \{01, 10, 11, (110)^k\}$, $k = 2s$;

7) $V = \{(01)^k 0, 0(10)^{k+1}, 1(01)^m\}$;

8) $V = \{0, 0^k 10^m, (10^m)^2 1\}$;

9) $V = \{(01)^k, (10)^{k+1}, (01)^k 0\}$;

10) $V = \{(01)^k, (10)^{k+1} 0, (01)^{k+2}\}$.

3.2.5. Выбрать максимальное по числу элементов подмножество B множества A с условием, что двоичные разложения наименьшей длины чисел из B представляют собой

а) префиксный код; б) взаимно-однозначный код:

1) $A = \{1, 5, 6, 7, 12, 13, 17\}$;

2) $A = \{1, 3, 6, 8, 10, 13, 19, 33, 37\}$;

3) $A = \{2, 6, 7, 9, 12, 15, 18, 35, 36, 37\}$;

4) $A = \{2, 3, 7, 8, 11, 12, 13, 14\}$;

5) $A = \{1, 2, 5, 8, 9, 10, 13, 15\}$;

6) $A = \{3, 5, 6, 9, 10, 13, 17\}$;

7) $A = \{1, 2, 5, 8, 9, 12, 13, 14\}$;

8) $A = \{5, 6, 7, 8, 9, 10, 11, 12, 13\}$;

9) $A = \{4, 6, 7, 10, 13, 15, 20, 23, 25\}$;

10) $A = \{5, 7, 9, 10, 12, 14, 17, 23, 24\}$.

3.3. Построение префиксного кода по набору длин элементарных кодов

Пусть задан набор натуральных чисел $l_1, l_2, \ldots l_m$, удовлетворяющих неравенству Мак-Миллана: $\sum_{i=1}^{m} 2^{-l_i} \le 1$. В силу теоремы 3.2.3 существует префиксный код с набором длин $(l_1, l_2, \ldots l_m)$ элементарных кодов.

Рассмотрим алгоритм К.Шеннона построения префиксного кода по набору длин.

Будем полагать $l_1 \le l_2 \le \ldots \le l_m$. Построим последовательность чисел $q_1, q_2, \ldots, q_m$, которая строится по следующим правилам:

$$q_1 = 0,$$
$$q_{i+1} = q_i + 2^{-l_i} (i = 1,2,\ldots,m-1).$$

Очевидно, $0 \le q_i < 1$ и q_i имеет единственное представление в виде двоичной дроби с l_i знаками после запятой: $q_i = \sum_{j=1}^{l_i} c_j^{(i)} \cdot 2^{-j}$, где $c_j^{(i)}$=0 или $c_j^{(i)}$=1.

Рассмотрим код $V = (v_1, v_2, \ldots, v_m)$, где $v_i = c_1^{(i)} c_2^{(i)} \ldots c_{l_i}^{(i)}$.

Так как наборы длин упорядочены по неубыванию, при $h > i$ выполняются неравенства $l_h \ge l_i$ и $q_h \ge q_i + 2^{-l_i}$. Поэтому элементарный код v_h отличается от элементарного кода v_i в l_i первых разрядах. Следовательно, построенный код является префиксным.

Пример 3.3.1. Рассмотрим набор чисел L=(2,3,3,3,4,4,4), удовлетворяющий неравенству Мак-Миллана $2^{-2}+2^{-3}+2^{-3}+2^{-3}+2^{-4}+2^{-4}+2^{-4}=\frac{13}{16}<1$.

Построим последовательность чисел $q_1, q_2, q_3, q_4, q_5, q_6, q_7$, записывая их в двоичной системе счисления.

$$q_1 = 0{,}00;$$
$$q_2 = 0 + 2^{-2} = 0{,}010;$$
$$q_3 = 0{,}01 + 2^{-3} = 0{,}01 + 0{,}001 = 0{,}011;$$
$$q_4 = 0{,}011 + 2^{-3} = 0{,}011 + 0{,}001 = 0{,}100;$$
$$q_5 = 0{,}1 + 2^{-3} = 0{,}1 + 0{,}001 = 0{,}1010;$$
$$q_6 = 0{,}101 + 2^{-4} = 0{,}101 + 0{,}0001 = 0{,}1011;$$
$$q_7 = 0{,}1011 + 2^{-4} = 0{,}1011 + 0{,}0001 = 0{,}1100.$$

Далее строим схему f алфавитного кодирования, выбирая в качестве элементарного кода v_i последовательность из 0 и 1 длины l_i, образующую дробную часть числа q_i (при этом недостающие разряды в записи дробной части дополняем справа нулями):

$$f: \begin{cases} v_1 = 00 \\ v_2 = 010 \\ v_3 = 011 \\ v_4 = 100 \\ v_5 = 1010 \\ v_6 = 1011 \\ v_7 = 1100. \end{cases}$$

Нетрудно убедиться в том, что построенный код является префиксным.

Задачи

3.3.1. С помощью неравенства Мак-Миллана выяснить, может ли набор чисел L быть набором длин элементарных кодовых слов взаимно-однозначного кода:

1) L = {1, 2, 2, 3}; 2) L = {2, 2, 2, 4, 4, 4}; 3) L = {1, 2, 4, 4, 4, 4};

4) L = {2, 2, 3, 4, 4}; 5) L = {1, 2, 3, 4, 4, 5}.

3.3.2. Построить префиксный код с заданной последовательностью длин элементарных кодов:

1) L = {1, 2, 3, 3}; 2) L = {1, 2, 4, 4, 4, 4};
3) L = {2, 2, 3, 3, 4, 4, 4, 4}; 4) L = {2, 2, 2, 4, 4, 4};
5) L = {2, 2, 3, 4, 4}; 6) L = {2, 3, 3, 3, 3, 4, 4};
7) L = {1, 3, 3, 4, 4, 4, 4}; 8) L = {1, 2, 3, 4, 5, 6, 6};
9) L = {2, 2, 2, 4, 4, 5}; 10) L = {2, 3, 4, 5, 5, 5, 5}.

3.4. Алгоритмы экономного алфавитного кодирования

При построении экономных кодов используется дополнительная информация о вероятностях (частотах) появления букв в сообщениях.

Пусть на буквах алфавита $B = \{b_1, b_2, \ldots, b_m\}$ задано распределение вероятностей $P = \{p_1, p_2, \ldots, p_m\},\ p_i > 0,\ \sum_{i=1}^{m} p_i = 1.$

Под ***стоимостью кодирования f*** понимается величина $C_f(P) = \sum_{i=1}^{m} p_i \,|\, v_i \,|$

(Здесь $|\, v_i \,|$ − длина элементарного кода буквы b_i).

Стоимость кодирования определяет число двоичных разрядов, которые тратятся на кодирование одной буквы.

$C_f(P)$ − это ***средняя длина элементарного кода***, которая показывает, во сколько раз увеличивается средняя длина слова при кодировании f.

Пример 3.4.1. Пусть $B = \{b_1, b_2, b_3, b_4\},\ P = \{0{,}40;\ 0{,}25;\ 0{,}20;\ 0{,}15\}.$ Рассмотрим две схемы кодирования : $f_1 : \begin{cases} b_1 \to 00, \\ b_1 \to 01, \\ b_1 \to 10, \\ b_1 \to 11, \end{cases}$ $f_2 : \begin{cases} b_1 \to 1, \\ b_1 \to 01, \\ b_1 \to 000, \\ b_1 \to 001. \end{cases}$

Для f_1 стоимость кодирования $C_{f_1}(P) = 2,$ для f_2 стоимость кодирования $C_{f_2}(P) = 1{,}95.$ Таким образом, стоимость кодирования может изменяться при переходе от одной схемы кодирования к другой.

Положим $C^*(P) = \inf_V C_V(P)$. Код V^* такой, что $C_{V^*}(P) = C^*(P)$, называется ***оптимальным*** для набора частот P. Можно показать, что величина $C^*(P)$ достигается при некоторой схеме f^* и может быть определена как $\min_V C_V(P)$. Оптимальные коды дают в среднем минимальное увеличение длин слов при соответствующем кодировании. В силу теоремы 3.2.3 при построении оптимальных кодов можно ограничиться рассмотрением префиксных кодов.

Рассмотрим алгоритмы построения оптимальных и близких к оптимальным кодов.

Алгоритм Хаффмана (1952 г.)

Алгоритм Хаффмана строит оптимальный префиксный код. При рассмотрении алгоритмов кодирования наряду с элементарными кодами вершинам кодового дерева будем приписывать вероятности соответствующих букв.

Кодовое дерево, соответствующее оптимальному префиксному коду, называется ***оптимальным.***

Алгоритм Хаффмана основан на следующих свойствах оптимальных кодов.

Лемма 3.4.1. *Если код* $V = (v_1, v_2, \ldots, v_m)$ - *оптимальный для* $P = (p_1, p_2, \ldots, p_m)$, *то* $|v_i| \le |v_j|$ *при* $p_i > p_j$.

Из леммы 3.4.1 следует, что в оптимальном дереве вероятности букв, приписанные вершинам k-го яруса, не меньше вероятностей, приписанных вершинам $(k+1)$-го яруса.

Лемма 3.4.2. *Оптимальному префиксному коду соответствует насыщенное кодовое дерево.*

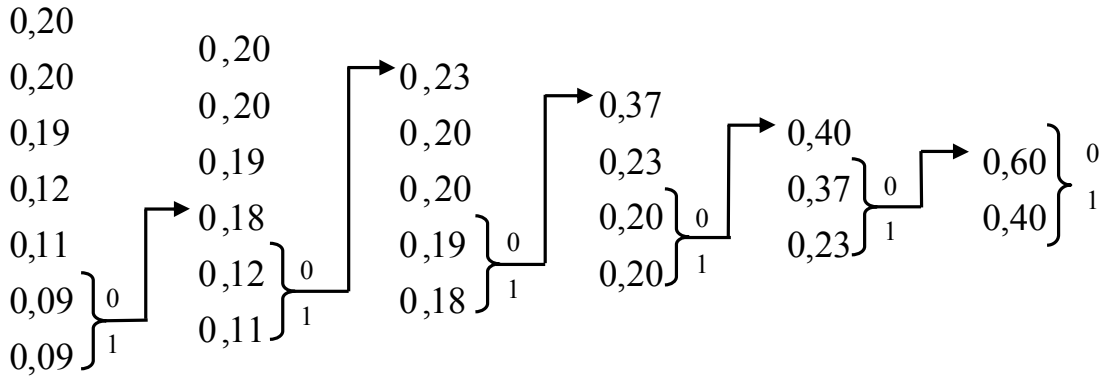
Лемма 3.4.3. *Две самые маленькие вероятности в оптимальном кодовом дереве находятся на нижнем ярусе. Перестановкой элементарных кодов нижнего яруса их можно поставить в вершины, для которых инцидентные им ребра выходят из одной вершины.*

Теорема 3.4.1 (*теорема редукции*). *Если код с длинами* $(l_1, l_2, \ldots, l_{m-1}, l_m)$ *является оптимальным для распределения вероятностей* $P = (p_1, p_2, \ldots, p_{m-1}, p_m)$, *то код с длинами* $(l_1, l_2, \ldots, l_{m-1} - 1)$ *также будет оптимальным для распределения вероятностей* $P' = (p_1, p_2, \ldots, p_{m-1} + p_m)$.

Теорема редукции позволяет свести задачу построения оптимального кода мощности m к задаче построения оптимального кода мощности m-1.

Алгоритм Хаффмана заключается в следующем. Пусть вероятности в распределении $P=(p_1,p_2,\ldots,p_m)$ расположены в порядке невозрастания. На каждом шаге объединяются две буквы, имеющие наименьшие вероятности. Вместо этих двух букв вводится новая буква с вероятностью $p=p_{m-1}+p_m$. Вероятность p вставляется в оставшийся набор вероятностей так, чтобы в получившемся новом наборе вероятности остались в порядке невозрастания. Продолжаем процесс объединения вероятностей до тех пор, пока не останутся две буквы алфавита. Одной из них приписывается символ 0, другой – символ 1 (оптимальный код для двух букв при произвольном распределении вероятностей). Затем из оптимального кода для двух букв строится оптимальный код для трех букв, и т.д. Продолжая этот процесс, придем к искомому оптимальному коду для m букв.

Пример 3.4.2. Пусть $B=\{b_1,b_2,b_3,b_4,b_5,b_6,b_7\}$, P = (0,20; 0,20; 0,19; 0,12; 0,11; 0,09; 0,09). Процесс построения оптимального кода можно представить следующим образом:



Фигурными скобками отмечены объединяемые вероятности. Для каждой скобки верхней вероятности приписываем символ 0, нижней – символ 1. Затем осуществляем движение в обратном направлении к $p_1,p_2,\ldots,p_7$ и, проходя скобки, выписываем соответствующие элементарные коды.

Таким образом, оптимальный код задается схемой $f:\begin{cases} b_1\to 10 \\ b_2\to 11 \\ b_3\to 000 \\ b_4\to 010 \\ b_5\to 011 \\ b_6\to 0010 \\ b_7\to 0011. \end{cases}$

Например, путь 0,60 – 0,23 – 0,11 дает элементарный код для буквы b_5. Стоимость оптимального кодирования $C^*(P) = 2{,}78$.

Алгоритм Фано (1961 г.)

Упорядоченный в порядке невозрастания вероятностей список букв делится на две последовательные части так, чтобы суммы вероятностей входящих в них букв как можно меньше отличались друг от друга. Буквам из первой части приписываем символ 0, а буквам из второй части – символ 1. Далее точно так же поступаем с каждой из полученных частей, если она содержит хотя бы две буквы. Построенный код является префиксным, и ему соответствует насыщенное кодовое дерево.

В алгоритме Фано кодовое дерево строится от корня, а в алгоритме Хаффмана – начиная с листьев. Это отличие позволяет в алгоритме Хаффмана полнее использовать специфику данного распределения вероятностей и строить оптимальный код. Алгоритм Фано строит код, близкий к оптимальному.

Пример 3.4.3. Пусть $B = \{b_1, b_2, b_3, b_4, b_5, b_6, b_7\}$, P=(0,20; 0,20; 0,19; 0,12; 0,11; 0,09;0,09). При помощи алгоритма Фано построим код, близкий к оптимальному, для заданного распределения вероятностей.

0,59	0,20	0		0,20	0	{0,20}	
	0,20		0,39	0,20	1	{0,20}	0
	0,19			0,19		{0,19}	1
0,41	0,12	1	0,23	0,12	0	{0,12}	0
	0,11			0,11		{0,11}	1
	0,09		0,18	0,09	1	{0,09}	0
	0,09			0,09		{0,09}	1

Получаем следующую схему алфавитного кодирования: $f: \begin{cases} b_1 \to 00 \\ b_2 \to 010 \\ b_3 \to 011 \\ b_4 \to 100 \\ b_5 \to 101 \\ b_6 \to 110 \\ b_7 \to 111. \end{cases}$

Стоимость кодирования $C_{\Phi}(P) = 2{,}80$.

Алгоритм Шеннона (1948 г.)

Алгоритм Шеннона применим в случае, когда все вероятности $p_i > 0$. Букве b_i ставится в соответствие последовательность из $l_i = \left\lceil \log \frac{1}{p_i} \right\rceil$ двоичных символов. Алгоритм Шеннона основан на том, что выбранные длины l_i $(i = \overline{1,m})$ удовлетворяют неравенству Мак-Миллана. После выбора длин применяется алгоритм Шеннона построения схемы кодирования по заданному набору длин элементарных кодов, описанный ранее.

Пример 3.4.4. Пусть $B = \{b_1, b_2, b_3, b_4, b_5, b_6, b_7\}$, P=(0,20; 0,20; 0,19; 0,12; 0,11; 0,09; 0,09). Вычислим набор длин для *P*.

$$l_1 = l_2 = \left\lceil \frac{1}{0{,}20} \right\rceil = 3, \quad l_3 = \left\lceil \frac{1}{0{,}19} \right\rceil = 3, \quad l_4 = \left\lceil \frac{1}{0{,}12} \right\rceil = 4, \quad l_5 = \left\lceil \frac{1}{0{,}11} \right\rceil = 4,$$

$$l_6 = l_7 = \left\lceil \frac{1}{0{,}09} \right\rceil = 4.$$

Построим префиксный код по алгоритму Шеннона с вычисленными длинами элементарных кодов.

$$f: \begin{cases} b_1 \to 000 \\ b_2 \to 001 \\ b_3 \to 011 \\ b_4 \to 1001 \\ b_5 \to 1011 \\ b_6 \to 1101 \\ b_7 \to 1110. \end{cases}$$

Стоимость кодирования $C_{Ш}(P) = 3{,}41$.

Задачи

3.4.1. Для заданного распределения вероятностей построить двоичные коды, используя алгоритмы Хаффмана, Фано и Шеннона. Сравнить их эффективность:

1) P = (0,2; 0,4; 0,2; 0,2);

2) P = (0,1; 0,1; 0,1; 0,7);

3) P = (0,2; 0,2; 0,2; 0,2; 0,2);

4) P = (0,08; 0,03; 0,09; 0,1; 0,5; 0,2);

5) P = (0,3; 0,4; 0,06; 0,08; 0,04; 0,04; 0,04; 0,04);

6) P = (0,3; 0,3; 0,03; 0,03; 0,03; 0,03; 0,03; 0,01; 0,2; 0,04);

7) P = (0,06; 0,06; 0,06; 0,06; 0,06; 0,3; 0,2; 0,1; 0,1);

8) P = (0,1; 0,2; 0,4; 0,05; 0,05; 0,05; 0,05; 0,05; 0,05);

9) P = (0,25; 0,15; 0,05; 0,1; 0,2; 0,1; 0,05);

10) P = (0,11; 0,08; 0,01; 015; 0,25; 0,21; 0,09; 0,1).

3.4.2. Выяснить, является ли код оптимальным для распределения вероятностей P = (0,15; 0,25; 0,05; 0,01; 0,09; 0,25; 0,15; 0,05):

1) V = (001; 010; 10; 11; 101; 011; 0110; 00);

2) V = (0000, 001; 1001; 10010; 0111; 111, 0010, 1110);

3) V = (000; 001; 010; 011; 100; 101; 110; 111);

4) V = (100; 00; 11110; 11111; 1110; 01; 101; 110);

5) V = (000; 01; 1110; 11111; 110; 10; 001; 11110);

6) V = (000; 10; 0111; 01111; 011; 01; 100; 11111).

3.4.3. Указать набор вероятностей P, при котором существует двоичный префиксный код с заданным набором длин элементарных кодов, являющийся оптимальным:

1) L = (1, 2, 3, 4, 5, 5);

2) L = (2, 2, 2, 3, 3);

3) L = (2, 2, 3, 3, 4, 4, 4, 4);

4) L = (1, 2, 4, 4, 4, 4).

3.4.4. Для префиксного кода V выяснить, является ли код оптимальным для заданного распределения вероятностей P:

1) V = (00, 01, 10; 110; 111); P = (1/4; 1/4 ; 1/4; 1/8; 1/8);

2) V= (0, 10, 110; 1110; 1110; 1111); P = (1/2; 1/4; 1/8; 1/16; 1/16);

3) V= (0, 100, 101; 110; 1110; 1111); P = (1/6; 1/6; 1/6; 1/6; 1/6; 1/6).

3.5. Энтропия и ее связь со стоимостью оптимального алфавитного кодирования. Блочное кодирование

Важную роль для оценки эффективности кодирования играет энтропия вероятностного распределения:

$$H(P) = -\sum_{i=1}^{m} p_i \log p_i.$$

Пусть $C^*(P)$ − стоимость оптимального алфавитного кодирования.

Теорема 3.5.1. $C^*(P) \geq H(P)$.

При некоторых распределениях стоимость $C^*(P)$ может достигать нижней границы.

Пример 3.5.1. Пусть $P_m = \left(\frac{1}{2}, \frac{1}{2^2}, \ldots, \frac{1}{2^{m-2}}, \frac{1}{2^{m-1}}, \frac{1}{2^{m-1}}\right)$ − распределение вероятностей. Вычислим $C(m) = C(P_m)$.

Положим $l_i = \left\lceil \log \frac{1}{p_i} \right\rceil$. Тогда имеем $l_1 = 1,\ l_2 = 2, \ldots,\ l_i = i, \ldots,\ l_{m-2} = m-2,$ $l_{m-1} = l_m = m-1$.

Величины l_i $(i = \overline{1,m})$ удовлетворяют неравенству Мак-Миллана, следовательно, существует префиксный код с таким набором длин элементарных кодов. Так как p_i являются степенью двойки, $l_i = -\log p_i$, поэтому $C(P_m) = -\sum_{i=1}^{m} p_i \log p_i = H(P_m)$.

В общем же случае $C^*(P) = H(P) + \varepsilon$, где $0 \leq \varepsilon < 1$, как показывает следующая теорема.

Теорема 3.5.2. $C^*(P) < H(P) + 1$.

Стоимость оптимального кодирования может быть как угодно близка и к верхней оценке.

Дополнительные возможности для сжатия могут возникнуть при конечно-автоматном кодировании. Вместо того, чтобы кодировать каждую букву, разобьем сообщение на блоки длины *N*, которые и будем рассматривать как буквы нового алфавита B_N.

Пусть P_N − распределение вероятностей на B_N, которое индуцируется распределением P на B:

$$p_{i_1 i_2 \ldots i_N} = p(b_{i_1} b_{i_2} \ldots b_{i_N}) = p_{i_1} p_{i_2} \ldots p_{i_N}.$$

Теорема 3.5.2. $H(P_N) = N \cdot H(P)$**.**

Выбирая длину блока N достаточно большой, можно сделать стоимость кодирования на одну букву сообщения $C_N(P)$ сколь угодно близкой к $H(P)$ в силу следующей теоремы:

Теорема 3.5.3. $H(P) \le C_N(P) < H(P) + \dfrac{1}{N}$.

Пример 3.5.2. Пусть $B = \{b_1, b_2, b_3\}$, P=$\{0{,}5;\ 0{,}4;\ 0{,}1\}$. Применяя алгоритм Хаффмана к распределению P, получаем следующую схему кодирования:

$$f_1 : \begin{cases} b_1 \to 1 \\ b_1 \to 00 \\ b_1 \to 01. \end{cases}$$

Вычислим стоимость оптимального кодирования: $C_1 = 1{,}5$.

Положим N=2 и рассмотрим всевозможные блоки длины 2. Определим произведение вероятностей каждого блока как произведение вероятностей входящих в него букв:

$$\begin{array}{lll} p(b_1 b_1) = 0{,}25; & p(b_2 b_1) = 0{,}20; & p(b_3 b_1) = 0{,}05; \\ p(b_1 b_2) = 0{,}20; & p(b_2 b_2) = 0{,}16; & p(b_3 b_2) = 0{,}04; \\ p(b_1 b_3) = 0{,}05; & p(b_2 b_3) = 0{,}04; & p(b_3 b_3) = 0{,}01. \end{array}$$

Применяя алгоритм Хаффмана к построенному вероятностному распределению, получим следующую схему кодирования для блоков длины N=2:

$$f_2 : \begin{cases} b_1 b_1 \to 01 \\ b_1 b_2 \to 10 \\ b_1 b_3 \to 00000 \\ b_2 b_1 \to 11 \\ b_2 b_2 \to 001 \\ b_2 b_3 \to 00011 \\ b_3 b_1 \to 00001 \\ b_3 b_2 \to 000100 \\ b_3 b_3 \to 000101. \end{cases}$$

Построенная схема имеет стоимость кодирования одного блока $C = 2{,}78$, и стоимость кодирования одной буквы $C_2 = \frac{C}{2} = 1{,}39$.

Найдем энтропию $H(P)$:

$$H(P) = -(0{,}5 \cdot \log 0{,}5 + 0{,}4 \cdot \log 0{,}4 + 0{,}1 \cdot \log 0{,}1) \approx 1{,}36.$$

Получаем $H(P) \approx 1{,}36 < \frac{C_2}{2} = 1{,}39 < C_1 = 1{,}5$.

Задачи

3.5.1. Для заданного распределения вероятностей построить код по алгоритму Хаффмана и блочный равномерный код с длиной блока $N = 2$. Сравнить их стоимость. Найти нижнюю оценку стоимости кодирования одной буквы:

1) $P = (0{,}3;\ 0{,}2,\ 0{,}5)$;
2) $P = (0{,}3;\ 0;4;\ 0{,}3)$;
3) $P = (0{,}4;\ 0{,}2;\ 0{,}4)$;
4) $P = (0{,}2;\ 0{,}25;\ 0{,}55)$;
5) $P = (0{,}15;\ 0{,}15;\ 0{,}7)$;
6) $P = (0{,}25;\ 0{,}25;\ 0{,}5)$;
7) $P = (0{,}25;\ 0{,}45;\ 0{,}3)$;
8) $P = (0{,}25;\ 0{,}35;\ 0{,}4)$;
9) $P = (0{,}5;\ 0{,}25;\ 0{,}25)$;
10) $P = (0{,}25;\ 0{,}55;\ 0{,}2)$.

3.6. Помехоустойчивое кодирование

Пусть $A=\{0,1\}$ и $M = \{\gamma_1, \gamma_2, \ldots, \gamma_s\}$ − множество всех слов в алфавите A, имеющих длину m, $s = 2^m$. Пусть действует источник помех, который в словах может выдавать ошибки не более чем в t разрядах. Одиночная ошибка состоит в том, что символ 0 заменяется на символ 1, и наоборот.

Рассматривается модель равномерного кодирования, когда слова длины m кодируются словами длины l ($m<l$). Множество слов длины l, которыми кодируются исходные слова длины m, называется ***кодом***. Цель помехоустойчивого кодирования состоит в том, чтобы построить код, который позволял бы обнаруживать ошибки (код, обнаруживающий ошибки), либо код, позволяющий не только обнаруживать, но и исправлять ошибки (код, исправляющий ошибки).

Пусть $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ и $\beta = \{\beta_1, \beta_2, \ldots, \beta_n\}$ - двоичные последовательности длины n, α и β можно рассматривать как вершины n-мерного булева куба B^n.

Расстоянием Хэмминга между α и β называется число $\rho(\alpha, \beta) = \sum_{i=1}^{n} |\alpha_i - \beta_i|$, равное числу координат, в которых векторы α и β различаются. Векторы α и β называются соседними, если $\rho(\alpha, \beta) = 1$.

Пусть $V \in B^n$. Множество V называется ***кодом с расстоянием d***, если $\min \rho(\alpha, \beta) = d$ по всем $\alpha, \beta \in V$. Число d называется ***кодовым расстоянием*** множества V и обозначается $d(V)$.

Код V обнаруживает t ошибок, если любое слово, которое можно получить из произвольного кодового слова $\alpha \in V$ в результате не более t ошибок, отлично от любого другого кодового слова.

Теорема 3.6.1. *Код V обнаруживает t ошибок тогда и только тогда, когда кодовое расстояние $d(V) \geq t + 1$.*

Теорема 3.6.2. *Код V исправляет t ошибок тогда и только тогда, когда кодовое расстояние $d(V) \geq 2t + 1$.*

Рассмотрим простейший случай $t = 1$, то есть случай, когда может произойти не более одной ошибки.

Чтобы обнаружить ошибку, достаточно добавить один дополнительный разряд, в который заносить сумму по модулю 2 передаваемых (информационных) разрядов. Если произошла ошибка, содержимое дополнительного разряда будет отличаться от суммы по модулю 2 информационных разрядов.

Задача исправления ошибки является более сложной, чем задача обнаружения ошибки. Она решена Хэммингом (1950 г.). Код Хэмминга, исправляющий одну ошибку, строится следующим образом.

Сообщения $\alpha_1 \alpha_2 \ldots \alpha_m$ кодируются наборами $\beta_1 \beta_2 \ldots \beta_l$, где l − длина кода, и $l=m+k$. Для каждого исходного слова $\alpha_1 \alpha_2 \ldots \alpha_m$ число различных вариантов на выходе равно l+1, так как ошибка может произойти в любом из l разрядов

набора $\beta_1\beta_2\ldots\beta_l$ или может не произойти. Чтобы дополнительных разрядов хватало для кодировки (l+1) случаев, необходимо, чтобы выполнялось условие: $2^k \geq l+1$, или $2^m \leq 2^l/(l+1)$.

Из этих соображений выберем l как наименьшее целое число, удовлетворяющее неравенству $2^m \leq 2^l/(l+1)$.

Далее построения будут состоять из трех этапов:

1) построение кода Хэмминга;
2) обнаружение ошибки;
3) декодирование.

Построение кода Хэмминга

Построим по отрезку натуральных чисел 1,2, …, l k последовательностей следующим образом.

Пусть $V = v_k v_{k-1}\ldots v_1$ - двоичная запись V, $1 \leq V \leq l$.

Последовательность 1, 3, 5, 7, 9, …содержит все числа V с $V_1 = 1$.

Последовательность 2, 3, 6, 7, 10, … содержит все числа V с $V_2 = 1$.

Последовательность 4, 5, 6, 7, 12, … содержит все числа V с $V_3 = 1$.

… … …

Последовательность $2^{k-1}, 2^{k-1}+1, \ldots$ содержит все числа V с $V_k = 1$.

Первыми членами этих последовательностей являются числа $1 = 2^0$, $2 = 2^1$, … , 2^{k-1}, то есть степени 2.

Разряды β_i набора $\beta_1\beta_2\ldots\beta_l$, у которых номер $i \in \{1,2,\ldots,2^{k-1}\}$, называются ***проверочными***, остальные – ***информационными***. Информационных – m, проверочных – ($l-m$) = k разрядов.

Правила построения набора $\beta_1\beta_2\ldots\beta_l$ по набору $\alpha_1\alpha_2\ldots\alpha_m$:

для информационных разрядов $\alpha_i = \beta_i$,

для проверочных разрядов:

$$\beta_1 = \beta_3 + \beta_5 + \beta_7 + \ldots (\mathrm{mod}\, 2),$$
$$\beta_2 = \beta_3 + \beta_6 + \beta_7 + \ldots (\mathrm{mod}\, 2),$$
$$\ldots \quad \ldots \quad \ldots$$
$$\beta_{2^{k-1}} = \beta_{2^{k-1}+1} + \ldots (\mathrm{mod} 2).$$

Обнаружение ошибки в кодах Хэмминга

Обозначим через H множество всех построенных наборов. Пусть $\beta_1\beta_2\ldots\beta_l \in H$ и при передаче произошла ошибка в S-м разряде, в результате которой мы получили последовательность $\beta'_1\beta'_2\ldots\beta'_l$.

Пусть $S = S_k S_{k-1}\ldots S_1$ – запись числа S в двоичной системе счисления. Рассмотрим число $S' = S'_k\, S'_{k-1}\ldots S'_1$, где:

$$S'_1 = \beta'_1+\beta'_3+\beta'_5+\beta'_7+\ldots(\mathrm{mod}\,2),$$
$$S'_2 = \beta'_2+\beta'_3+\beta'_6+\beta'_7+\ldots(\mathrm{mod}\,2),$$
$$\ldots \qquad \ldots \qquad \ldots$$
$$S'_{2^{k-1}} = \beta'_{2^{k-1}}+\beta'_{2^{k-1}+1}+\ldots(\mathrm{mod}\,2).$$

Номер разряда, в котором произошла ошибка, определяется из равенства $S = S'$. Если ошибки не произошло, то $S' = 0$.

Декодирование

После исправления ошибки в S-м разряде (если она произошла) для декодирования достаточно оставить информационные разряды.

Пример 3.6.1. Пусть m=4. Определим наименьшее число l, удовлетворяющее неравенству $2^4 \le 2^l/(l+1)$: l=7, k=3.

Информационными будут разряды с номерами 3, 5, 6, 7. Разряды с номерами 1, 2, 4, которые являются степенями 2, будут проверочными. Содержимое проверочных разрядов определяется следующими равенствами:

$$\beta_1 = \beta_3 + \beta_5 + \beta_7 (\mathrm{mod}\,2),$$
$$\beta_2 = \beta_3 + \beta_6 + \beta_7 (\mathrm{mod}\,2),$$
$$\beta_4 = \beta_5 + \beta_6 + \beta_7 (\mathrm{mod}\,2).$$

Код Хэмминга запишем в виде таблицы, где i-й столбец соответствует i-му разряду закодированного слова. Проверочные разряды помечены символом *.

1*	2*	3	4*	5	6	7
0	0	0	0	0	0	0
1	1	0	1	0	0	1
0	1	0	1	0	1	0
1	0	0	0	0	1	1
1	0	0	1	1	0	0
0	1	0	0	1	0	1

1	1	0	0	1	1	0
0	0	0	1	1	1	1
1	1	1	0	0	0	0
0	0	1	1	0	0	1
1	0	1	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	1	0	1	1	0
0	0	1	0	1	1	0
1	1	1	1	1	1	1

Пусть получено слово $\beta = 0110111$. Определим, произошла ли при передаче ошибка и в каком разряде. Вычислим S'.

$$S'_1 = \beta'_1 + \beta'_3 + \beta'_5 + \beta'_7 \pmod 2 = 0 + 1 + 1 + 1 = 1,$$

$$S'_2 = \beta'_2 + \beta'_3 + \beta'_6 + \beta'_7 \pmod 2 = 1 + 1 + 1 + 1,$$

$$S'_4 = \beta'_4 + \beta'_5 + \beta'_6 + \beta'_7 \pmod 2 = 0 + 1 + 1 + 1.$$

Получаем, что S'=101 в двоичной системе счисления, или S'=5. Значит, ошибка произошла в 5-м разряде. После исправления ошибки получим слово 0110011. Выделив в нем информационные разряды, получим исходное слово 1011.

Задачи

3.6.1. Для данного множества $V \subseteq B^n$ найти кодовое расстояние:
1) V = {11000, 10101, 01110},
2) V = {111100, 110011, 001111},
3) V = {101010, 010110, 000001},
4) V = {01101010, 11000110, 00011001, 10101100}.

3.6.2. Для каждого из кодов V предыдущей задачи найти:
1) число ошибок, которые код обнаруживает;
2) число ошибок, которые код исправляет.

3.6.3. Построить по методу Хэмминга кодовое слово для сообщения:

1) $\alpha = 010;$ 5) $\alpha = 10101011;$

2) $\alpha = 011;$ 6) $\alpha = 111001111;$

3) $\alpha = 1001;$ 7) $\alpha = 100010011;$

4) $\alpha = 1101;$ 8) $\alpha = 01110111011.$

3.6.4. По каналу связи передавалось кодовое слово, построенное по методу Хэмминга для сообщения α. После передачи по каналу связи, искажающему слово не более чем в одном разряде, было получено слово β. Восстановить исходное сообщение.

1) $\beta = 110;$ 6) $\beta = 1011101;$

2) $\beta = 101110;$ 7) $\beta = 1100011;$

3) $\beta = 011110;$ 8) $\beta = 11011100110;$

4) $\beta = 1001011;$ 9) $\beta = 1010101010100;$

5) $\beta = 0101101;$ 10) $\beta = 00101111011111.$

3.6.5. Закодировать сообщения кодом Хэмминга в случае *m*=3:

1) 001011011010;

2) 011010100110111.

Ответы к задачам главы 1

1.1. 1) $n_1+n_2+n_3$; 2) $n_1\cdot n_2+n_1\cdot n_3+n_2\cdot n_3$; 3) $n_1\cdot n_2\cdot n_3$; 4) $\binom{n_1}{2}+\binom{n_2}{2}+\binom{n_3}{2}$; 5) $\binom{n_1}{3}+\binom{n_2}{3}+\binom{n_3}{3}$; 6) $n_1\cdot\binom{n_2}{2}\cdot\binom{n_3}{3}$. **1.2**. 1) 2^n; 2) 3^n. **1.3**. q^n; $q^{\left[\frac{n+1}{2}\right]}$. **1.4**. $q\cdot(q-1)^{n-1}$. **1.5**. $2^{m\cdot n}$. **1.6.** $\binom{2^n}{m}$. **1.7**. 3136. **1.8.** 3612. **1.9.** 1)126; 2)210. **1.10.** 1) 2^n; 2) $\binom{n}{k}$. **1.11.** 1) A_n^3; 2) $\binom{n}{3}$. **1.12.** 1) 2^k-1; 2) $2^{n-k}-1$; 3) 2^{n-k}; 4) 2^n-2^{n-k}; 5) $k\cdot 2^{n-k}$; 6) $2^n-2^{n-k}-k\cdot 2^{n-k}$; 7) $\binom{k}{2}\cdot 2^{n-k}$; 8) $\binom{n-k}{2}\cdot 2^k$; 9) $\binom{k}{3}\cdot\binom{n-k}{4}$; 10) n. **1.13**. 12; 32; 42; $(k_1+1)\ldots(k_s+1)$. **1.14.** 8!. 1**.15.** $\binom{52}{10}$**;** 1) $\binom{52}{10}-\binom{48}{10}$; 2) $4\cdot\binom{48}{9}$; 3) $\binom{48}{6}$; 4) $\binom{52}{10}-\binom{48}{10}-4\cdot\binom{48}{9}$; 5) $6\cdot\binom{48}{8}$. **1.16**. 1) $4(n-4)$; 2) $4n(n-1)$; 3) $24n(n-1)$; 4) $4\binom{n}{5}$; 5) $4^5(n-4)$; 6) $4n\cdot\binom{4n-4}{2}$; 7) $4^3 n\cdot\binom{4}{2}\cdot\binom{n-1}{3}$. **1.17**. 1)27216; 2)62784; 3)24192; 4)21504; 5)8100; 6)45000. **1.18.** k^n**;** A_k^n**. 1.19.** $26\cdot 36^4$. **1.20.** 1)11!; 2)34560; 3)11520. **1.21**.6930. **1.22**. 88200. **1.23**. 45360. **1.24**. 105. **1.25**. 1)27; 2)5. **1.26**. 1)2; 2)6; 3)3. **1.27**. 1)10; 2)90; 3)13; 4)25. **1.28.** 1)45360; 2)14580; 3)5040; 4)34560; 5)70000. **1.29.** 1)13428; 2)8008; 3)2851; 4)12800; 5)3024.

Литература

1. Алексеев В.Е., Киселева Л.Г., Смирнова Т.Г. Сборник задач по дискретной математике. − Методическая разработка, Нижний Новгород, 2007. 48 с.

2. Алексеев В.Е., Таланов В.А. Графы. Модели вычислений. Структуры данных. − Нижний Новгород: Изд-во ННГУ, 2005. 307 с.

3. Гаврилов Г.П., Сапоженко А.А. Задачи и упражнения по дискретной математике. − М.: ФИЗМАТЛИТ, 2006. 416 с.

4. Емеличев В.А., Мельников О.И., Сарванов В.И., Тышкевич Р.И. Лекции по теории графов. − М.: Наука, 1990. 384 с.

5. Марков А.А. Введение в теорию кодирования. − М.: Наука, 1982. 192 с.

6. Яблонский С.В. Введение в дискретную математику. − М.: Наука, 2000. 384 с.

Оглавление

Глава 1. Элементы комбинаторики……3
Задачи……7

Глава 2. Теория графов……12
2.1. Основные понятия теории графов……12
Задачи……13
2.2. Пути, циклы, обходы графа. Метрические характеристики……15
Задачи……17
2.3. Деревья……19
Задачи……20
2.4. Планарные графы ……21
Задачи……23
2.5. Паросочетания……24
Задачи……27
2.6. Независимые множества. Клики графа……28
Задачи……30
2.7. Раскраски……30
Задачи……32
2.8. Задача о кратчайших путях……33
Задачи……36

Глава 3. Теория кодирования……38
3.1. Основные понятия теории кодирования……38
3.2. Проблема распознавания взаимной однозначности кодирования……39
Задачи……43
3.3. Построение префиксного кода по набору длин элементарных кодов……45
Задачи……46
3.4. Алгоритмы экономного алфавитного кодирования……47
Задачи……51
3.5. Энтропия и ее связь со стоимостью оптимального алфавитного кодирования. Блочное кодирование……53
Задачи……55
3.6. Помехоустойчивое кодирование……55
Задачи……59

Ответы к задачам главы 1……61

Литература……62

Лариса Павловна ***Жильцова***

Татьяна Геннадьевна ***Смирнова***

ОСНОВЫ ТЕОРИИ ГРАФОВ И ТЕОРИИ КОДИРОВАНИЯ В ПРИМЕРАХ И ЗАДАЧАХ

Учебно-методическое пособие

Государственное образовательное учреждение высшего профессионального образования «Нижегородский государственный университет им. Н.И.Лобачевского»
603950, Нижний Новгород, пр.Гагарина, 23

Подписано в печать Формат 60 × 84 1/16.
Бумага офсетная. Печать офсетная. Гарнитура Таймс.
Усл.-печ. л. 4,0. Заказ Тираж 200 экз.

Отпечатано в типографии Нижегородского госуниверситета им. Н.И. Лобачевского
603600, Н.Новгород, ул. Большая Покровская, 37
Лицензия ПД № 18-0099 от 14.05.01