Министерство образования и науки, молодёжи и спорта Украины
Донбасская государственная машиностроительная академия (ДГМА)

Л. В. Белевцов, Е. Ю. Гудкова

ВВЕДЕНИЕ В ДИСКРЕТНУЮ МАТЕМАТИКУ

Учебное пособие

Утверждено
на заседании ученого совета
Протокол № 4 от 29.11.2012

Краматорск
ДГМА
2013

УДК 519.854
ББК 22.176
Б 43

Рецензенты:

Кухтик Т. В., д-р техн. наук, профессор, Донбасский институт техники и менеджмента Международного технического университета;

Бойко В. Г., канд. техн. наук, доцент, Краматорский экономико-гуманитарный институт.

У навчальному посібнику висвітлено основні теми згідно з навчальною програмою з дискретної математики у процесі підготовки бакалаврів і додаткові теми для розв'язання різноманітних комп'ютерних задач.

Для покращення засвоєння теоретичного матеріалу та закріплення вмінь і навичок із дискретної математики наведено практичні приклади, контрольні питання у кінці кожного розділу та тестові завдання для самоперевірки.

Белевцов, Л. В.

Б 43 Введение в дискретную математику: учебное пособие / Л. В. Белевцов, Е. Ю. Гудкова. – Краматорск : ДГМА, 2013. – 144 с.

ISBN 978-966-379-3.

В учебном пособии отражены основные темы учебной программы по дискретной математике при подготовке бакалавров и дополнительные темы для решения разнообразных компьютерных задач.

Для улучшения усвоения теоретического материала и закрепления умений и навыков по дискретной математике приведены практические примеры, контрольные вопросы в конце каждого раздела и тестовые задания для самопроверки.

УДК 519.854
ББК 22.176

ISBN 978-966-379-3

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Данное учебное пособие представляет собой конспективный учебник по дискретной математике, включающий в себя описания важнейших алгоритмов над объектами дискретной математики. Учебник ориентирован на студентов компьютерных специальностей и студентов вузов, которым по роду их занятий приходится иметь дело с конструированием и анализом нетривиальных алгоритмов.

В настоящее время имеется масса доступной литературы, отражающей данную тематику. С одной стороны, существуют десятки достойных книг по дискретной математике, начиная с элементарных учебников для начинающих и кончая исчерпывающими справочниками для специалистов. С другой стороны, большое количество монографий, многие из которых стали классическими, посвящены вопросам теории программирования. Как правило, такие монографии содержат детальное описание и анализ важнейших и известнейших алгоритмов. Данное учебное пособие принадлежит к математической литературе, в которой математическое изложение доводится до уровня практически исполнимых программ.

Учебник основан на лекционном курсе, который авторы в течение ряда лет читают студентам кафедры «Интеллектуальные системы принятия решений» Донбасской государственной машиностроительной академии, что наложило определенный отпечаток на состав материала.

Учебное пособие охватывает почти все основные разделы дискретной математики, включая теорию множеств, комбинаторику, теорию графов, теорию булевых функций, теорию кодирования, необходимые для программиста. Также включены более специальные разделы, такие как теория формальных грамматик и теория конечных автоматов.

В целом учебное пособие преследует следующие основные цели:

1. Сформировать у студента терминологический запас, необходимый для самостоятельного изучения специальной математической литературы, ознакомив его с максимально широким кругом понятий дискретной математики;

2. Довести до студента необходимые конкретные сведения из дискретной математики, предусмотренные образовательно-профессиональной программой подготовки бакалавров;

3. Изучить алгоритмы решения типовых задач дискретной математики и способов представления математических объектов в программах.

МОДУЛЬ 1. ТЕОРИЯ МНОЖЕСТВ, ГРАФЫ, КОМБИНАТОРНЫЙ АНАЛИЗ, БУЛЕВА ЛОГИКА

1 МНОЖЕСТВА

1.1 Основные понятия теории множеств

Выделение объектов и их совокупностей является естественным способом организации человеческого мышления. Понятие множества принадлежит к числу фундаментальных неопределяемых понятий математики. О множестве известно как минимум, что оно состоит из элементов.

Множеством называется совокупность каких-либо объектов, обладающая общим для всех характеристическим свойством. Это определение нельзя считать строгим, так как понятие множества является исходным понятием математики и не может быть определено через другие математические объекты.

Пример 1.1. Следующие совокупности объектов являются множествами: множество деревьев в лесу, множество целых чисел, множество корней уравнения $e^x \cdot sin(x) = 0{,}5$.

Всякое множество состоит из *элементов*. Множества обозначают большими буквами, например *A, B, C*, а элементы – маленькими буквами, например, *a, b, c*.

Множество и его элементы обозначаются следующим образом:

$A = \{a_1, a_2, a_3\}$ – множество, состоящее из трех элементов;

$A = \{a_1, a_2, \ldots\}$ – множество, состоящее из бесконечного числа элементов.

В первом случае множество называется ***конечным***, поскольку содержит конечное множество элементов. Во втором случае – ***бесконечным***, поскольку содержит неограниченное количество элементов.

Способы задания множеств

1. Множество может быть задано простым перечислением элементов.

Пример 1.2. $A = \{2, 4, 6, 8, 10\}$ – конечное множество; $B = \{1, 2, \ldots, n, \ldots\}$ – бесконечное множество.

2. Указанием свойств элементов множества. Для этого способа пользуются следующим форматом записи: $A = \{a \mid$ указание свойства элементов$\}$. Здесь a является элементом множества A, $a \in A$.

Пример 1.3. $A = \{a \mid a$ – простое число$\}$ – множество простых чисел; $B = \{b \mid b^2 - 1 = 0,\ b$ – действительное число$\}$ – множество, состоящее из двух элементов, $B = \{-1, 1\}$.

3. Множество может быть задано рекурсивным указанием способа последовательного порождения его элементов.

Пример 1.4. Множество значений рекурсивной функции является рекурсивно-заданным множеством $F = \{\varphi_1, \varphi_2, \varphi_3, \ldots\}$, где $\varphi_i \in N$, $i = 1, 2, 3, \ldots$ Пусть $\varphi_1 = 1$, $\varphi_2 = 2$, а каждое следующее число зависит от двух предыдущих таким образом: $\varphi_n = \varphi_{n-2} + \varphi_{n-1}$, $n = 3, 4, \ldots$. Тогда $\varphi_3 = \varphi_1 + \varphi_2 = 1 + 2 = 3$, $\varphi_4 = \varphi_2 + \varphi_3 = 2 + 3 = 5$.

Множество может состоять из элементов, которые сами являются множествами. Нужно различать элемент a и множество, состоящее из единственного элемента a.

Пример 1.5. Множество $A = \{1, 2\}$ состоит из двух элементов 1, 2; но множество $\{A\}$ состоит из одного элемента A.

Если элемент a принадлежит множеству A, это записывается следующим образом: $a \in A$. Если элемент a не принадлежит множеству A, то записывают так: $a \notin A$.

Пример 1.6. Пусть A_1 – множество простых чисел, A_2 – множество четных чисел, $a = 5$. Тогда $a \in A_1$, $a \notin A_2$.

Если все элементы множества A являются элементами множества B и наоборот, т. е. множества A и B совпадают, то говорят, что $A = B$.

Если каждый элемент множества A является элементом множества B, говорят, что множество A является ***подмножеством*** множества B, и записывают $A \subseteq B$ или $B \supseteq A$. Отметим, что по определению само множество A является своим подмножеством, т. е. $A \subseteq A$.

Если $A \subseteq B$ и $B \subseteq A$, то по ранее введенному определению $A = B$.

Если $A \subseteq B$ и $A \neq B$, то A есть *собственное подмножество* B, $A \subset B$. Если A не является собственным подмножеством B, то записывают $A \not\subset B$.

Пример 1.7. Пусть A – множество четных чисел, B – множество целых чисел, C – множество нечетных чисел. Тогда $A \subset B$, $C \subset B$, $A \not\subset C$, $B \not\subset A$.

Необходимо различать *отношение принадлежности* ($\in$) и *отношение включения* ($\subseteq$).

Пример 1.8. Пусть $A = \{1\}$ – множество, состоящее из одного элемента, $B = \{\{1\}, \{3\}\}$ – множество, состоящее из двух элементов, каждое из которых является одноэлементным множеством. Тогда имеют место следующие соотношения:

$1 \in \{1\}$;

$\{1\} \subset \{\{1\}, \{3\}\}$;

$1 \notin \{\{1\}, \{3\}\}$.

Множество, не содержащее ни одного элемента, называется ***пустым множеством*** и обозначается $\varnothing$. Принято считать, что пустое множество является подмножеством любого множества, $\varnothing \subseteq A$, где A – любое множество. Таким образом, всякое множество содержит в качестве своих подмножеств пустое множество и само себя.

Пример 1.9. Множество корней уравнения $sin(x) = 2$ является пустым.

Универсальным называется множество, которое содержит все возможные элементы, которые встречаются в определенной задаче. Универсальное множество обозначается символом ***U***.

Множество всех подмножеств данного множества A называется ***множеством-степенью*** и обозначается $P(A)$. Множество $P(A)$ состоит из 2^n элементов.

Пример 1.10. Пусть множество $A = \{1, 2\}$ состоит из двух элементов 1, 2. Тогда множество $P(A)$ включает в себя пустое множество $\varnothing$, два одноэлементных множества $\{1\}$ и $\{2\}$ и само множество $A = \{1, 2\}$, т. е. $P(A) = \{\varnothing, \{1\}, \{2\}, \{1, 2\}\}$. Множество $P(A)$ состоит из четырех элементов ($4 = 2^2$).

1.2 Геометрическая интерпретация множеств. Операции над множествами

Диаграммы Эйлера-Венна – геометрические представления множеств. Построение диаграммы заключается в изображении большого прямоугольника, представляющего универсальное множество U, а внутри его – кругов (или каких-нибудь других замкнутых фигур), представляющих множества. Фигуры должны пересекаться в наиболее общем случае,

требуемом в задаче, и должны быть соответствующим образом обозначены. Точки, лежащие внутри различных областей диаграммы, могут рассматриваться как элементы соответствующих множеств. Имея построенную диаграмму, можно заштриховать определенные области для обозначения вновь образованных множеств.

Операции над множествами рассматриваются для получения новых множеств из уже существующих. Рассмотрим основные операции над множествами с помощью диаграмм Эйлера-Венна.

Объединением множеств A и B называется множество, состоящее из всех тех элементов, которые принадлежат хотя бы одному из множеств A, B (рис. 1.1, а): $A \cup B = \{x \mid x \in A \text{ или } x \in B\}$. Из определения следует, что $A \subseteq A \cup B$ и $B \subseteq A \cup B$.

Пример 1.11. Пусть $A = \{4, 5, 6\}$, $B = \{2, 4, 6\}$. Тогда $A \cup B = \{2, 4, 5, 6\}$.

Пересечением множеств A и B называется множество, состоящее из всех тех и только тех элементов, которые принадлежат одновременно как множеству A, так и множеству B (рис. 1.1, б): $A \cap B = \{x \mid x \in A \text{ и } x \in B\}$ Из определения следует, что $A \cap B \subseteq A$, $A \cap B \subseteq B$ и $A \cap B \subseteq A \cup B$.

Пример 1.12. Пусть $A = \{4, 5, 6\}$, $B = \{2, 4, 6\}$.Тогда $A \cap B = \{4, 6\}$.

Может оказаться, что множества не имеют ни одного общего элемента. Тогда говорят, что множества не пересекаются или их пересечение – пустое множество.

Относительным дополнением множества B до множества A (разностью множеств A и B) называется множество всех тех и только тех элементов A, которые не содержатся в B (рис. 1.1, в): $A \setminus B = \{x \mid x \in A \text{ и } x \notin B\}$.

Пример 1.13. Рассмотрим данные из примера 1.11. $A = \{4, 5, 6\}$, $B = \{2, 4, 6\}$, тогда $A \setminus B = \{5\}$, $B \setminus A = \{2\}$.

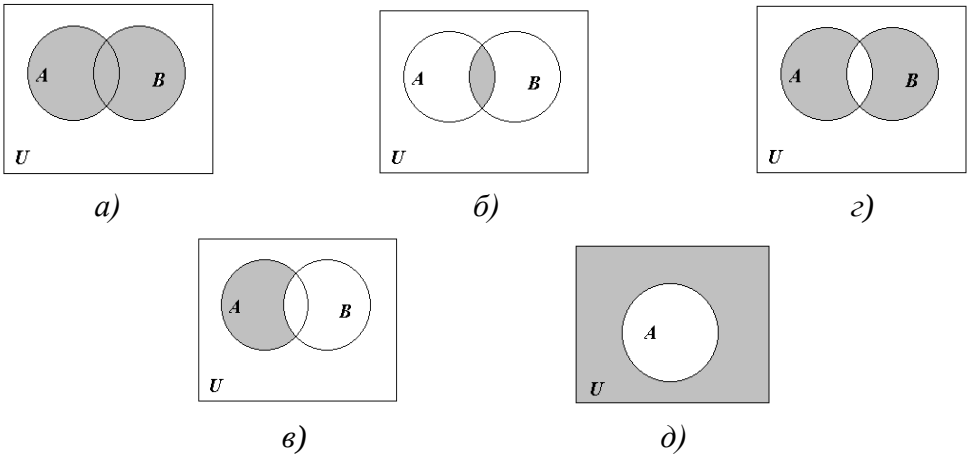
Симметрической разностью множеств A и B называется множество элементов этих множеств, которые принадлежат либо только множеству A, либо только множеству B (рис. 1.1, г): $A + B = ((A \setminus B) \cup (B \setminus A)) = \{x \mid \text{либо } x \in A, \text{ либо } x \in B\}$.

Пример 1.14. Рассмотрим данные из примера 1.13. $A = \{4, 5, 6\}$, $B = \{2, 4, 6\}$. $A \setminus B = \{5\}$, $B \setminus A = \{2\}$, $A + B = \{2, 5\}$.

Абсолютным дополнением множества A называется множество всех тех элементов, которые не принадлежат множеству A (рис. 1.1, д): $\overline{A} = U \setminus A$.

Пример 1.15. Пусть A – множество положительных нечетных чисел,

тогда U – множество всех натуральных чисел и $\overline{A}$ – множество положительных четных чисел



а – $A \cup B$; *б* – $A \cap B$; *в* – $A \setminus B$; *г* – $A + B$; *д* – $\overline{A}$

Рисунок 1.1 – Геометрическая интерпретация операций над множествами

1.3 Алгебра множеств. Основные тождества алгебры множеств

Множества вместе с определенными на них операциями образуют *алгебру множеств*. Последовательность выполнения операций задается с помощью *формулы алгебры множеств*. Например, $\overline{A} \cap (B \cup C)$, $(A \setminus B) + C$ – формулы алгебры множеств.

Основные тождества алгебры множеств

Для любых множеств *A, B, C* справедливы следующие тождества:

1. *Коммутативность*:

а) $A \cup B = B \cup A$ (для объединения);

б) $A \cap B = B \cap A$ (для пересечения).

2. *Ассоциативность*:

а) $A \cup (B \cup C) = (A \cup B) \cup C$ (для объединения);

б) $A \cap (B \cap C) = (A \cap B) \cap C$ (для пересечения).

3. *Дистрибутивность*:

а) $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ (для объединения относительно пересечения);

б) $A\cap(B\cup C) = (A\cap B)\cup(A\cap C)$ (для пересечения относительно объединения).

4. *Закон де Моргана*:

а) $\overline{A\cup B} = \overline{A}\cap\overline{B}$ (дополнение к объединению есть пересечение дополнений);

б) $\overline{A\cap B} = \overline{A}\cup\overline{B}$ (дополнение к пересечению есть объединение дополнений).

5. *Идемпотентность*:

а) $A\cup A = A$ (для объединения);

б) $A\cap A = A$ (для пересечения).

6. *Поглощение (закон элиминации):*

а) $A\cup (A\cap B) = A$;

б) $A\cap (A\cup B) = A$.

7. *Расщепление* (*склеивание*):

а) $(A\cup B)\cap (A\cup \overline{B}) = A$;

б) $(A\cap B)\cup (A\cap \overline{B}) = A$.

8. *Двойное дополнение (закон инволюции)*:

$\overline{\overline{A}} = A$.

9. *Закон исключенного третьего:*

$A\cup \overline{A} = U$.

10. *Закон противоречия:*

$A\cap \overline{A} = \varnothing$.

11. *Операции с пустым и универсальным множествами:*

а) $A\cup U = U$;

б) $A\cup \varnothing = A$;

в) $A\cap U = A$;

г) $A\cap \varnothing = \varnothing$;

д) $\overline{\varnothing} = U$;

е) $\overline{U} = \varnothing$.

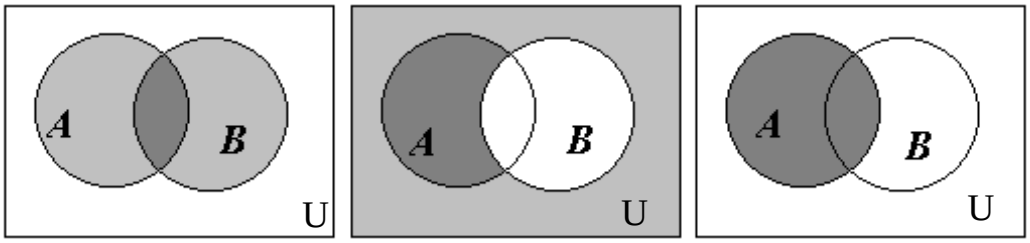
12. $A\setminus B = A\cap\overline{B}$.

Пример 1.16. Доказать следующее тождество $(A\cap B)\cup\left(A\cap\overline{B}\right) = A$.

Докажем это тождество двумя способами: аналитически (используя равносильности алгебры множеств) и конструктивно (используя диаграммы Эйлера – Венна).

1. $(A\cap B)\cup\left(A\cap\overline{B}\right) = \left(A\cup\left(A\cap\overline{B}\right)\right)\cap\left(B\cup\left(A\cap\overline{B}\right)\right) = A\cap\left((B\cup A)\cap\left(B\cup\overline{B}\right)\right) = A\cap$ $\cap\left((B\cup A)\cap U\right) = A\cap\left(B\cup A\right) = A$.

2. Построим соответствующие диаграммы Эйлера – Венна (рис. 1.2).



$A \cap B$ $A \cap \overline{B}$ $(A \cap B) \cup (A \cap \overline{B}) = A$

Рисунок 1.2 – Доказательство тождества с применением диаграмм Эйлера – Венна

При выполнении аналитических преобразований сначала выполняются операции дополнения, затем пересечения, объединения и разности, которые имеют одинаковый приоритет.

1.4 Эквивалентность множеств

Если каждому элементу множества A сопоставлен единственный элемент множества B и при этом всякий элемент множества B оказывается сопоставленным одному и только одному элементу множества A, то говорят, что между множествами A и B существует *взаимно однозначное соответствие.* Множества A и B в этом случае называют *эквивалентными или равномощными.*

Эквивалентность множеств обозначается следующим образом: $A \sim B$.

Эквивалентность множеств обладает следующим свойством *транзитивности*: если $A \sim B$ и $B \sim C$, то $A \sim C$.

Докажем это свойство. Так как $A \sim B$, то для всякого элемента $a \in A$ существует единственный элемент $b \in B$. Но так как $B \sim C$, то для всякого элемента $b \in B$ существует единственный элемент $c \in C$. Сопоставим этот элемент элементу $a \in A$. Значит, для всякого элемента $a \in A$ существует единственный элемент $c \in C$ и для всякого элемента $c \in C$ существует единственный элемент $a \in A$. Следовательно, $A \sim C$.

Очевидно, что два конечных множества эквивалентны тогда и только тогда, когда количество элементов в них одинаково. Например, множества $A = \{4, 5, 6\}$ и $B = \{x, y, z\}$ эквивалентны, $A \sim B$. Взаимно однозначное соответствие может быть установлено между элементами 4 и x, 5 и y, 6 и z.

Мощностью конечного множества A (обозначается $|A|$) называется число элементов этого множества. Например, мощность множества $A = \{1, 2, 3\}$ равна $|A| = 3$.

Ранее рассматривалось множество всех подмножеств данного множества A, которое называется множеством-степенью и обозначается $P(A)$. Множество $P(A)$ состоит из 2^n элементов. Таким образом, $|P(A)| = 2^n$.

Рассмотрим задачу определения мощности объединения n конечных множеств.

Пусть $n = 2$ и A и B – два пересекающихся множества. Докажем с помощью диаграммы Эйлера – Венна следующее соотношение:

$$|A \cup B| = |A| + |B| - |A \cap B|. \qquad (1.1)$$

Из рис. 1.3 видно, что

$|A \cup B| = n_1 + n_2 + n_3$;

$|A| = n_1 + n_2$;

$|B| = n_2 + n_3$;

$|A \cap B| = n_2$.

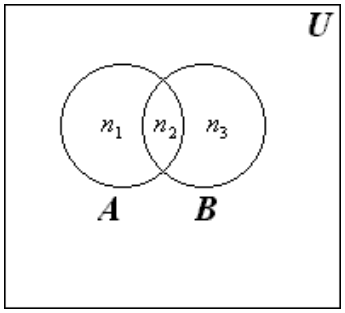


Рисунок 1.3 – Графическая интерпретация метода включений – исключений для 2-х множеств

Очевидно, что $n_1 + n_2 + n_3 = (n_1 + n_2) + (n_2 + n_3) - n_2$, что и доказывает формулу (1.1).

Формула (1.1) справедлива и для случая, если множества A и B не пересекаются. В этом случае $|A \cup B| = |A| + |B|$.

Пусть $n = 3$ и A, B и C – три пересекающихся множества. В этом случае справедливо следующее соотношение:

$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C| \quad (1.2)$$

Из рис. 1.4 видно, что

$|A \cup B \cup C| = n_1 + n_2 + n_3 + n_4 + n_5 + n_6 + n_7$;

$|A| = n_1 + n_2 + n_4 + n_5$;

$|B| = n_2 + n_3 + n_5 + n_6$;

$|C| = n_4 + n_5 + n_6 + n_7$;

$|A \cap B| = n_2 + n_5$;

$|A \cap C| = n_4 + n_5$;

$|B \cap C| = n_5 + n_6$;

$|A \cap B \cap C| = n_5$.

Очевидно, что $n_1 + n_2 + n_3 + n_4 + n_5 + n_6 + n = (n_1 + n_2 + n_4 + n_5) + (n_2 + n_3 + n_5 + n_6) + (n_4 + n_5 + n_6 + n_7) - (n_2 + n_5) - (n_4 + n_5) - (n_5 + n_6) + n_5$, что и доказывает формулу (1.2).

Формула (1.2) справедлива и для случая, если множества A, B и C попарно не пересекаются. В этом случае $|A \cup B \cup C| = |A| + |B| + |C|$.

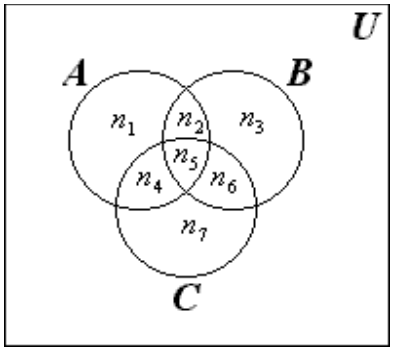


Рисунок 1.4 – Графическая интерпретация метода включений – исключений для 3-х множеств

В общем случае мощность объединения n множеств определяется по формуле:

$$|A_1 \cup A_2 \cup \ldots \cup A_n| = |A_1| + |A_2| + \ldots + |A_n| - (|A_1 \cap A_2| + |A_1 \cap A_3| + \ldots + |A_{n-1} \cap A_n|) + |A \cap B \cap C| + (|A_1 \cap A_2 \cap A_3| + \ldots + |A_{n-2} \cap A_{n-1} \cap A_n|) - \ldots + (-1)^{n-1} |A_1 \cap A_2 \ldots \cap A_n|. \quad (1.3)$$

Эта формула выводится индукцией по n и называется *формулой включений – исключений.*

Если множества A_i попарно не пересекаются, т. е.

$A_i \cap A_j = \varnothing$, $i \neq j$, то получим частный случай формулы (1.3):

$|A_1 \cup A_2 \cup \ldots \cup A_n| = |A_1| + |A_2| + \ldots + |A_n|$.

В общем случае справедливо неравенство

$|A_1 \cup A_2 \cup \ldots \cup A_n| \leq |A_1| + + |A_2| + \ldots + |A_n|$.

Понятие эквивалентности справедливо и для бесконечных множеств. Пусть, например, $A = \{1, 2, 3, \ldots, n,\ldots\}$, $B = \{-1, -2, \ldots, -n, \ldots\}$, тогда $A \sim B$. Взаимно однозначное соответствие устанавливается по правилу: элементу $n \in A$ соответствует элемент $-n \in B$, т.е. $n \leftrightarrow -n$.

Пример 1.17. $A = \{1, 2, 3, \ldots, n,\ldots\}$, $B = \{2, 4, \ldots, 2n, \ldots\}$. Тогда $A \sim B$. Взаимно однозначное соответствие устанавливается по правилу: $n \leftrightarrow 2n$.

1.5 Бесконечные множества

Множество, эквивалентное множеству натуральных чисел $N = \{1, 2, 3, \ldots, n,\ldots\}$, называется *счетным*.

Можно сказать также, что множество счетно, если его элементы можно перенумеровать.

Пример 1.18. Следующие множества являются счетными:

1. $A_1 = \{-1, -2, \ldots, -n, \ldots\}$;
2. $A_2 = \{2, 2^2, \ldots, 2^n,\ldots\}$;
3. $A_3 = \{2, 4, \ldots, 2n,\ldots\}$.

Чтобы установить, является ли множество счетным, достаточно указать взаимно однозначное соответствие между элементами данного множества и множества натуральных чисел. Для примера 1.18 взаимно однозначное соответствие устанавливается по следующим правилам: для множества A_1: $-n \leftrightarrow n$; для множества A_2: $2^n \leftrightarrow n$; для множества A_3: $2n \leftrightarrow n$.

Установить, является ли множество счетным можно также, используя следующие *теоремы о счетных множествах*.

Теорема 1. Всякое бесконечное подмножество счетного множества счетно.

Теорема 2. Объединение конечной или счетной совокупности счетных множеств счетно.

Теорема 3. Множество всех рациональных чисел, т. е. чисел вида $\frac{p}{q}$, где p и q – целые числа, счетно.

Теорема 4. Если $A = \{a_1, a_2, \ldots\}$ и $B = \{b_1, b_2, \ldots\}$ – счетные множества, то множество всех пар $C = \{(a_k, b_n), k = 1, 2,\ldots; n = 1, 2, \ldots\}$ счетно.

Теорема 5. Множество всех многочленов $P(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$ любых степеней с рациональными коэффициентами $a_0, a_1, a_2, \ldots, a_n$ счетно.

Теорема 6. Множество всех корней многочленов любых степеней с рациональными коэффициентами счетно.

Существуют бесконечные множества, элементы которых нельзя перенумеровать. Такие множества называются *несчетным,* а их мощность считается большей, чем |N|. Так, множество точек отрезка [0, 1] = $= \{x \in R; 0 \le x \le 1\}$ – несчетно (теорема Кантора). Его мощность называется континуум и обозначается c: $|[0,1]| = c$. Само множество и любое эквивалентное ему множество называются ***континуальными***.

Установить мощность континуума можно, используя следующие *теоремы о множествах мощности континуума*.

Теорема 1. Множество всех подмножеств счетного множества счетно.

Теорема 2. Множество иррациональных чисел имеет мощность континуума.

Теорема 3. Множество всех точек n-мерного пространства при любом n имеет мощность континуума.

Теорема 4. Множество всех комплексных чисел имеет мощность континуума.

Теорема 5. Множество всех непрерывных функций, определенных на отрезке $[a, b]$, имеет мощность континуума.

Итак, мощности бесконечных множеств могут различаться. Мощность континуума больше, чем мощность счетного множества. Ответ на вопрос, существуют ли множества более высокой мощности, чем мощность континуума, дает следующая теорема (приводится без доказательства).

Теорема о множествах высшей мощности. Множество всех подмножеств данного множества имеет более высокую мощность, чем данное множество.

Из этой теоремы следует, что множеств с максимально большой мощностью не существует.

Контрольные вопросы к разделу 1

1. Что такое множество? Привести примеры множеств.

2. В каких случаях множества считаются равными?

3. Что такое подмножество?

4. Любое ли множество содержит пустое подмножество? Ответ обосновать.

5. Существует ли различие между понятиями включения и принадлежности?

6. Что такое множество-степень?

7. Что такое универсальное множество (универсум)?

8. Какие существуют способы задания множеств? Привести пример.

9. Как определить мощность множества?

10. Дать следующие определения: бесконечное множество, счетное множество, множество континуума.

2 ТЕОРИЯ ГРАФОВ

2.1 Основные характеристики графов

Зарождение теории графов как математической дисциплины связывают с работой Л. Эйлера (1736 г.). Вначале эта теория была связана с математическими головоломками и играми. Однако впоследствии теория графов стала использоваться в топологии, алгебре, теории чисел. В настоящее время теория графов находит применение в самых разнообразных областях науки, техники и практической деятельности. Она используется при проектировании электрических сетей, планировании транспортных перевозок, построении молекулярных схем. Применяется теория графов также в экономике, психологии, социологии, биологии.

Граф G – это математический объект, состоящий из множества *вершин* $X = \{x_1, x_2, ..., x_n\}$ и множества *ребер* $A = \{a_1, a_2, ..., a_n\}$. Таким образом, граф полностью определяется совокупностью множеств X, A: $G = (X, A)$.

Если ребрам графа приданы направления от одной вершины к другой, то такой граф называется ***ориентированным.*** Ребра ориентированного графа называются ***дугами***. Соответствующие вершины ориентированного графа называют ***началом*** и ***концом.*** Если направления ребер не указываются, то граф называется ***неориентированным*** (или просто графом).

Пример 2.1. На рис. 2.1 изображен неориентированный граф $G = (X, A)$.

$X = \{x_1, x_2, x_3, x_4\}$,

$A = \{a_1 = (x_1, x_2), a_2 = (x_2, x_3), a_3 = (x_1, x_3), a_4 = (x_3, x_4)\}$.

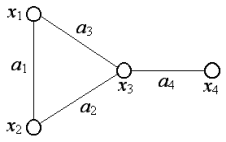


Рисунок 2.1 – Неориентированный граф

Пример 2.2. На рис. 2.2. изображен ориентированный граф $G = (X, A)$. $X = \{x_1, x_2, x_3, x_4\}$, $A = \{a_1 = (x_1, x_2), a_2 = (x_1, x_3), a_3 = (x_3, x_4), a_4 = (x_3, x_2)\}$.

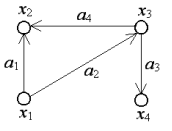


Рисунок 2.2 – Ориентированный граф

Граф, имеющий как ориентированные, так и неориентированные ребра, называется ***смешанным***.

Различные ребра могут соединять одну и ту же пару вершин. Такие ребра называют ***кратными.*** Граф, содержащий кратные ребра, называется ***мультиграфом***.

Неориентированное ребро графа эквивалентно двум противоположно направленным дугам, соединяющим те же самые вершины.

Ребро может соединять вершину саму с собой. Такое ребро называется ***петлей***. Граф с кратными ребрами и петлями называется ***псевдографом***.

Множество ребер графа может быть пустым. Множество вершин графа не может быть пустым.

Как в случае ориентированного, так и в случае неориентированного ребра говорят, что вершины x и y ***инцидентны*** ребру a, если эти вершины соединены a.

Две вершины называются ***смежными***, если они инцидентны одному и тому же ребру. Два ребра называются ***смежными***, если они имеют общую вершину.

Степенью вершины графа называется число ребер, инцидентных этой вершине. Вершина, имеющая степень 0, называется ***изолированной***, а степень 1 – ***висячей***.

Для ориентированного графа множество вершин, в которые ведут дуги, исходящие из вершины x, обозначают $G(x)$, то есть $G(x) = \{y: (x,y) \in G\}$. Множество $G(x)$ называют ***образом*** вершины x. Соответственно, $G^{-1}(y)$ – множество вершин, из которых исходят дуги, ведущие в вершину y, $G^{1}(y) = \{x: (x, y) \in G\}$. Множество $G^{-1}(y)$ называют ***прообразом*** вершины y.

Пример 2.3. В графе, изображенном на рис. 2.1, концами ребра a_1 являются вершины x_1, x_2; вершина x_2 инцидентна ребрам a_1, a_2; степень вершины x_3 равна 3; вершины x_1 и x_3 смежные; ребра a_1 и a_2 смежные; вершина x_4 висячая. В ориентированном графе, изображенном на рис. 2.2, началом дуги a_1 является вершина x_1, а ее концом – вершина x_2; вершина x_1 инцидентна дугам a_1 и a_2; $G(x_1) = \{x_2, x_3\}$, $G(x_2) = \varnothing$, $G^{-1}(x_3) = \{x_1\}$, $G^{-1}(x_1) = \varnothing$.

Подграфом неориентированного графа G называется граф, все вершины и ребра которого содержатся среди вершин и ребер графа G. Аналогично определяется подграф ориентированного графа. Подграф называется *собственным*, если он отличен от самого графа.

Граф $G = (X, A)$ – ***полный***, если для любой пары вершин x_i и x_j существует ребро (x_i, x_j).

Граф $G = (X, A)$ – ***симметрический***, если для любой дуги (x_i, x_j) существует противоположно ориентированная дуга(x_j, x_i).

Граф $G = (X, A)$ – ***планарный***, если он может быть изображен на плоскости так, что не будет пересекающихся дуг.

Неориентированный граф $G = (X, A)$ – ***двудольный***, если множество его вершин X можно разбить на два такие подмножества X_1 и X_2, что каждое ребро имеет один конец в X_1, а другой – в X_2.

Графы $G_1 = (X_1, A_1)$ и $G_2 = (X_2, A_2)$ ***изоморфны*** (рис. 2.3), если существует взаимно однозначное соответствие между множествами вершин X_1 и X_2 такое, что любые две вершины одного графа соединены тогда и только тогда, когда соответствующие вершины соединены в другом графе.

Изоморфные графы отличаются только нумерацией вершин. Матрицы смежности двух изоморфных графов могут быть получены одна из другой перестановкой строк и столбцов. Чтобы узнать, являются ли два графа

изоморфными, нужно произвести все возможные перестановки строк и столбцов матрицы смежности одного из графов. Если после какой-нибудь перестановки получится матрица смежности второго графа, то эти графы изоморфны. Чтобы убедиться, что графы неизоморфны, надо выполнить все $n!$ возможных перестановок строк и столбцов.

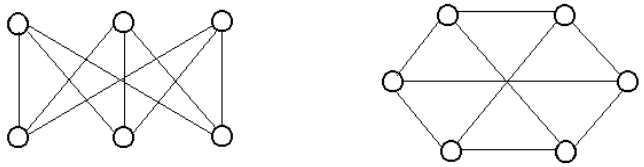


Рисунок 2.3 – Изоморфные графы

2.2 Матричные способы задания графов

Для алгебраического задания графов используются матрицы смежности и инцидентности.

Матрица смежности $\boldsymbol{A = (a_{ij})}$ определяется одинаково для ориентированного и неориентированного графов. Это квадратная матрица порядка n, где n – число вершин, у которой

$$a_{ij} = \begin{cases} 1, \text{ если } (x_i x_j) \in A, \\ 0, \text{ если } (x_i x_j) \notin A. \end{cases}$$

Пример 2.4. Матрица смежности графа, изображенного на рис. 2.1, имеет вид: $A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$.

Пример 2.5. Матрица смежности ориентированного графа, изображенного на рис. 2.2, имеет вид: $A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$.

Матрица смежности полностью задает граф.

Матрицей инцидентности $B = (b_{ij})$ ориентированного графа называется прямоугольная матрица $(n \times m)$, где n – число вершин, m – число ребер, у которой $b_i = \begin{cases} 1, \text{если вершина } x_i \text{ является началом дуги } a_j; \\ -1, \text{если вершина } x_i \text{ является концом дуги } a_j; \\ 0, \text{если вершина } x_i \text{ не инцидентна дуге } a_j. \end{cases}$

Для неориентированного графа матрица инцидентности B задается следующим образом: $b_i = \begin{cases} 1, \text{если вершина } x_i \text{ инцидентна ребру } a_j; \\ 0, \text{если вершина } x_i \text{ не инцидентна ребру } a_j. \end{cases}$

Пример 2.6. Матрица инцидентности графа, изображенного на рис. 2.1, имеет вид: $B = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$.

Пример 2.7. Матрица инцидентности ориентированного графа, изображенного на рис. 2.2, имеет вид: $B = \begin{pmatrix} 0 & 1 & 1 & 0 \\ -1 & 0 & -1 & 0 \\ -1 & 1 & 0 & 1 \\ 0 & 0 & -1 & 0 \end{pmatrix}$.

Матрица инцидентности, также как и матрица смежности, полностью задает граф.

Матрицы смежности и инцидентности удобны для задания графов на ЭВМ.

Основные свойства матриц смежности и инцидентности

1. Матрица смежности неориентированного графа является симметричной.

2. Сумма элементов *i*-ой строки или *i*-го столбца матрицы смежности неориентированного графа равна степени вершины x_i.

3. Сумма элементов *i*-ой строки матрицы смежности ориентированного графа равна числу дуг, исходящих из x_i.

4. Сумма элементов *i*-го столбца матрицы смежности ориентированного графа равна числу дуг, входящих в вершину x_i.

5. Сумма строк матрицы инцидентности ориентированного графа является нулевой строкой.

Таким образом, способами задания графа являются:

а) графическое изображение;

б) указание множества вершин и множества ребер (дуг);

в) матрица смежности;

г) матрица инцидентности.

2.3 Маршруты, циклы в неориентированном графе

Пусть G – неориентированный граф. ***Маршрутом*** или ***цепью*** в G называется такая последовательность (конечная или бесконечная) ребер a_1, a_2, ..., a_n, что каждые соседние два ребра a_i и a_{i+1} имеют общую инцидентную вершину. Одно и то же ребро может встречаться в маршруте несколько раз. В конечном маршруте (a_1, a_2, ..., a_n) имеется первое ребро a_1

и последнее ребро a_n. Вершина x_1, инцидентная ребру a_1, но не инцидентная ребру a_2, называется началом маршрута, а вершина x_n, инцидентная ребру a_n, но не инцидентная ребру a_{n-1}, называется концом маршрута.

Длиной (или ***мощностью***) маршрута называется число ребер, входящих в маршрут, причем каждое ребро считается столько раз, сколько оно входит в данный маршрут.

Замкнутый маршрут называется ***циклом***.

Маршрут (цикл), в которой все ребра различны, называется ***простой цепью*** (циклом). Маршрут (цикл), в которой все вершины, (кроме первой и последней), различны, называется ***элементарной цепью*** (циклом).

Пример 2.8. В приведенном на рис. 2.4 графе выделим следующие маршруты:

(a_1, a_3, a_4) – простая элементарная цепь длины 3, т. к. все ребра и вершины попарно различны;

(a_2, a_4, a_3) – простой элементарный цикл, т. к. это замкнутый маршрут, у которого все ребра и вершины, кроме первой и последней, различны;

(a_1, a_2, a_4, a_3) – цепь, которая является простой, но не элементарной, т. к. все ребра различны, но вершина x_2 встречается дважды;

(a_1, a_2, a_2) – маршрут длины 3, не являющийся ни простой, ни элементарной цепью, т. к. ребро a_2 и вершина x_2 встречаются дважды.

Рассмотрим два маршрута из вершины x_1 в вершину x_4: $M_1 = (a_1, a_2, a_4)$ и $M_2 = (a_1, a_3)$. Длина маршрута M_1 равна 3, а длина маршрута M_2 равна 2.

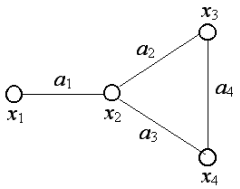


Рисунок 2.4 – Пример маршрутов, циклов в неориентированном графе

Цепь (цикл) в графе G называется ***гамильтоновой (гамильтоновыми)***, если она (он) проходит по одному разу через каждую вершину псевдографа G. Граф является ***гамильтоновым***, если он содержит гамильтонов цикл.

Цепь (цикл) в G называется ***эйлеровой (эйлеровым)***, если она (он) проходит по одному разу через каждое ребро псевдографа G. Граф является ***эйлеровым***, если он содержит эйлеров цикл.

Гамильтоновы и эйлеровы циклы содержатся как в неориентированном, так и в ориентированном графе.

2.4 Пути, контуры в ориентированном графе

Понятия пути, контура в ориентированном графе аналогичны понятиям маршрута, цикла в неориентированном графе.

Путем ориентированного графа называется последовательность дуг, в которой конечная вершина всякой дуги, отличной от последней, является начальной вершиной следующей дуги.

Число дуг пути называется ***длиной*** пути.

Путь называется ***контуром***, если его начальная вершина совпадает с конечной вершиной**.**

Путь (контур), в котором все дуги различны, называется ***простым***.

Путь (контур), в котором все вершины, кроме первой и последней, различны, называется ***элементарным***.

Следует усвоить, что понятиям ребра, маршрута, цепи, цикла в неориентированном графе соответствуют понятия дуги, пути, ориентированной цепи, контура в ориентированном графе.

Пример 2.9. В приведенном на рис 2.5 графе выделим следующие пути:

(x_1, x_2, x_3, x_4) – простой элементарный путь, т. к. каждая вершина и каждая дуга используются не более одного раза;

(x_2, x_5, x_6, x_7, x_2) – простой элементарный контур, т. к. это замкнутый путь, в котором все дуги и вершины, кроме первой и последней, различны.

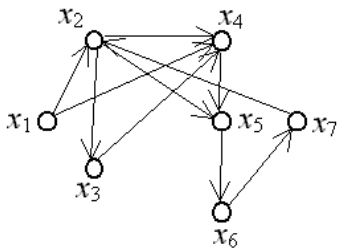


Рисунок 2.5 – Пример пути, контура в ориентированном графе

2.5 Связность графа

Неориентированный граф называется ***связным***, если каждая пара различных вершин может быть соединена хотя бы одной цепью.

Ориентированный граф называется ***сильно связным***, если для любых двух его вершин x_i и x_j существует хотя бы один путь, соединяющий x_i с x_j.

Ориентированный граф называется ***односторонне связным***, если для любых двух его вершин хотя бы одна достижима из другой.

Компонентой связности неориентированного графа называется его связный подграф, не являющийся собственным подграфом никакого другого связного подграфа данного графа (максимально связный подграф).

Компонентой сильной связности ориентированного графа называется его сильно связный подграф, не являющийся собственным подграфом никакого другого сильно связного подграфа данного графа (максимально сильно связный подграф).

Компонентой односторонней связности неориентированного графа называется его односторонне связный подграф, не являющийся собственным подграфом никакого другого односторонне связного подграфа данного графа (максимально односторонне связный подграф).

Пусть $G = (X, A)$ неориентированный граф с множеством вершин $X = \{x_1, ..., x_n\}$. Квадратная матрица $S = (s_{ij})$ порядка n, у которой

$$s_{ij} = \begin{cases} 1, & \text{если вершины } x_i \text{ и } x_j \text{ принадлежат одной компоненте связности,} \\ 0 & \text{в противном случае.} \end{cases}$$

называется ***матрицей связности*** графа G.

Для ориентированного графа квадратная матрица $T = (t_{ij})$ порядка n, у которой

$$t_{ij} = \begin{cases} 1, & \text{если существует путь из } x_i \text{ в } x_j, \\ 0, & \text{в противном случае.} \end{cases}$$

называется ***матрицей односторонней связности (достижимости).***

Квадратная матрица $S = (s_{ij})$ порядка n, у которой

$$s_{ij} = \begin{cases} 1, & \text{если существует путь из } x_i \text{ в } x_j \text{ и из } x_j \text{ в } x_i, \\ 0, & \text{в противном случае.} \end{cases}$$

называется ***матрицей сильной связности.***

Пример 2.10. У неориентированного графа, изображенного на рис. 2.6, две компоненты связности. Первая компонента связности включает вершины x_1, x_2, x_4, x_5, а вторая состоит из одной вершины x_3.

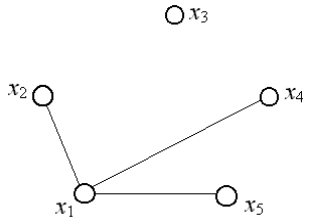


Рисунок 2.6 – Неориентированный граф с двумя компонентами связности

Матрица связности этого графа имеет вид: $S = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{pmatrix}$,

1-я, 2-ая, 4-ая и 5-ая строки матрицы S одинаковы.

Пример 2.11. У ориентированного графа, изображенного на рис. 2.7, две компоненты сильной связности. Первая компонента связности включает вершины x_1, x_2, x_3, x_5, а вторая состоит из одной вершины x_4. Действительно, для любой пары вершин из множества $\{x_1, x_2, x_3, x_5\}$ существует хотя бы один путь, соединяющий эти вершины. Например, путь (x_1, x_2, x_5, x_3, x_1) соединяет все эти вершины. Из вершины x_4 нет пути ни в одну вершину графа.

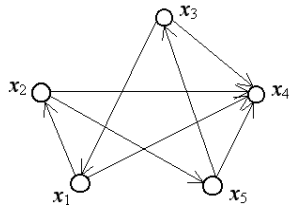


Рисунок 2.7 – Ориентированный граф с двумя компонентами сильной связности

Матрица сильной связности этого графа имеет вид: $S = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \end{pmatrix}$,

1-я, 2-ая, 3-ая и 5-ая строки матрицы S одинаковы.

2.6 Экстремальные пути в нагруженных ориентированных графах. Алгоритм Форда – Беллмана нахождения минимального пути

Ориентированный граф называется ***нагруженным,*** если дугам этого графа поставлены в соответствие веса, так что дуге (x_i, x_j) сопоставлено некоторое число $c(x_i, x_j) = c_{ij}$, называемое ***длиной*** **(**или ***весом,*** или ***стоимостью*** дуги**)**. ***Длиной*** (или ***весом,*** или ***стоимостью***) пути s, состоящего из некоторой последовательности дуг (x_i, x_j), называется число $l(s)$, равное сумме длин дуг, входящих в этот путь, т.е. $l(s) = \sum c_{ij}$, причем суммирование ведется по всем дугам $(x_i, x_j) \in s$.

Матрица $C = (c_{ij})$ называется ***матрицей длин дуг*** или ***матрицей весов.***

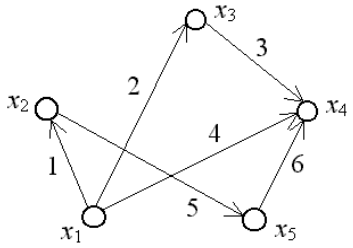


Рисунок 2.8 – Нагруженный ориентированный граф

Для графа, изображенного на рис. 2.8, матрица C имеет вид:

$$C = \begin{pmatrix} \infty & 1 & 2 & 4 & \infty \\ \infty & \infty & \infty & \infty & 5 \\ \infty & \infty & \infty & 3 & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 6 & \infty \end{pmatrix}.$$

Длина пути (x_1, x_2, x_5, x_4) равна 1 + 5 + 6 = 12.

Для ненагруженного графа введем понятие ***кратчайшего пути***. Это путь с минимальным общим числом дуг, причем каждая дуга считается столько раз, сколько она содержится в этом пути.

Для нахождения минимального пути между двумя произвольными вершинами для случая, когда все $c_{ij} \geq 0$ применяют алгоритм Дейкстры. В общем случае задача решается с помощью алгоритмов Флойда, Форда, Беллмана.

Алгоритмы нахождения минимального пути могут быть использованы для поиска кратчайших путей в ориентированном графе без контуров. Для этого нужно каждой дуге приписать вес, равный единице.

Рассмотрим алгоритм Форда – Беллмана для нахождения минимального пути в ориентированном графе. Предполагается, что ориентированный граф не содержит контуров отрицательной длины.

Основными вычисляемыми величинами этого алгоритма являются величины $l_j(k)$, где $i = 1, 2, \ldots, n$ (n – число вершин графа); $k = 1, 2, \ldots, n-1$. Для фиксированных i и k величина $l_j(k)$ равна длине минимального пути, ведущего из заданной начальной вершины x_1 в вершину x_i и содержащего не более k дуг.

1. Установка начальных условий. Ввести число вершин графа n и матрицу весов $C = (c_{ij})$.

2. Положить $k = 0$. Положить $l_i(0) = \infty$ для всех вершин, кроме x_1; положить $l_1(0) = 0$.

3. В цикле по k, $k = 1,..., n-1$, каждой вершине x_i на k-ом шаге приписать индекс $l_i(k)$ по следующему правилу:

$$l_i(k) = \min_{1 \leq j \leq n} \{l_j(k-1) + c_{ji}\} \qquad (2.1)$$

для всех вершин, кроме x_1, положить $l_1(k) = 0$.

В результате работы алгоритма формируется таблица индексов $l_i(k)$, $i = 1, 2, \ldots, n$, $k = 0, 1, 2, \ldots, n-1$. При этом $l_i(k)$ определяет длину минимального пути из первой вершины в i-ую, содержащего не более k дуг.

4. Восстановление минимального пути. Для любой вершины x_s предшествующая ей вершина x_r определяется из соотношения:

$$l_r(n-2) + c_{rs} = l_s(n-1), \qquad x_r \in G^{-1}(x_s), \qquad (2.2)$$

где $G^{-1}(x_s)$ – прообраз вершины x_s.

Для найденной вершины x_r предшествующая ей вершина x_q определяется из соотношения:

$$l_q(n-3) + c_{qr} = l_r(n-2), \qquad x_q \in G^{-1}(x_r), \qquad (2.3)$$

где $G^{-1}(x_r)$ – прообраз вершины x_r и т. д.

Последовательно применяя это соотношение, начиная от последней вершины x_i, найдем минимальный путь.

Пример 2.12. С помощью алгоритма Форда – Беллмана найдем минимальный путь из вершины x_1 в вершину x_3 в графе, изображенном на рис. 2.9.

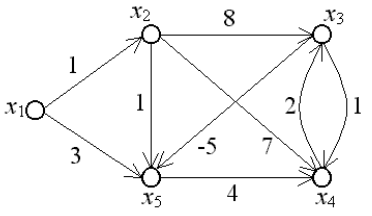


Рисунок 2.9 – Пример графа для нахождения минимального пути из вершины x_1 в вершину x_3

Рассмотрим работу алгоритма Форда – Беллмана для этого примера. Значения индексов $l_i(k)$ будем заносить в таблицу индексов (табл. 2.1).

1. Введем число вершин графа $n = 5$. Матрица весов этого графа имеет вид: $C = \begin{pmatrix} \infty & 1 & \infty & \infty & 3 \\ \infty & \infty & 8 & 7 & 1 \\ \infty & \infty & \infty & 1 & -5 \\ \infty & \infty & 2 & \infty & \infty \\ \infty & \infty & \infty & 4 & \infty \end{pmatrix}$.

2. Положим $k = 0$, $l_1(0) = 0$, $l_2(0) = l_3(0) = l_4(0) = l_5(0) = \infty$. Эти значения занесем в первый столбец табл. 2.1.

3. $\underline{k = 1}$.

$l_1(1) = 0$.

Равенство (2.1) для $k = 1$ имеет вид:

$l_i(1) = \min\limits_{1 \le j \le 5} \{l_j(0) + c_{ji}\}$.

$l_2(1) = min\{l_1(0) + c_{12};\ l_2(0) + c_{22};\ l_3(0) + c_{32};\ l_4(0) + c_{42};\ l_5(0) + c_{52};\} = min\{0 + 1;\ \infty + \infty;\ \infty + \infty;\ \infty + \infty;\ \infty + \infty\} = 1.$

$l_3(1) = min\{l_1(0) + c_{13};\ l_2(0) + c_{23};\ l_3(0) + c_{33};\ l_4(0) + c_{43};\ l_5(0) + c_{53};\} = min\{0 + \infty\ ;\ \infty + 8;\ \infty + \infty;\ \infty + 2;\ \infty + \infty\} = \infty\ .$

$l_4(1) = min\{l_1(0) + c_{14};\ l_2(0) + c_{24};\ l_3(0) + c_{34};\ l_4(0) + c_{44};\ l_5(0) + c_{54};\} = min\{0 + \infty\ ;\ \infty + 7;\ \infty + 1;\ \infty + \infty;\ \infty + 4\} = \infty\ .$

$l_5(1) = min\{l_1(0) + c_{15};\ l_2(0) + c_{25};\ l_3(0) + c_{35};\ l_4(0) + c_{45};\ l_5(0) + c_{55};\} = min\{0 + 3;\ \infty + 1;\ \infty\ - 5;\ \infty + \infty;\ \infty + \infty\} = 3.$

Полученные значения $l_i(1)$ занесем во второй столбец табл. 2.1. Убеждаемся, что второй столбец, начиная со второго элемента, совпадает с первой строкой матрицы весов, что легко объясняется смыслом величин $l_i(1)$, которые равны длине минимального пути из первой вершины в i-ую, содержащего не более одной дуги.

$\underline{k = 2.}$

$l_1(2) = 0.$

Равенство (2.1) для $k = 2$ имеет вид:

$l_i(2) = \min\limits_{1 \le j \le 5} \{l_j(1) + c_{ji}\}.$

$l_2(2) = min\{0 + 1;\ 1 + \infty; \infty + \infty; \infty + \infty; 3 + \infty\} = 1.$

$l_3(2) = min\{0 + \infty\ ; 1 + 8; \infty + \infty; \infty + 2; 3 + \infty\} = 9.$

$l_4(2) = min\{0 + \infty\ ; 1 + 7; \infty + 1; \infty + \infty; 3 + 4\} = 7.$

$l_5(2) = min\{0 + 3;\ \ 1 + 1; \infty - 5; \infty + \infty; 3 + \infty\} = 2.$

Полученные значения $l_i(2)$ занесем в третий столбец табл. 2.1. Величины $l_i(2)$ равны длине минимального пути из первой вершины в i-ую, содержащего не более двух дуг.

$\underline{k = 3.}$

$l_1(3) = 0.$

Равенство (2.1) для $k = 3$ имеет вид:

$l_i(3) = \min\limits_{1 \le j \le 5} \{l_j(2) + c_{ji}\}.$

$l_2(3) = min\{0 + 1;\ 1 + \infty; 9 + \infty; 7 + \infty; 2 + \infty\} = 1.$

$l_3(3) = min\{0 + \infty\ ; 1 + 8; 9 + \infty; 7 + 2; 2 + \infty\} = 9.$

$l_4(3) = min\{0 + \infty\ ; 1 + 7; 9 + 1; 7 + \infty; 2 + 4\} = 6.$

$l_5(3) = min\{0 + 3;\ \ 1 + 1; 9 - 5; 7 + \infty; 2 + \infty\} = 2.$

Полученные значения $l_i(3)$ занесем в четвертый столбец табл. 2.1. Величины $l_i(3)$ равны длине минимального пути из первой вершины в i-ую, содержащего не более трех дуг.

$\underline{k = 4.}$

$l_1(4) = 0.$

Равенство (2.1) для $k = 4$ имеет вид:

$l_i(4) = \min\limits_{1 \le j \le 5} \{l_j(3) + c_{ji}\}.$

$l_2(4) = min\{0 + 1;\ 1 + \infty; 9 + \infty; 6 + \infty; 2 + \infty\} = 1.$

$l_3(4) = min\{0 + \infty; 1 + 8; 9 + \infty; 6 + 2; 2 + \infty\} = 8\ .$

$l_4(4) = min\{0 + \infty; 1 + 7; 9 + 1; 6 + \infty; 2 + 4\} = 6\ .$

$l_5(4) = min\{0 + 3; 1 + 1; 9 - 5; 6 + \infty; 2 + \infty\} = 2.$

Полученные значения $l_i(4)$ занесем в пятый столбец табл. 2.1. Величины $l_i(4)$ равны длине минимального пути из первой вершины в i-ую, содержащего не более четырех дуг.

Таблица 2.1 – Индексы $l_i(k)$ для определения минимального пути

i (номер вершины)	$l_i(0)$	$l_i(1)$	$l_i(2)$	$l_i(3)$	$l_i(4)$
1	0	0	0	0	0
2	∞	1	1	1	1
3	∞	∞	9	9	8
4	∞	∞	7	6	6
5	∞	3	2	2	2

4. Восстановление минимального пути.

Для последней вершины x_3 предшествующая ей вершина x_r определяется из соотношения (2.2), полученного при s =3: $l_r(3) + c_{r3} = l_3(4)$, $x_r \in G^{-1}(x_3)$, $G^{-1}(x_3)$ – прообраз вершины x_3. $G^{-1}(x_3)= \{x_2, x_4\}$. Подставим последовательно $r = 2$ и $r = 4$, чтобы определить, для какого r это равенство выполняется: $l_2(3) + c_{23} = 1 + 8 \neq l_3(4) = 8$, $l_4(3) + c_{43} = 6 + 2 = l_3(4) = 8$.

Таким образом, вершиной, предшествующей вершине x_3, является вершина x_4.

Для вершины x_4 предшествующая ей вершина x_r определяется из соотношения (2.2), полученного при s =4: $l_r(2) + c_{r4} = l_4(3)$, $x_r \in G^{-1}(x_4)$, $G^{-1}(x_4)$ – прообраз вершины x_4. $G^{-1}(x_4)= \{x_2, x_3, x_5\}$. Подставим последовательно $r = 2$, $r = 3$ и $r = 5$, чтобы определить, для какого r это равенство выполняется: $l_2(2) + c_{24} = 1 + 7 \neq l_4(3) = 6$, $l_3(2) + c_{34} = 1 + 1 \neq l_4(3) = 6$, $l_5(2) + c_{54} = 2 + 4 = l_4(3) = 6$.

Таким образом, вершиной, предшествующей вершине x_4, является вершина x_5.

Для вершины x_5 предшествующая ей вершина x_r определяется из соотношения (2.2) полученного при s =5: $l_r(1) + c_{r5} = l_5(2)$, $x_r \in G^{-1}(x_5)$, $G^{-1}(x_5)$ – прообраз вершины x_5. $G^{-1}(x_5)= \{x_1, x_2\}$. Подставим последовательно $r = 1$ и $r = 2$, чтобы определить, для какого r это равенство выполняется: $l_1(1) + c_{15} = 0 + 3 \neq l_5(2) = 2$, $l_2(1) + c_{25} = 1 + 1 = l_5(2) = 2$,

Таким образом, вершиной, предшествующей вершине x_5, является вершина x_2.

Для вершины x_2 предшествующая ей вершина x_r определяется из соотношения (2.2), полученного при s = 2: $l_r(0) + c_{r2} = l_2(1)$, $x_r \in G^{-1}(x_2)$, $G^{-1}(x_2)$ – прообраз вершины x_2. $G^{-1}(x_2) = \{x_1\}$. Подставим $r = 1$, чтобы определить, выполняется ли это равенство: $l_1(0) + c_{12} = 0 + 1 = l_2(1) = 1$.

Таким образом, вершиной, предшествующей вершине x_2, является вершина x_1.

Итак, найден минимальный путь – x_1, x_2, x_5, x_4, x_3, его длина равна 8.

2.7 Деревья. Минимальные остовные деревья нагруженных графов

Неориентированным деревом (или просто ***деревом***) называется связный граф без циклов. Этому определению эквивалентны следующие определения:

а) дерево есть связный граф, содержащий n вершин и n-1 ребер;

б) дерево есть граф, любые две вершины которого можно соединить простой цепью.

Пример 2.13. Графы, изображенные на рис. 2.10, являются деревьями.

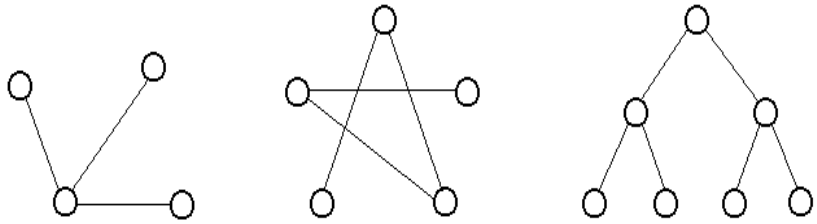
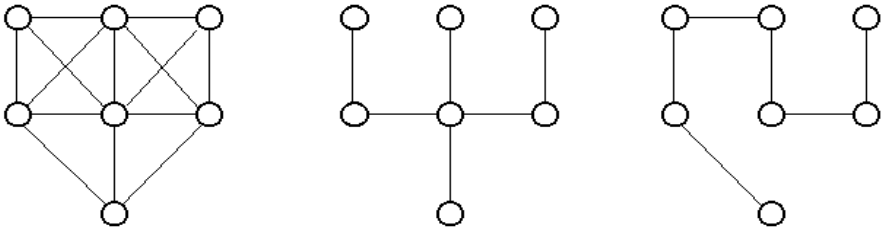


Рисунок 2.10 – Деревья

Если граф несвязный и не имеет циклов, то каждая его связная компонента будет деревом. Такой граф называется ***лесом***. Можно интерпретировать рис. 2.10 как лес, состоящий из трех деревьев.

Остовным деревом связного графа G называется любой его подграф, содержащий все вершины графа G и являющийся деревом.

Пример 2.14. Для графа, изображенного на рис. 2.11, а, графы на рис. 2.11, б и 2.11, в являются остовными деревьями.



а) *б)* *в)*

а) неориентированный граф; б), в) остовные деревья

Рисунок 2.11 – Граф и остовные деревья

Пусть граф G имеет n вершин и m ребер. Поскольку всякое дерево с n вершинами по определению имеет $n - 1$ ребер, то любое остовное дерево графа G получается из этого графа в результате удаления $m - (n - 1) = m - n + 1$ ребер. Число $\gamma = m - n + 1$ называется ***цикломатическим числом*** графа.

Граф $G = (X, A)$ называется ***нагруженным***, если для каждого ребра (x_i, x_j) определена его длина (или вес) c_{ij}. Пусть G – связный нагруженный граф. Задача построения ***минимального остовного дерева*** заключается в том, чтобы из множества остовных деревьев найти дерево, у которого сумма длин ребер минимальна.

Задачу построения минимального остовного дерева можно решить с помощью алгоритма Краскала.

1. Установка начальных значений. Вводится матрица длин ребер C графа G.

2. Выбрать в графе G ребро минимальной длины. Построить граф G_2, состоящий из данного ребра и инцидентных ему вершин. Положить $i = 2$.

3. Если $i = n$, где n – число ребер графа, то закончить работу (задача решена), в противном случае перейти к пункту 4.

4. Построить граф $G_{i+1,}$ добавляя к графу G_i новое ребро минимальной длины, выбранное среди всех ребер графа G, каждое из которых инцидентно какой-нибудь вершине графа G_i и одновременно инцидентно какой-нибудь вершине графа G, не содержащейся в G_i. Вместе с этим ребром включаем в G_{i+1}и инцидентную ему вершину, не содержащуюся в G_i. Присваиваем $i: = i + 1$ и переходим к пункту 3.

Пример 2.15. Найдем минимальное остовное дерево для графа G, изображенного на рис. 2.12.

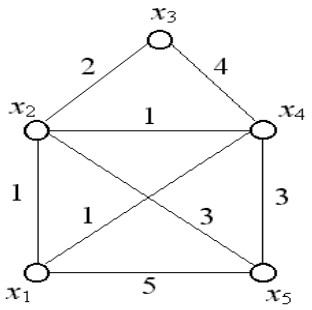


Рисунок 2.12 – Нагруженный граф G

1. Установка начальных значений. Введем матрицу длин ребер C:

$$C = \begin{pmatrix} \infty & 1 & \infty & 1 & 5 \\ 1 & \infty & 2 & 1 & 3 \\ \infty & 2 & \infty & 4 & \infty \\ 1 & 1 & 4 & \infty & 3 \\ 5 & 3 & \infty & 3 & \infty \end{pmatrix}.$$

2. Выберем ребро минимальной длины. Минимальная длина ребра равна единице. Таких ребер три: (x_1, x_2), (x_1, x_4), (x_2, x_4). В этом случае можно взять любое. Возьмем (x_1, x_2). Построим граф G_2, состоящий из данного ребра и инцидентных ему вершин x_1 и x_2. Положим $i = 2$.

3. Так как $n = 5$, то $i \neq n$, поэтому переходим к пункту 4.

4. Строим граф $G_{3,}$ добавляя к графу G_2 новое ребро минимальной длины, выбранное среди всех ребер графа G, каждое из которых инцидентно одной из вершин x_1, x_2 и одновременно инцидентно какой-нибудь вершине графа G, не содержащейся в G_2, т. е. одной из вершин x_3, x_4, x_5. Таким образом, нужно выбрать ребро минимальной длины из ребер (x_1, x_4), (x_1, x_5), (x_2, x_3), (x_2, x_4), (x_2, x_5). Таких ребер длины единица два: (x_1, x_4) и (x_2, x_4). Можно выбрать любое. Возьмем (x_1, x_4). Вместе с этим ребром включаем в G_3 вершину x_4, не содержащуюся в G_2. Полагаем i =3 и переходим к пункту 3.

3. Так как $i \neq n$, поэтому переходим к пункту 4.

4. Строим граф G_4, добавляя к графу G_3 новое ребро минимальной длины из ребер (x_1, x_5), (x_2, x_3), (x_2, x_5), (x_4, x_5). Такое ребро длины два одно: (x_2, x_3). Вместе с этим ребром включаем в G_4 вершину x_3, не содержащуюся в G_3. Полагаем $i = 4$ и переходим к пункту 3.

3. Так как $i \neq n$, поэтому переходим к пункту 4.

4. Строим граф G_5, добавляя к графу G_3 новое ребро минимальной длины из ребер (x_1, x_5), (x_2, x_5), (x_4, x_5). Таких ребер, имеющих длину три, – два: (x_2, x_5) и (x_4, x_5). Возьмем (x_2, x_5). Вместе с этим ребром включаем в G_5 вершину x_5, не содержащуюся в G_4. Полагаем $i = 5$ и переходим к пункту 3.

3. Так как $i = n$, то граф G_5 – искомое минимальное остовное дерево. Суммарная длина ребер равна 1 + 1 + 2 + 3 = 7.

Процесс построения минимального остовного дерева изображен на рис. 2.13.

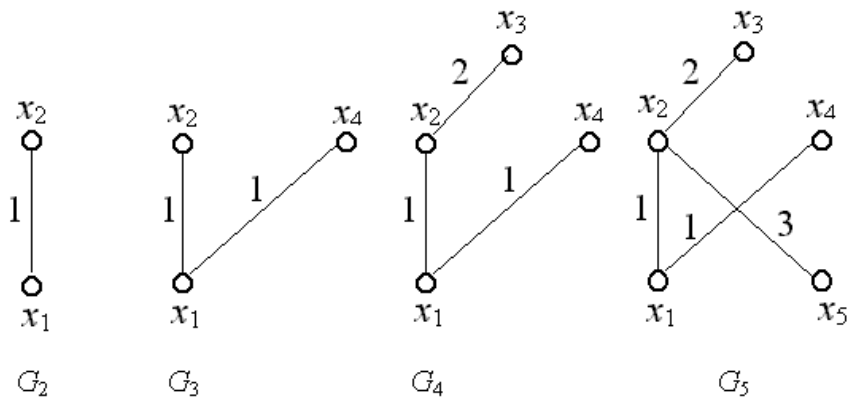


Рисунок 2.13 – Процесс построения остовного дерева

Контрольные вопросы к разделу 2.

1. Перечислите основные способы представления графов.
2. Чему равна сумма степеней всех вершин неориентированного графа?
3. В чем отличия матричного представления ориентированных и неориентированных графов?
4. В чем особенности представления графа матрицей смежности, матрицей инцидентности?
5. В чем особенности представления графа?
6. Дайте определение пути, маршрута, цепи, контура.
7. Какой граф называется нагруженным?
8. Алгоритм нахождения кратчайшего пути на графе.
9. Дайте определение дерева; ориентированного дерева.
10. Какое дерево называется остовным?

3 ЭЛЕМЕНТЫ КОМБИНАТОРНОГО АНАЛИЗА

3.1 Основные понятия и теоремы комбинаторики

Комбинаторика – это раздел математики, посвященный задаче выбора и расположения элементов некоторого конечного множества в соответствии с заданными правилами. Каждое такое правило определяет способ построения некоторой конструкции из элементов исходного множества, называемой комбинаторной конфигурацией. Целью комбинаторики является изучение комбинаторных конфигураций. Одной из основных задач комбинаторного анализа (комбинаторики) является подсчет числа элементов конечных множеств, заданных каким-либо описательным условием. Для этого разработаны различные формулы и правила.

Пусть имеется k групп $A_1, A_2, ..., A_k$, причем i-ая группа содержит n_i элементов. Тогда справедливы следующие правила.

Теорема умножения (основная формула комбинаторики). Общее число N способов, которыми можно получить упорядоченную совокупность $(a_1, a_2, ..., a_k)$, где $a_i \in A_i$, т. е. выбрать по одному элементу из каждой группы и расставить их в определенном порядке, равно

$$N = n_1 \cdot n_2 \cdot ... \cdot n_k . \qquad (3.1)$$

Это правило распространяется и на ситуации, когда новые группы образуются в процессе выбора элементов, если численности этих групп не зависят от того, какие именно элементы были выбраны.

Пример 3.1. В группе 30 студентов. Необходимо выбрать старосту, заместителя старосты и профорга. Сколько существует способов это сделать? Старостой может быть выбран любой из 30 студентов, заместителем – любой из оставшихся 29, а профоргом – любой из оставшихся 28 студентов, т. е. $n_1 = 30$, $n_2 = 29$, $n_3 = 28$. По правилу умножения общее число N способов выбора старосты, его заместителя и профорга равно $N = n_1 \cdot n_2 \cdot n_3 = 30 \cdot 29 \cdot 28 = 24360$.

Теорема сложения. Если один элемент из группы A_i можно выбрать n_i способами, и при этом любые две группы A_i и A_j не имеют общих элементов, то выбор одного элемента или из A_1, или из A_2, ..., или из A_k можно осуществить N способами:

$$N = n_1 + n_2 + ... + n_k . \qquad (3.2)$$

Пример 3.2. В ящике 100 деталей, из них 30 деталей 1-го сорта, 50 – 2-го, остальные – 3-го. Сколько существует способов извлечения из ящика одной детали 1-го или 2-го сорта? Деталь 1-го сорта может быть извлечена

$n_1 = 30$ способами, 2-го сорта – $n_2 = 50$ способами. По правилу суммы существует $N = n_1 + n_2 = 30 + 50 = 80$ способов извлечения одной детали 1-го или 2-го сорта.

3.2 Упорядоченные совокупности (последовательный выбор)

Пусть имеется некоторая конечная совокупность элементов $\{a_1, a_2, ..., a_n\}$, называемая генеральной совокупностью, и n – объем этой совокупности. Пусть эксперимент состоит в том, что из генеральной совокупности последовательно выбирают k элементов и располагают их в порядке выбора. Возможны две ситуации.

Размещения без повторений. Отобранный элемент перед отбором следующего не возвращается в генеральную совокупность. Такой выбор называется *размещением k элементов из n* (или *последовательным выбором без возвращения*).

Размещения – это упорядоченные совокупности k элементов из n, отличающиеся друг от друга либо составом, либо порядком элементов.

Пример 3.3. Пусть имеется множество $\{a, b, c\}$ из трех элементов. Тогда все размещения двух элементов из трех таковы: *ab, ba, ac, ca, bc, cb*.

Число различных способов, которыми можно произвести последовательный выбор без возвращения k элементов из генеральной совокупности объема n, равно

$$A_n^k = \frac{n!}{(n-k)!}. \qquad (3.3)$$

В частном случае, когда выбираются все элементы генеральной совокупности, т. е. когда $k = n$, размещения называются ***перестановками.*** Их число обозначается P_n.

Перестановки – это упорядоченные совокупности, отличающиеся друг от друга только порядком элементов. Число всех перестановок множества из n элементов равно

$$P_n = n! \qquad (3.4)$$

Пример 3.4. Все перестановки множества $\{a, b, c\}$ из трех элементов устроены так: *abc, bac, cba, acb, cab, bca* и $P_3 = 3! = 6$.

Размещения с повторениями. Если каждый отобранный элемент перед отбором следующего возвращается в генеральную совокупность, то такой выбор называется *размещением с повторениями* (или *последовательным выбором с возвращением*).

Общее число различных способов, которыми можно произвести выбор с возвращением *k* элементов из генеральной совокупности объема *n*, равно

$$\overline{A}_n^k = n^k \tag{3.5}$$

Пример 3.5. Все размещения с повторениями двух элементов из множества с тремя элементами {*a*, *b*, *c*}: *aa*, *ab*, *ac*, *ba*, *bb*, *bc*, *ca*, *cb*, *cc*.

Пример 3.6. Группа из 15 человек выиграла 3 различных книги. Сколькими способами можно распределить эти книги среди группы? Имеем: $A_{15}^3 = \frac{15!}{12!} = 15 \cdot 14 \cdot 13 = 2730.$

Пример 3.7. В конкурсе по 5 номинациям участвуют 10 кинофильмов. Сколько существует вариантов распределения призов, если по каждой номинации установлены **различные** премии? Каждый из вариантов распределения призов представляет собой комбинацию 5 фильмов из 10, отличающуюся от других комбинаций как составом, так и их порядком. Так как каждый фильм может получить призы как по одной, так и по нескольким номинациям, то одни и те же фильмы могут повторяться. Поэтому число таких комбинаций равно числу размещений с повторениями из 10 элементов по 5: $N = \overline{A}_{10}^5 = 10^5 = 100000.$

3.3 Неупорядоченные совокупности (одновременный выбор)

Сочетания без повторений. В результате *одновременного неупорядоченного выбора k* элементов из генеральной совокупности объема *n* получаются комбинации, которые называют *сочетаниями из n элементов по k.*

Сочетания – это неупорядоченные совокупности, отличающиеся друг от друга только составом элементов.

Пример 3.8. Все сочетания без повторений двух элементов из множества {*a*, *b*, *c*}: $\{a, b\}$, $\{a, c\}$, $\{b, c\}$.

Число сочетаний из *n* элементов по *k* равно:

$$C_n^k = \frac{A_n^k}{k!} = \frac{n!}{(n-k)! \cdot k!}. \tag{3.6}$$

Свойства числа сочетаний:

1. $C_n^0 = C_n^n = 1;$
2. $C_n^k = C_n^{n-k};$
3. $C_n^k + C_n^{k+1} = C_{n+1}^{k+1};$
4. $C_n^0 + C_n^1 + C_n^2 + \ldots + C_n^n = 2^n;$
5. $A_n^k = P_k \cdot C_n^k.$

Числа C_n^k называют также *биномиальными коэффициентами*, поскольку они участвуют в разложении бинома Ньютона: $(a+b)^n = \sum_{k=0}^{n} C_n^k \cdot a^k \cdot b^{n-k}$.

Сочетания с повторениями. Если в сочетаниях из *n* элементов по *k* некоторые из элементов или все могут оказаться одинаковыми, то такие сочетания называются *сочетаниями с повторениями из n элементов по k*.

Число сочетаний с повторениями из *n* элементов по *k* равно

$$\overline{C}_n^k = P(k, n-1) = \frac{(k+n-1)!}{k!\cdot(n-1)!} = C_{n+k-1}^k. \qquad (3.7)$$

Пример 3.9. Все сочетания с повторениями двух элементов из множества $\{a,b,c\}$: $\{a,a\}$, $\{a,b\}$, $\{a,c\}$, $\{b,b\}$, $\{b,c\}$, $\{c,c\}$.

Пример 3.10. В шахматном турнире участвуют 16 человек. Сколько партий должно быть сыграно в турнире, если между любыми участниками должна быть сыграна одна партия? Каждая партия играется двумя участниками из 16 и отличается от других только составом пар участников, т. е. представляет собой сочетания из 16 элементов по 2. Их число равно $C_{16}^2 = \frac{16!}{14!\cdot 2!} = \frac{15\cdot 16}{1\cdot 2} = 120.$

Пример 3.11. В условиях примера 3.7 определить, сколько существует вариантов распределения призов, если по каждой номинации установлены **одинаковые** призы? Если по каждой номинации установлены одинаковые призы, то порядок фильмов в комбинации 5 призов значения не имеет, и число вариантов представляет собой число сочетаний с повторениями из 10 элементов по 5, определяемое по формуле $\overline{C}_{10}^5 = C_{10+5-1}^5 = C_{14}^5 = \frac{10\cdot 11\cdot 12\cdot 13\cdot 14}{1\cdot 2\cdot 3\cdot 4\cdot 5} = 2002.$

3.4 Разбиение множества на группы

Пусть множество из *n* различных элементов разбивается на *k* групп так, что в первую группу попадают n_1 элементов, во вторую – n_2 элементов, в *k*-ую группу – n_k элементов, причем $n_1 + n_2 + ... + n_k = n$. Такую ситуацию называют *разбиением множества на группы.* Заметим, что порядок элементов при разбиении на группы не важен, а вот порядок групп (какую считать первой, какую – второй и т. д.) существенен.

Число разбиений рассчитывается по формуле:

$$N_n(n_1, n_2, ..., n_k) = \frac{n!}{n_1!\cdot n_2!\cdot ...\cdot n_k!}. \qquad (3.8)$$

Пример 3.12. Перечислим разбиения множества из 4 элементов *a, b, c, d* на 2 группы по 2 элемента (6 штук): [{*a, b*},{*c, d*}], [{*a, c*},{*b, d*}], [{*a, d*},{*b, c*}], [{*c, d*},{*a, b*}], [{*b, d*},{*a, c*}], [{*b, c*},{*a, d*}].

Пример 3.13.Сколькими способами можно разбить группу из 25 студентов на три подгруппы по 6, 9 и 10 человек, соответственно? В данном примере $n = 25$, $k = 3$, $n_1 = 6$, $n_2 = 9$, $n_3 = 10$. Согласно формуле, число таких разбиений равно $N_{25}(6, 9, 10) = \dfrac{25!}{6!\cdot 9!\cdot 10!}$.

3.5 Рекуррентные соотношения

В предыдущих пунктах были рассмотрены основные формулы комбинаторики. Были затронуты два логических правила: правило суммы и правило произведения. Вывод формул комбинаторики осуществим с применением метода рекуррентных соотношений. Метод рекуррентных соотношений состоит в том, что решение комбинаторной задачи с n предметами выражается через решение аналогичной задачи с меньшим количеством предметов с помощью некоторого соотношения, которое называется рекуррентным. Рассмотрим метод рекуррентных соотношений на примере сочетаний с повторениями.

Обозначим количество сочетаний из n предметов $\{a_1, \ldots, a_n\}$ по k с повторениями через f_n^k. Любое из сочетаний или содержит, или не содержит a_1. Количество сочетаний, которые не содержат a_1, составляет f_{n-1}^k (это сочетания из предметов $a_2, \ldots, a_n$ по k). Каждое сочетание, которое содержит a_1 как минимум один раз, может быть получено присоединением к a_1 некоторого сочетания из n предметов по $(k-1)$ (количество таких сочетаний составляет f_n^{k-1}. Таким образом,

$$f_n^k = f_{n-1}^k + f_n^{k-1}. \qquad (3.9)$$

Последовательно применяя это рекуррентное соотношение, получим:

$$f_n^k = f_n^{k-1} + f_{n-1}^k = f_n^{k-1} + (f_{n-1}^{k-1} + f_{n-2}^k) = \ldots = f_n^{k-1} + f_{n-1}^{k-1} + \ldots + f_2^{k-1} + f_1^k \qquad (3.10)$$

Понятно, что $f_n^1 = n$, $f_1^k = 1$. Учитывая, что k = 2, имеем:

$$f_n^2 = n + (n-1) + \ldots + 2 + 1 = \frac{n\cdot(n+1)}{2} = C_{n+1}^2. \qquad (3.11)$$

Если k = 3, тогда имеем:

$$f_n^3 = C_{n+1}^2 + C_n^2 + \ldots + C_3^2 + C_2^2 = C_{n+2}^3 . \quad (3.12)$$

На (k-1)-ом шаге получим:

$$f_n^k = C_{n+k-1}^k = P(k, n-1), \quad (3.13)$$

что согласуется с предыдущей формулой (3.7).

Контрольные вопросы к разделу 3

1. Сколько трехзначных чисел можно составить из цифр 1; 3; 5; 7, используя в записи числа каждую из них не более одного раза?

2. Сколько пятизначных чисел, делящихся на три, можно составить из цифр 3; 4; 6; 7; 9, если каждое число не содержит одинаковых цифр?

3. У рояля 88 клавиш. Сколькими способами можно извлечь последовательно 4 звука?

4. На тарелке лежат 10 яблок и 6 апельсинов. Сколькими способами можно выбрать один плод?

5. Сколькими различными способами можно распределить между шестью лицами две различные путевки в санаторий?

6. Пять разных предметов раздают 8 людям, причем может случиться так, что некоторые получат по несколько предметов. Сколькими способами может быть произведено разделение?

7. В семестре изучается 7 дисциплин. В понедельник 3 пары, причем все разные. Сколькими способами можно составить расписание на понедельник?

8. Сколько различных кортежей получится, если переставлять буквы слова «математика»?

9. Сколькими способами можно разместить 10 разных шаров в трех разных урнах?

4 БУЛЕВЫ ФУНКЦИИ

4.1 Основные понятия и определения

Булевой функцией $f(x_1, x_2, \dots, x_n)$ называется произвольная функция n переменных, аргументы которой $x_1, x_2, \dots, x_n$ и сама функция f принимают значения 0 или 1, т. е. $x_i \in \{0, 1\}$, $i = 1, 2, \dots, n$; $f(x_1, x_2, \dots, x_n) \in \{0, 1\}$.

Одной из важнейших интерпретаций теории булевых функций является теория *переключательных функций*. Первоначально математический аппарат теории булевых функций был применен для анализа и синтеза релейно-контактных схем с операциями последовательного и параллельного соединения контактов.

Любая булева функция может быть представлена таблицей, в левой части которой перечислены все наборы переменных (их 2^n), а в правой части – значения функции. Пример такого задания представлен в табл. 4.1.

Таблица 4.1 – Табличный способ задания булевой функции

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Для формирования столбца значений переменных удобен *лексикографический порядок*, в соответствии с которым каждый последующий набор значений получается из предыдущего прибавлением 1 в двоичной системе счисления, например, 100 = 011 + 1. Всего существует 2^{2^n} различных булевых функций n переменных. Функций одной переменной – 4. Из них выделим функцию «отрицание x» (обозначается $\overline{x}$). Эта функция представлена в табл. 4.2.

Таблица 4.2 – Булева функция одной переменной

x	$\overline{x}$
0	1
1	0

Булевых функций двух переменных – 16 (2^{2^n} при $n = 2$). Те из них, которые имеют специальные названия, представлены в табл. 4.3.

Таблица 4.3 – Булева функция двух переменных

x_1	x_2	$x_1 \vee x_2$	$x_1 \wedge x_2$	$x1 \to x2$	$x_1 \sim x_2$	$x_1 \oplus x_2$	$x_1 \downarrow x_2$	$x_1 \mid x_2$
0	0	0	0	1	1	0	1	1
0	1	1	0	1	0	1	0	1
1	0	1	0	0	0	1	0	1
1	1	1	1	1	1	0	0	0

В табл. 4.3 представлены следующие функции двух переменных:

$x_1 \vee x_2$ – *дизъюнкция*;

$x_1 \wedge x_2$ – *конъюнкция*;

$x_1 \to x_2$ – *импликация*;

$x_1 \sim x_2$ – *эквивалентность*;

$x_1 \oplus x_2$ – *сложение по модулю 2;*

$x_1 \downarrow x_2$ – *стрелка Пирса*;

$x_1 \mid x_2$ – *штрих Шеффера*.

Остальные функции специальных названий не имеют и могут быть выражены через перечисленные выше функции.

Формула логики булевых функций определяется индуктивно следующим образом:

1. Любая переменная, а также константы 0 и 1 есть формула.

2. Если A и B – формулы, то $\overline{A}$, $A \vee B$, $A \wedge B$, $A \to B$, $A \sim B$ есть формулы.

3. Ничто, кроме указанного в пунктах 1–2, не есть формула.

Пример 4.1. Выражение $((\overline{x} \vee y) \wedge ((y \to z) \sim x)$ является формулой.

Выражение $\overline{x} \wedge y \to z \sim \overline{x}$ не является формулой.

Часть формулы, которая сама является формулой, называется ***подформулой.***

Пример 4.2. $x \wedge (y \to z)$ – формула; $y \to z$ – ее подформула.

Функция f есть *суперпозиция* функций $f_1, f_2, \ldots, f_n$, если f получается с помощью подстановок этих формул друг в друга и переименованием переменных.

Пример 4.3. $f_1 = x_1 \wedge x_2$ (конъюнкция); $f_2 = \overline{x}$ (отрицание).

Возможны две суперпозиции:

1. $f = f_1(f_2) = (\overline{x_1}) \wedge (\overline{x_2})$ – конъюнкция отрицаний;

2. $f = f_2(f_1) = \overline{(x_1 \wedge x_2)}$ – отрицание конъюнкции.

Порядок подстановки задается формулой.

Всякая формула задает способ вычисления функции, если известны значения переменных.

Пример 4.4. Построим таблицу значений функции $f(x_1, x_2, x_3) = \overline{(x_2 \to \overline{x_3})} \sim (\overline{x_1} \vee x_2)$.

Таблица 4.4 – Вычисление функции $f(x_1, x_2, x_3) = \overline{(x_2 \to \bar{x}_3)} \sim (\bar{x}_1 \vee x_2)$

x_1	x_2	x_3	$\bar{x}_3$	$x_2 \to \bar{x}_3$	$\overline{(x_2 \to \bar{x}_3)}$	$\bar{x}_1$	$\bar{x}_1 \vee x_2$	$f(x_1, x_2, x_3)$
0	0	0	1	1	0	1	1	0
0	0	1	0	1	0	1	1	0
0	1	0	1	1	0	1	1	0
0	1	1	0	0	1	1	1	1
1	0	0	1	1	0	0	0	1
1	0	1	0	1	0	0	0	1
1	1	0	1	1	0	0	1	0
1	1	1	0	0	1	0	1	1

Таким образом, формула каждому набору аргументов ставит в соответствие значение функции. Следовательно, формула так же, как и таблица, может служить способом задания функции. В дальнейшем формулу будем отождествлять с функцией, которую она реализует. Последовательность вычислений функции задается скобками. Упрощение записи формул осуществляется в соответствии со следующими этапами:

1. Внешние скобки можно отпускать;

2. Приоритет применения связок возрастает в следующем порядке: $\sim$, $\to$, $\vee$, $\wedge$, $\bar{}$;

3. Связка $\bar{}$ – над одной переменной сильнее всех связок;

4. Если связка – стоит над формулой, то сначала выполняется формула, затем отрицание;

5. Если нет скобок, то операции $\sim$ и $\to$ выполняются в последнюю очередь.

4.2 Равносильные преобразования формул. Булева алгебра (алгебра логики). Полные системы булевых функций

В отличие от табличного задания, представление функции формулой не единственно. Например, две различные формулы $(\bar{x}_1) \vee (\bar{x}_2)$ и $\overline{(x_1 \wedge x_2)}$ реализуют одну функцию – штрих Шеффера.

Две формулы, реализующие одну и ту же функцию, называются ***равносильными***. Равносильность формул *A* и *B* будем обозначать следующим образом: $A \equiv B$.

Для того чтобы установить равносильность формул, можно составить таблицы значений функции для каждой формулы и сравнить их. Для равносильных формул эти таблицы совпадают. Другой способ установления равносильности формул заключается в использовании некоторых установленных равносильностей булевых формул.

Основные равносильности булевых формул

Для любых формул *A, B, C* справедливы следующие равносильности:

1. *Коммутативность*:

а) $A \wedge B \equiv B \wedge A$ (для конъюнкции);

б) $A \vee B \equiv B \vee A$ (для дизъюнкции).

2. *Ассоциативность*:

а) $A \wedge (B \wedge C) \equiv (A \wedge B) \wedge C$ (для конъюнкции);

б) $A \vee (B \vee C) \equiv (A \vee B) \vee C$ (для дизъюнкции).

3. *Дистрибутивность*:

а) $A \wedge (B \vee C) \equiv A \wedge B \vee A \wedge C$ (для конъюнкции относительно дизъюнкции);

б) $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$ (для дизъюнкции относительно конъюнкции).

4. *Закон де Моргана*:

а) $\overline{(A \wedge B)} \equiv \overline{A} \vee \overline{B}$ (отрицание конъюнкции есть дизъюнкция отрицаний);

б) $\overline{(A \vee B)} \equiv \overline{A} \wedge \overline{B}$ (отрицание дизъюнкции есть конъюнкция отрицаний).

5. *Идемпотентность:*

а) $A \wedge A \equiv A$ (для конъюнкции);

б) $A \vee A \equiv A$ (для дизъюнкции).

6. *Поглощение:*

а) $A \wedge (A \vee B) \equiv A$ (1-й закон поглощения);

б) $A \vee (A \wedge B) \equiv A$ (2-й закон поглощения).

7. *Расщепление (склеивание):*

а) $A \wedge B \vee A \wedge (\overline{B}) \equiv A$ (1-й закон расщепления);

б) $(A \vee B) \wedge (A \vee \overline{B}) \equiv A$ (2-й закон расщепления).

8. *Двойное отрицание*:

$\overline{\overline{A}} \equiv A$.

9. *Свойства констант:*

а) $A \wedge 1 \equiv A$;

б) $A \wedge 0 \equiv 0$;

в) $A \vee 1 \equiv 1$;

г) $A \vee 0 \equiv A$;

д) $\overline{0} \equiv 1$;

е) $\overline{1} \equiv 0$.

10. *Закон противоречия*:

$A \wedge \overline{A} \equiv 0$.

11. *Закон «исключенного третьего»:*

$A \vee \overline{A} \equiv 1$.

Каждая из перечисленных равносильностей может быть доказана с помощью таблиц значений функций, составленных для выражений, стоящих слева и справа от символа « ≡ ». Следующие важные равносильности показывают, что все логические операции могут быть выражены через операции конъюнкции, дизъюнкции и отрицания:

12. $A \rightarrow B \equiv \overline{A} \vee B \equiv \overline{(A \wedge \overline{B})}$;

13. $A \sim B \equiv (A \to B) \wedge (B \to A) \equiv (A \wedge B) \vee (\overline{A} \wedge \overline{B}) \equiv (A \vee \overline{B}) \wedge (\overline{A} \vee B)$;

14. $A \oplus B \equiv (A \vee \overline{B}) \vee (\overline{A} \wedge B)$;

15. $A \downarrow B \equiv \overline{A \vee B} \equiv (\overline{A} \wedge \overline{B})$;

16. $A \mid B \equiv \overline{A \wedge B} \equiv (\overline{A} \vee \overline{B})$.

Используя равносильности 3а и 3б и метод математической индукции, нетрудно получить также следующие равносильности (обобщенные законы дистрибутивности):

17. $(A_1 \vee A_2 \vee \ldots \vee A_n) \wedge (B_1 \vee B_2 \vee \ldots \vee B_m) \equiv A_1 \wedge B_1 \vee A_1 \wedge B_2 \vee \ldots \vee A_1 \wedge B_m \vee \ldots \vee A_n \wedge B_1 \vee A_n \wedge B_2 \vee \ldots \vee A_n \wedge B_m$;

18. $(A_1 \wedge A_2 \wedge \ldots \wedge A_n) \vee (B_1 \wedge B_2 \wedge \ldots \wedge B_m) \equiv (A_1 \vee B_1) \wedge (A_1 \vee B_2) \wedge \ldots \wedge (A_1 \vee B_m) \wedge \ldots \wedge (A_n \vee B_1) \wedge (A_n \vee B_2) \wedge \ldots \wedge (A_n \vee B_m)$.

Используя равносильности 4а и 4б и метод математической индукции, можно получить также следующие равносильности (обобщенные законы де Моргана):

19. $\overline{(A_1 \wedge A_2 \wedge \ldots \wedge A_n)} \equiv \overline{A_1} \vee \overline{A_2} \vee \ldots \vee \overline{A_n}$;

20. $\overline{(A_1 \vee A_2 \vee \ldots \vee A_n)} \equiv \overline{A_1} \wedge \overline{A_2} \wedge \ldots \wedge \overline{A_n}$.

В равносильностях 1…20 в качестве A, B, A_i, B_i могут быть подставлены любые формулы и, в частности, переменные. Приведем правило, с помощью которого можно переходить от одних равносильностей к другим.

Правило равносильных преобразований

Пусть для формул A и B справедливо утверждение $A \equiv B$. Пусть C_A – формула, содержащая A в качестве своей подформулы. Пусть C_B получается из C_A заменой A на B. Тогда $C_A \equiv C_B$.

Пример 4.5. Пусть $A = x \to y$, $B = \overline{x} \vee y$.

Равносильность 12 позволяет утверждать, что $A \equiv B$.

Символы $\wedge$, $\vee$ называются ***двойственными***.

Формула A^* называется *двойственной* формуле A, если она получена из A одновременной заменой всех символов $\wedge$, $\vee$ на двойственные. Отсюда, если $A \equiv B$, то $A^* \equiv B^*$.

Принцип двойственности можно использовать для нахождения новых равносильностей. Например, для 1-го закона поглощения (равносильность 6а) имеем: $A \wedge (A \vee B) \equiv A$. Следуя принципу двойственности, получим новую равносильность: $A \vee A \wedge B \equiv A$ (2-й закон поглощения).

Как известно, *алгеброй* называют систему, включающую в себя некоторое непустое множество объектов с заданными на нем функциями (операциями), результатами применения которых к объектам данного множества являются объекты того же множества.

Булевой алгеброй или *алгеброй логики* называется двухэлементное множество $B = \{0, 1\}$ вместе с операциями конъюнкции, дизъюнкции и отрицания.

Система булевых функций $\{f_1, f_2, \ldots, f_n\}$ называется полной, если любая булева функция может быть выражена в виде суперпозиции этих функций. Из равносильностей 12…16 следует, что все логические операции могут быть выражены через операции конъюнкции, дизъюнкции и от-

рицания. Поэтому система функций {¯, ∧, ∨} является полной. Также полными являются следующие системы функций: {¯, ∨}; {¯, ∧}; {¯, →}.

Полнота систем {¯, ∨} и {¯, ∧} следует из полноты системы {¯, ∧, ∨}, а также законов де Моргана и двойного отрицания, следствием которых является возможность выразить конъюнкцию через дизъюнкцию и наоборот: $A \wedge B \equiv (\overline{\overline{A} \vee \overline{B}})$; $A \vee B \equiv (\overline{\overline{A} \wedge \overline{B}})$.

Поэтому система {¯, ∧, ∨} может быть сокращена на одну функцию:

Полнота системы {¯, →} следует из полноты системы {¯, ∨} и равносильности 12, позволяющей выразить импликацию через отрицание и дизъюнкцию: $A \to B \equiv \overline{A} \vee B$.

4.3 Нормальные формы изображения булевых функций

Элементарной конъюнкцией называется конъюнкция (возможно одночленная), составленная из переменных или отрицаний переменных.

Пример 4.6. x, y, $x \wedge y$, $\bar{x}_1 \wedge x_2 \wedge (\bar{x}_3)$ – элементарные конъюнкции, а $x \vee y$, $x_1 \wedge \bar{x}_2 \vee \bar{x}_1 \wedge x_2$ – не элементарные конъюнкции.

Дизъюнктивной нормальной формой (ДНФ) называется формула, имеющая вид дизъюнкции элементарных конъюнкций (в вырожденном случае это может быть одна конъюнкция).

Пример 4.7. Данные формулы находятся в ДНФ $x \wedge y$; $\bar{x}_1 \wedge x_2 \wedge \bar{x}_3$; $a \wedge b \vee a \wedge \bar{b} \wedge c \vee \bar{a} \wedge c$; $x \vee x \wedge y$.

Совершенной дизъюнктивной нормальной формой (СДНФ) называется такая дизъюнктивная нормальная форма, каждый конъюнктивный член которой содержит *все* переменные либо их отрицания.

Пример 4.8. $x \wedge y$, $x \wedge \bar{y} \vee \bar{x} \wedge y$ – СДНФ функции двух переменных.

Элементарной дизъюнкцией называется дизъюнкция (возможно одночленная), составленная из переменных или отрицаний переменных.

Пример 4.9. x, y, $x \vee y$, $\bar{x}_1 \vee x_2 \vee \bar{x}_3$ – элементарные дизъюнкции.

Конъюнктивной нормальной формой (КНФ) называется формула, имеющая вид конъюнкции элементарных дизъюнкций (в вырожденном случае это может быть одна дизъюнкция).

Пример 4.10. Следующие формулы находятся в КНФ: $x \vee \bar{y}$; $(x \vee y) \wedge (x \vee \bar{y} \vee z) \wedge (\bar{x} \vee y \vee \bar{z})$.

Совершенной конъюнктивной нормальной формой (СКНФ) называется такая конъюнктивная нормальная форма, каждый дизъюнктивный член которой содержит *все* переменные, либо их отрицания.

Пример 4.11. $x \vee y$, $(x \vee \bar{y}) \wedge (\bar{x} \vee y)$ – СКНФ функции двух переменных.

Для каждой формулы булевой функции A имеется равносильная ей дизъюнктивная нормальная форма (ДНФ) и конъюнктивная нормальная форма (КНФ).

Для каждой формулы A имеется бесконечное множество ДНФ и КНФ, но для решения задач, в которых эти формы нужны, требуется, как правило, найти хотя бы одну из них.

Алгоритм приведения формул булевых функций к ДНФ (КНФ)

Шаг 1. Все подформулы A вида $B \to C$ (т. е. содержащие импликацию) заменяем на $\overline{B} \vee C$ или на $\overline{(B \wedge \overline{C})}$ (в соответствии с равносильностью 12).

Шаг 2. Все подформулы A вида $B \sim C$ (т. е. содержащие эквивалентность) заменяем на $(A \wedge B) \vee (\overline{A} \wedge \overline{B})$ или $(A \vee \overline{B}) \wedge (\overline{A} \vee B)$ (в соответствии с равносильностью 13).

Шаг 3. Все отрицания, стоящие над сложными подформулами, опускаем по законам де Моргана (в соответствии с равносильностями 4, 19, 20).

Шаг 4. Устраняем все двойные отрицания над формулами (в соответствии с равносильностью 8).

Шаг 5. Осуществляем раскрытие всех скобок по закону дистрибутивности конъюнкции относительно дизъюнкции для ДНФ (в соответствии с равносильностями 3а и 17) или по закону дистрибутивности дизъюнкции относительно конъюнкции для КНФ (в соответствии с равносильностями 3б и 18).

Шаг 6. Для получения более простой формулы целесообразно использовать равносильности 5, 6, 7, 9, 10, 11.

Очевидно, что в результате всех указанных операций формула имеет вид ДНФ или КНФ. Указанные операции, вообще говоря, могут осуществляться в любом порядке, однако целесообразно придерживаться изложенного выше, за исключением снятия двойных отрицаний (шаг 4), от которых следует избавляться по мере их появления.

Пример 4.12. Привести к ДНФ формулу $(x \vee y) \wedge (x \vee \overline{y})$.

Решение: $(x \vee y) \wedge (x \vee \overline{y}) \equiv x$.

Привести к КНФ формулу $(x \to y) \wedge x \wedge y$.

Решение: $(x \to y) \wedge x \wedge y \equiv (\overline{x} \vee y) \wedge x \wedge y$.

Приведение некоторой формулы к ДНФ и КНФ не является однозначным. Количество равносильных ДНФ и КНФ, в принципе, бесконечно. Однако, совершенные дизъюнктивные и конъюнктивные нормальные формы (СДНФ и СКНФ) или не существуют, или единственны.

Каждая формула A, не равная тождественно нулю, может быть приведена к СДНФ, которая является единственной с точностью до перестановки дизъюнктивных членов.

Каждая формула A, не равная тождественно единице, может быть приведена к СКНФ, которая является единственной с точностью до перестановки конъюнктивных членов.

Доказательство этих теорем состоит в указании алгоритма приведения формулы A к СДНФ и СКНФ.

Алгоритм приведения формулы булевой функции к СДНФ

Шаг 1. Используя алгоритм построения ДНФ, находим формулу В, являющуюся ДНФ формулы А.

Шаг 2. Вычеркиваем в *B* все элементарные конъюнкции, в которые одновременно входят какая-нибудь переменная и ее отрицание. Это обосновывается равносильностями: $A \wedge \bar{A} \equiv 0$, $B \wedge 0 \equiv 0$, $C \vee 0 \equiv C$.

Шаг 3. Если в элементарной конъюнкции формулы *B* некоторая переменная или ее отрицание встречается несколько раз, то оставляем только одно ее вхождение. Это обосновывается законом идемпотентности для конъюнкции: $A \wedge A \equiv A$.

Шаг 4. Если в элементарную конъюнкцию С формулы В не входит ни переменная *x*, ни ее отрицание $\bar{x}$, то на основании 1-го закона расщепления (равносильность 7а) заменяем С на $(C \wedge x) \vee (C \wedge \bar{x})$.

Шаг 5. В каждой элементарной конъюнкции формулы *B* переставляем конъюнктивные члены так, чтобы для каждого i ($i = 1, ..., n$) на i-ом месте была либо переменная x_i, либо ее отрицание $\bar{x}_i$.

Шаг 6. Устраняем возможные повторения конъюнктивных членов согласно закону идемпотентности для дизъюнкции: $C \vee C \equiv C$.

Пример 4.13. Привести к СДНФ формулу $(x \vee y) \wedge (x \vee \bar{y})$.

Решение: $(x \vee y) \wedge (x \vee \bar{y}) \equiv x \equiv x \wedge (y \vee \bar{y}) \equiv x \wedge y \vee x \wedge \bar{y}$.

Привести к СКНФ формулу $(x \to y) \wedge x \wedge y$.

Решение:

$$(x \to y) \wedge x \wedge y \equiv (\bar{x} \vee y) \wedge x \wedge y \equiv (\bar{x} \vee y) \wedge (x \vee y \wedge \bar{y}) \wedge (y \vee x \wedge \bar{x}) \equiv$$
$$\equiv (\bar{x} \vee y) \wedge (x \vee y) \wedge (x \vee \bar{y}) \wedge (y \vee x) \wedge (y \vee \bar{x}) \equiv\equiv (x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{x} \vee y).$$

4.4 Разложение булевой функции по переменным

Пусть *s* принимает значения 0 или 1, т. е. $s \in \{0, 1\}$. Введем обозначение: $x^s = \bar{x}$, если $s = 0$, $x^s = x$, если $s = 1$, т. е. $x^0 = \bar{x}$, $x^1 = x$. Очевидно, что $x^s = 1$, если $x = s$ и $x^s = 0$, если $x \neq s$.

Каждая булева функция $f(x_1, x_2, ..., x_n)$ может быть представлена в виде:

$$f(x_1, x_2, ..., x_n) = f(x_1, x_2, ..., x_m, x_{m+1}, ..., x_n) =$$
$$= \bigvee (x_1^{s1} \wedge x_2^{s2} \wedge \ldots \wedge x_m^{sm} \wedge f(s_1, s_2, \ldots, s_m, x_{m+1}, \ldots, x_n)),\ m \le n, \qquad (4.1)$$

где дизъюнкция берется по всем наборам $(s_1, s_2, ..., s_m)$ (их 2^m).

Всякая булева функция $f(x_1, x_2, \ldots, x_n)$, не равная тождественно 0, может быть представлена формулой в СДНФ, которая определяется однозначно с точностью до перестановки дизъюнктивных членов.

Алгоритм представления булевой функции, заданной таблицей, формулой в СДНФ

Шаг 1. Выбираем в таблице все наборы переменных $s_1, s_2, \ldots, s_n$, для которых значение f равно 1, т. е. $f(s_1, s_2, \ldots, s_n) = 1$.

Шаг 2. Для каждого такого набора (строки таблицы) составляем конъюнкцию $x_1^{s1} \wedge x_2^{s2} \wedge \ldots \wedge x_n^{sn}$, где $x_i^{si} = x_i$, если $s_i = 1$ и $x_i^{si} = \bar{x}_i$, если $s_i = 0$, $i = 1, 2, \ldots, n$.

Шаг 3. Составляем дизъюнкцию всех полученных конъюнкций. В результате получится формула данной функции в СДНФ.

Пример 4.14. Найдем формулу в СДНФ для функции $f(x_1, x_2, x_3)$, заданной таблицей 4.4.

1. Выберем в таблице строки, где $f(x_1, x_2, x_3) = 1$. Это 4-я, 5-я. 6-я и 8-я строки.

2. Для каждой выбранной строки составляем конъюнкции по правилу, указанному в шаге 2. Получим, соответственно, для четырех выбранных строк:

$x_1^0 \wedge x_2^1 \wedge x_3^1 = \bar{x}_1 \wedge x_2 \wedge x_3$.

$x_1^1 \wedge x_2^0 \wedge x_3^0 = x_1 \wedge \bar{x}_2 \wedge \bar{x}_3$.

$x_1^1 \wedge x_2^0 \wedge x_3^1 = x_1 \wedge \bar{x}_2 \wedge x_3$.

$x_1^1 \wedge x_2^1 \wedge x_3^1 = x_1 \wedge x_2 \wedge x_3$.

3. Составляем дизъюнкцию всех полученных конъюнкций и находим СДНФ:

$f(x_1, x_2, x_3) = \bar{x}_1 \wedge x_2 \wedge x_3 \vee x_1 \wedge \bar{x}_2 \wedge \bar{x}_3 \vee x_1 \wedge \bar{x}_2 \wedge x_3 \vee x_1 \wedge x_2 \wedge x_3$.

Всякая булева функция $f(x_1, x_2, \ldots, x_n)$, не равная тождественно 1, может быть представлена формулой в СКНФ, которая определяется однозначно с точностью до перестановки дизъюнктивных членов.

Для получения формулы булевой функции $f(x_1, x_2, \ldots, x_n)$ в СКНФ следует воспользоваться следующим алгоритмом.

Алгоритм представления булевой функции, заданной таблицей, формулой в СКНФ

Шаг 1. Выбираем в таблице все наборы переменных $s_1, s_2, \ldots, s_n$, для которых значение $f(s_1, s_2, \ldots, s_n) = 0$.

Шаг 2. Для каждого такого набора (строки таблицы) составляем дизъюнкцию $x_1^{\bar{s}1} \vee x_2^{\bar{s}2} \vee \ldots \vee x_n^{\bar{sn}} = x_i^{\,n}$, где $x_i^{\bar{si}} = x_i$, если $s_i = 0$ и $x_i^{\bar{si}} = \bar{x}_i$, если $s_i = 1$, $i = 1, 2, \ldots, n$.

Шаг 3. Составляем конъюнкцию всех полученных дизъюнкций. В результате получится СКНФ.

Пример 4.15. Найдем формулу в СКНФ для функции $f(x_1, x_2, x_3)$, заданной таблицей 4.4.

1. Выберем в таблице строки, где $f(x_1, x_2, x_3) = 0$. Это 1-я, 2-я и 3-я и 7-я строки.

2. Для каждой выбранной строки составляем дизъюнкции по правилу, указанному в шаге 2. Получим, соответственно, для трех выбранных строк:

$x_1^1 \vee x_2^1 \vee x_3^1 = x_1 \vee x_2 \vee x_3$.

$x_1^1 \vee x_2^1 \vee x_3^0 = x_1 \vee x_2 \vee \bar{x}_3$.

$x_1^1 \vee x_2^0 \vee x_3^1 = x_1 \vee \bar{x}_2 \vee x_3$.

$x_1^0 \vee x_2^0 \vee x_3^1 = \bar{x}_1 \vee \bar{x}_2 \vee x_3$.

3. Составляем конъюнкцию всех полученных дизъюнкций и находим СКНФ:

$f(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \bar{x}_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3)$.

Так, для функции, рассмотренной в примерах 4.14 и 4.15, $n = 3$, $p = 4$, $q = 4$, $p + q = 8 = 2^3$.

4.5 Минимизация формул булевых функций в классе дизъюнктивных нормальных форм

Произвольная булева функция может быть представлена формулой в ДНФ и КНФ, причем такое представление неоднозначно. Равносильными преобразованиями можно получить формулу, содержащую меньшее число вхождений переменных. Например, две различные формулы: $f_1(x_1, x_2) = x_1 \vee x_1 \wedge x_2 \vee \bar{x}_2$ и $f_2(x_1, x_2) = x_1 \vee \bar{x}_2$ равносильны, так как в силу 2-го закона поглощения $x_1 \vee x_1 \wedge x_2 \equiv x_1$.

Но в формуле $f_1(x_1, x_2)$ содержится четыре вхождений переменных, а в формуле $f_2(x_1, x_2)$ – два.

ДНФ называется ***минимальной***, если она содержит наименьшее общее число вхождений переменных среди всех равносильных ей ДНФ.

Импликантом булевой функции $f(x_1, x_2, \ldots, x_n)$ называется элементарная конъюнкция C, не равная тождественно 0, такая, что $C \vee f \equiv f$. Отметим, что любая конъюнкция любой ДНФ в силу закона идемпотентности является импликантом этой функции.

Импликант C функции f называется ***простым импликантом***, если после отбрасывания любой переменной из C получается элементарная конъюнкция, не являющаяся импликантом функции f.

Пример 4.16. Для функции $x \wedge y \vee x \wedge z \vee x \wedge y \wedge z \equiv x \wedge (y \vee z)$ конъюнкции $x \wedge y$ и $x \wedge z$ – простые импликанты, а $x \wedge y \wedge z$ – импликант, но не простой.

Дизъюнкция всех простых импликантов булевой функции f называется ***сокращенной ДНФ функции f.***

Для нахождения сокращенной ДНФ используется следующий алгоритм, в основе которого лежит метод Квайна.

Алгоритм Квайна построения сокращенной ДНФ

Шаг 1. Находим для данной булевой функции f ее формулу F, находящуюся в СДНФ.

Шаг 2. Находим все простые импликанты функции f. Для этого все элементарные конъюнкции формулы F попарно сравниваем между собой. Если две элементарные конъюнкции таковы, что они имеют вид $C \wedge x_i$ и $C \wedge \bar{x}_i$, то выписываем конъюнкцию C. Это является следствием 1-го закона расщепления (склеивания). Конъюнкция C содержит $n - 1$ вхождение переменных. Элементарные конъюнкции, для которых произошло склеивание, помечаем. После построения всех конъюнкций, включающих $n - 1$ вхождение переменных, вновь сравниваем их попарно, производим, если возможно, склеивание, выписываем полученные конъюнкции из $n - 2$ членов, помечаем склеивающиеся конъюнкции из $n - 1$ членов и т. д. Процесс заканчивается, когда дальнейшее склеивание невозможно. Все непомеченные элементарные конъюнкции являются простыми импликантами. Их дизъюнкция даст нам формулу F_1, равносильную F, находящуюся в ДНФ и состоящую из простых импликантов, т. е. сокращенную ДНФ.

Пример 4.17. Найдем сокращенную ДНФ функции $f(x_1, x_2, x_3) = \overline{(x_2 \to \bar{x}_3)} \sim (\bar{x}_1 \vee x_2)$.

1. Шаг 1 был выполнен ранее (пример 4.14).

СДНФ формулы $f(x_1, x_2, x_3)$ является формула $F(x_1, x_2, x_3) = \bar{x}_1 \wedge x_2 \wedge \wedge x_3 \vee x_1 \wedge \bar{x}_2 \wedge \bar{x}_3 \vee x_1 \wedge \bar{x}_2 \wedge x_3 \vee x_1 \wedge x_2 \vee x_3$.

2. Попарно сравниваем все 4 трехчленные конъюнкции (всех сравнений $C_4^2 = 6$) и применяем, где это возможно, закон склеивания:

$\bar{x}_1 \wedge x_2 \wedge x_3 \vee x_1 \wedge x_2 \wedge x_3 = x_2 \wedge x_3$;

$x_1 \wedge \bar{x}_3 \vee x_1 \wedge \bar{x}_2 \wedge x_3 = x_1 \wedge \bar{x}_2$;

$x_1 \wedge \bar{x}_2 \wedge x_3 \vee x_1 \wedge x_2 \wedge x_3 = x_1 \wedge x_3$.

На первом этапе получили 3 двучленные конъюнкции:

$x_2 \wedge x_3$, $x_1 \wedge \bar{x}_2$, $x_1 \wedge x_3$.

Все трехчленные конъюнкции оказались помеченными.

Попарно сравниваем все 3 двучленные конъюнкции (всех сравнений 3) и замечаем, что склеивание невозможно.

В результате получим сокращенную ДНФ формулы f:

$F_1(x_1, x_2, x_3) = x_2 \wedge x_3 \vee x_1 \wedge \bar{x}_2 \vee x_1 \wedge x_3$.

На практике для построения сокращенной ДНФ удобнее пользоваться модифицированным методом Квайна – Мак-Класки. Суть метода состоит в автоматизации процесса склеивания.

Алгоритм Квайна – Мак-Класки построения сокращенной ДНФ

Шаг 1. Составим таблицу значений булевой функции. Для примера воспользуемся данными таблицы 4.4.

Очевидно, в силу алгоритма 4. 3 (см. также пример 4.15), эта функция имеет следующую формулу в СДНФ:

$f(x_1, x_2, x_3) = \bar{x}_1 \wedge x_2 \wedge x_3 \vee x_1 \wedge \bar{x}_2 \wedge \bar{x}_3 \vee x_1 \wedge \bar{x}_2 \wedge x_3 \vee x_1 \wedge x_2 \wedge x_3$.

Шаг 2. Выпишем наборы переменных, на которых функция принимает значение 1, причем эти наборы упорядочим по группам так, что в каждую группу входят наборы с одинаковым числом единиц. Пусть A_i – группа наборов переменных, таких, что каждый набор содержит *i* единиц, и функция на этом наборе переменных принимает значение, равное единице.

Группы A_0 (где все переменные нули, а значение функции равно 1) нет.

Группа A_1 (где одна переменная единица, остальные нули, и значение функции равно 1):

1 0 0.

Группа A_2:

0 1 1;

1 0 1.

Группа A_3:

1 1 1.

Шаг 3. Производим попарное сравнение наборов переменных, входящих в соседние группы. Если при этом сравнении обнаружатся два набора, которые отличаются только в одном разряде, то вместо них записывается один набор, в котором вместо несовпадающих разрядов ставится « – » (прочерк). После всех возможных сравнений из предшествующего списка вычеркиваются все наборы, которые участвовали в образовании хотя бы одного набора с прочерком. Формируются два массива наборов: наборы с прочерками (массив *P*) и не вычеркнутые (массив R).

Эти действия соответствуют склеиванию конъюнкций и уменьшению числа вхождений переменных.

Для нашего примера при сравнении групп A_1 и A_2:

вместо (1 0 0) и (1 0 1) получим (1 0 –).

При сравнении групп A_2 и A_3:

вместо (0 1 1) и (1 1 1) получим (– 1 1);

вместо (1 0 1) и (1 1 1) получим (1 – 1).

После этого этапа массив *R* пуст, т. к. все наборы участвовали в образовании наборов с прочерками, а массив $P = P(1)$ включает следующие наборы: (1 0 –); (– 1 1); (1 – 1).

Далее рассмотрим наборы с прочерками. Они вновь попарно сравниваются между собой. При этом имеет смысл сравнивать лишь наборы, в которых прочерк стоит в совпадающих разрядах. Если сравниваемые наборы отличаются друг от друга только в одном разряде, то выписываем набор с двумя прочерками (старым и новым). После всех попарных сравнений из множества наборов с одним прочерком вычеркиваются все наборы, имеющие один прочерк и участвовавшие в образовании набора с двумя прочерками. Наборы с одним прочерком, не участвовавшие в образовании наборов с двумя прочерками, помещаются в массив *R*.

Для примера попарное сравнение наборов с одним прочерком не приводит к образованию наборов с двумя прочерками.

Далее рассмотрим наборы с двумя прочерками и т. д. Процесс прекращается, если на очередном шаге все рассматриваемые наборы попадают

в R. Нетрудно убедиться, что каждому набору из R соответствует простой импликант, причем единице соответствует переменная, взятая без отрицания, нулю – переменная, взятая с отрицанием, а прочерку – отсутствие соответствующей переменной. Сокращенная ДНФ есть дизъюнкция этих простых импликантов.

Для примера, процедура сравнения заканчивается после формирования наборов с одним прочерком. Массив R после этого включает наборы: (1 0 –); (– 1 1); (1 – 1).

Сокращенная ДНФ имеет вид: $f(x_1, x_2, x_3) = x_2 \wedge x_3 \vee x_1 \wedge \bar{x}_2 \vee x_1 \wedge x_3$.

Далее процесс нахождения минимальной ДНФ сводится к отбрасыванию некоторых простых импликантов.

Простой импликант называется ***существенным (ядровым) импликантом,*** если его удаление из сокращенной ДНФ функции приводит к ДНФ, которая не равносильна исходной ДНФ.

Построение минимальной ДНФ сводится к отбрасыванию несущественных импликантов из сокращенной ДНФ.

Элементарная конъюнкция A *покрывает* элементарную конъюнкцию B, если она является частью этой конъюнкции, т. е. целиком входит в нее.

Пример 4.18. Элементарная конъюнкция $x_1 \wedge \bar{x}_2$ покрывает элементарные конъюнкции $x_1 \wedge \bar{x}_2 \wedge x_3$ и $x_1 \wedge \bar{x}_2 \wedge \bar{x}_3$, но не покрывает элементарные конъюнкции $x_1 \wedge x_2 \wedge x_3$ и $\bar{x}_1 \wedge \bar{x}_2 \wedge \bar{x}_3$.

Нахождение минимальной ДНФ состоит в выборе таких простых импликантов из сокращенной ДНФ, которые в совокупности покрывают все элементарные конъюнкции СДНФ и содержат минимальное число вхождений переменных. Такая ДНФ равносильна СДНФ, поскольку ее значения на некотором наборе переменных совпадают со значениями СДНФ.

Рассмотрим следующий алгоритм нахождения минимальной ДНФ.

Алгоритм построения минимальной ДНФ с помощью таблицы покрытий

Шаг 1. Составление таблицы покрытий.

Для данной функции

$$f(x_1, x_2, \dots, x_n) = \bigvee_{i=1}^{k} A_i \equiv \bigvee_{j=1}^{m} B_j,\ m \le k, \qquad (4.2)$$

где A_i – элементарные конъюнкции, входящие в СДНФ, $i = 1, 2, \dots, k$, $k \le n$, а B_j – простые импликанты сокращенной ДНФ, $j = 1, 2, \dots, m$, $m \le k$, построим таблицу покрытий, число строк которой равно числу полученных простых импликантов, а число столбцов – числу элементарных конъюнкций в СДНФ. Если в некоторую элементарную конъюнкцию входит какой-либо простой импликант, то на пересечении соответствующего столбца и строки ставится метка (например, «*»).

Шаг 2. Выделение столбцов, содержащих одну метку.

Если в каких-нибудь столбцах составленной таблицы имеется только одна метка, то строки, в которых стоят эти метки, определяют простые импликанты, которые не могут быть исключены из сокращенной ДНФ, т. к. без них не может быть получено покрытие всех элементарных конъюнкций СДНФ. Они являются существенными и обязательно входит в минимальную ДНФ. Поэтому из таблицы покрытий исключаются строки, соответствующие существенным импликантам, и столбцы элементарных конъюнкций СДНФ, покрываемые этими существенными импликантами.

Шаг 3. Вычеркивание лишних столбцов.

Если в таблице есть столбцы, в которых имеются метки в одинаковых строках, то оставляем только один из них (все равно, какой), так как покрытие элементарных конъюнкций, соответствующих выброшенным столбцам, осуществляется за счет простого импликанта, соответствующего оставшемуся столбцу.

Шаг 4. Вычеркивание лишних существенных импликантов.

Если после выбрасывания некоторых столбцов в результате шага 3, в таблице появляются строки, в которых нет ни одной метки, то простые импликанты, соответствующие этим строкам, исключаются из дальнейшего рассмотрения, т. к. они не покрывают оставшиеся в рассмотрении элементарные конъюнкции СДНФ.

Шаг 5. Выбор минимального покрытия существенными импликантами.

Выбирается такая совокупность строк (т. е. существенных импликантов), чтобы они покрывали все оставшиеся столбцы (элементарные конъюнкции СДНФ). При нескольких возможных вариантах предпочтение отдается варианту покрытия с минимальным общим числом вхождения переменных.

Пример 4.19. Продолжим пример 4.18.

1. Составляем таблицу покрытий. Для формулы, булевой функции с СДНФ: $f(x_1, x_2, x_3) = \bar{x}_1 \wedge x_2 \wedge x_3 \vee x_1 \wedge \bar{x}_2 \wedge \bar{x}_3 \vee x_1 \wedge \bar{x}_2 \wedge x_3 \vee x_1 \wedge x_2 \wedge x_3$ была получена равносильная ей сокращенная ДНФ: $F_1(x_1, x_2, x_3) = x_2 \wedge x_3 \vee x_1 \wedge \bar{x}_2 \vee x_1 \wedge x_3$.

Каждой элементарной конъюнкции $x_1^{s1} \wedge x_2^{s2} \wedge \ldots \wedge x_n^{sn}$, ($x_i^{si} = x_i$, если $s_i = 0$ и $x_i^{si} = \bar{x}_i$, если $s_i = 1$, $i = 1, 2, \ldots, n$), входящей в СДНФ, можно сопоставить набор переменных из нулей и единиц. Этот наборы идентифицируют столбцы. Каждому простому импликанту из сокращенной ДНФ также можно сопоставить набор из нулей, единиц и прочерков, где 0 означает, что переменная берется с отрицанием, 1 – переменная берется без отрицания, «–» – переменная отсутствует. Для примера получаем следующую таблицу (табл. 4.5) из 4 столбцов, соответствующих 4 элементарным конъюнкциям СДНФ $F(x_1, x_2, x_3, x_4)$ и 3 строк, соответствующих 3 простым импликантам сокращенной ДНФ $F_1(x_1, x_2, x_3, x_4)$.

Таблица 4.5 – Пример построения таблицы покрытий

	011	100	101	111
–11	*			*

10–		*	*	
1–1			*	*

2. Выделяем столбцы, содержащие одну метку – это 1-й и 2-й столбцы. Импликант $x_2 \wedge x_3$ (ему соответствует 1-я строка) является существенным. Он покрывает две элементарные конъюнкции СДНФ: $\bar{x}_1 \wedge x_2 \wedge x_3$ и $x_1 \wedge x_2 \wedge x_3$ (им соответствуют 1-й и 4-й столбцы). Импликант $x_1 \wedge \bar{x}_2$ (ему соответствует 2-я строка) тоже является существенным. Он покрывает две элементарные конъюнкции СДНФ: $x_1 \wedge \bar{x}_2 \wedge \bar{x}_3$ и $x_1 \wedge \bar{x}_2 \wedge x_3$ (им соответствуют 2-й и 3-й столбцы).

Все указанные строки (1-ю и 2-ю) и столбцы (1-й, 2-й, 3-й и 4-й) вычеркиваем из таблицы покрытий. После этого все элементы таблицы окажутся вычеркнутыми. Следовательно, два существенных импликанта $x_2 \wedge x_3$ и $x_1 \wedge \bar{x}_3$ покрывают все элементарные конъюнкции СДНФ.

Итак, минимальная ДНФ для нашей функции имеет вид:

$F_2(x_1, x_2, x_3) = x_2 \wedge x_3 \vee x_1 \wedge \bar{x}_2$.

Рассмотрим еще один пример нахождения минимальной ДНФ булевой функции.

Пример 4.20. Пусть булева функция задана таблицей (табл. 4.6).

Применим вначале алгоритм Квайна – Мак-Класки для нахождения сокращенной ДНФ.

Таблица 4.6 – Представление булевой функции 4-х переменных в виде таблицы

x_1	x_2	x_3	x_4	$f(x_1,x_2, x_3, x_4)$
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Представленная в таблице булева функция имеет следующую формулу в СДНФ: $F(x_1, x_2, x_3, x_4) = (\bar{x}_1 \wedge \bar{x}_2 \wedge x_3 \wedge x_4) \vee (\bar{x}_1 \wedge x_2 \wedge \bar{x}_3 \wedge \bar{x}_4) \vee (\bar{x}_1 \wedge x_2 \wedge \bar{x}_3 \wedge x_4) \vee (\bar{x}_1 \wedge x_2 \wedge {}_x3 \wedge x_4) \vee (x_1 \wedge \bar{x}_2 \wedge \bar{x}_3 \wedge x_4) \vee (x_1 \wedge \bar{x}_2 \wedge x_3 \wedge x_4) \vee (x_1 \wedge x_2 \wedge \bar{x}_3 \wedge \bar{x}_4) \vee (x_1 \wedge x_2 \wedge \bar{x}_3 \wedge x_4)$.

Выпишем наборы переменных, на которых функция принимает значение 1, причем эти наборы упорядочим по группам так, что в каждую группу входят наборы с одинаковым числом единиц.

Группы A_0 нет.

Группа A_1:

0 1 0 0.

Группа A_2:

0 0 1 1;
0 1 0 1;
1 0 0 1;
1 1 0 0.

Группа A_3:

0 1 1 1;
1 0 1 1;
1 1 0 1.

Группы A_4 нет.

Производим попарное сравнение наборов переменных, входящих в соседние группы.

При сравнении групп A_1 и A_2:

вместо (0 1 0 0) и (0 1 0 1) получим (0 1 0 –);
вместо (0 1 0 0) и (1 1 0 0) получим (– 1 0 0).

При сравнении групп A_2 и A_3:

вместо (0 0 1 1) и (1 0 1 1) получим (0 – 1 1);
вместо (0 0 1 1) и (1 0 1 1) получим (– 0 1 1);
вместо (0 1 0 1) и (0 1 1 1) получим (0 1 – 1);
вместо (0 1 0 1) и (1 1 0 1) получим (– 1 0 1);
вместо (1 0 0 1) и (1 0 1 1) получим (1 0 – 1);
вместо (1 0 0 1) и (1 1 0 1) получим (1 – 0 1);
вместо (1 1 0 0) и (1 1 0 1) получим (1 1 0 –).

После этого этапа массив R пуст, т. к. все наборы участвовали в образовании наборов с прочерками, а массив $P = P(1)$ включает следующие наборы:

(0 1 0 –);
(– 1 0 0);
(0 – 1 1);
(– 0 1 1);
(0 1 – 1);
(– 1 0 1);
(1 0 – 1);

(1 – 0 1);
(1 1 0 –).

Теперь попарно сравниваются между собой наборы с прочерками. Наборы с одним прочерком, не участвовавшие в образовании наборов с двумя прочерками, помещаются в массив *R*.

Для примера, вместо (0 1 0 –) и (1 1 0 –) получим (– 1 0 –);

вместо (– 1 0 0) и (– 1 0 1) получим (– 1 0 –).

После этого этапа в массив *R* попадают наборы, не участвовавшие в образовании наборов с двумя прочерками:

(0 – 1 1);
(– 0 1 1);
(0 1 – 1);
(1 0 – 1);
(1 – 0 1).

Массив *P*(2) состоит из набора с двумя прочерками:

(– 1 0 –).

Набор с двумя прочерками один и процедура сравнения заканчивается. Поэтому все наборы из *P*(2) попадают в массив *R*, который после этого включает наборы:

(0 – 1 1);
(– 0 1 1)
(0 1 – 1);
(1 0 – 1);
(1 – 0 1);
(– 1 0 –).

Сокращенная ДНФ имеет вид: $F_1(x_1, x_2, x_3, x_4) = \bar{x}_1 \wedge x_3 \wedge x_4 \vee \bar{x}_2 \wedge x_3 \wedge \wedge x_4 \vee \bar{x}_1 \wedge x_2 \wedge x_4 \vee x_1 \wedge \bar{x}_2 \wedge x_4 \vee x_1 \wedge \bar{x}_3 \wedge x_4 \vee x_2 \wedge \bar{x}_3$.

Найдем теперь минимальную ДНФ с помощью таблицы покрытий. Составляем таблицу покрытий.

Для примера 4.20 получим следующую таблицу (табл. 4.7) из 8 столбцов, соответствующих 8 элементарным конъюнкциям СДНФ $F(x_1, x_2, x_3, x_4)$ и 6 строк, соответствующих 6 простым импликантам сокращенной ДНФ $F_1(x_1, x_2, x_3, x_4)$.

Таблица 4.7 – Таблица покрытий нахождения минимальной ДНФ (до преобразований)

	0011	0100	0101	0111	1001	1011	1100	1101
0–11	*			*				
–011	*					*		
01–1			*	*				
10–1					*	*		

1–01					*			*
–10–		*	*				*	*

Выделяем столбцы, содержащие одну метку – это 2-й и 7-й столбцы. Оба этих столбца определяют один и тот же импликант $x_2 \wedge \bar{x}_3$ (ему соответствует 6-я строка), который является существенным. Он покрывает следующие четыре элементарные конъюнкции СДНФ: $\bar{x}_1 \wedge x_2 \wedge \bar{x}_3 \wedge x_4$, $\bar{x}_1 \wedge x_2 \wedge \bar{x}_3 \wedge \bar{x}_4$, $x_1 \wedge x_2 \wedge \bar{x}_3 \wedge \bar{x}_4$, $x_1 \wedge x_2 \wedge \bar{x}_3 \wedge x_4$ (им соответствуют 2-й, 3-й, 7-й и 8-й столбцы). Все указанные строки и столбцы вычеркиваем из таблицы покрытий. После этого таблица примет вид (табл. 4.8):

Таблица 4.8 – Таблица покрытий нахождения минимальной ДНФ (после преобразований)

	0011	0111	1001	1011
0–11	*	*		
–011	*			*
01–1		*		
10–1			*	*
1–01			*	

В полученной таблице отсутствуют одинаковые столбцы, отсутствуют одинаковые строки. Выбирается такая совокупность существенных импликантов, которая покрывает все столбцы и содержит наименьшее количество букв. Это импликанты $\bar{x}_1 \wedge x_3 \wedge x_4$ и $x_1 \wedge \bar{x}_2 \wedge x_4$ (1-я и 4-я строки табл. 4.8), т. к. они покрывают все оставшиеся столбцы.

Таким образом, минимальная ДНФ для функции, представленной в табл. 4.6, имеет вид: $F_2(x_1, x_2, x_3, x_4) = \bar{x}_1 \wedge x_3 \wedge x_4 \vee x_1 \wedge \bar{x}_2 \wedge x_4 \vee x_2 \wedge \bar{x}_3$.

Контрольные вопросы к разделу 4

1. Что такое булева функция?
2. Назовите способы представления булевой функции.
3. Каковы особенности булевых функций?
4. Дать определение формулы в алгебре логики.
5. Что такое равносильность формул?
6. Что такое ДНФ, КНФ, СДНФ, СКНФ.
7. Раскрыть алгоритм нахождения ДНФ (КНФ) булевой функции.
8. Раскрыть алгоритм нахождения СДНФ (СКНФ) булевой функции.
9. Алгоритмы нахождения сокращенной ДНФ функции.
10. Алгоритм построения минимальной ДНФ с помощью таблицы покрытий.

МОДУЛЬ 2. ОСНОВЫ ТЕОРИИ КОДИРОВАНИЯ, ФОРМАЛЬНЫЕ ЯЗЫКИ И ГРАММАТИКИ, АВТОМАТЫ

5 ОСНОВНЫЕ ПОНЯТИЯ ТЕОРИИ КОДИРОВАНИЯ

5.1 Алфавитное и равномерное кодирование

Теория кодирования – это раздел теории информации, изучающий способы отождествления сообщений с отображающими их сигналами. Задачей теории кодирования является согласование источника информации с каналом связи. Например, это может быть обеспечение заданных значений помехоустойчивости при заданных характеристиках помех в канале связи, достижение максимальной скорости переработки информации при выполнении некоторых арифметических действий.

Рассмотрим основные понятия теории кодирования. Пусть заданы два алфавита A и B, состоящие из конечного числа символов: $A = \{a_1, a_2, \ldots, a_n\}$ и $B = \{b_1, b_2, \ldots, b_m\}$. Кортеж в алфавите A назовем словом $\alpha = a_1, a_2, \ldots, a_i$, где $i \in [1, n]$, число n показывает ***длину слова*** α и обозначается $n = l(\alpha)$:

$$l(\alpha) = |\alpha| = |a_1, \ldots, a_n| = n. \qquad (5.1)$$

Пустое слово обозначается λ: $l(\lambda) = |\lambda| = 0$. Для слова $\alpha = \alpha_1, \ldots, \alpha_2$ буква α_1 называется началом, или ***префиксом***, слова α, а буква α_2 – окончанием, или ***постфиксом***, слова α. Слова, как и кортежи, можно соединять. Для этого префикс второго слова должен следовать сразу за постфиксом первого, при этом в новом слове они, естественно, утрачивают свой статус, если только одно из слов не было пустым. Соединение слов α_1 и α_2 обозначается $\alpha_1\alpha_2$, соединение n одинаковых слов α обозначается α^n, причем $\alpha^0 = \lambda$.

Множество всех непустых слов алфавита А обозначим A^*: $A^* = \{\alpha \mid l(\alpha) > 0\}$. Множество А называют алфавитом сообщений, а множество В – кодирующим алфавитом. Множество слов, составленных в алфавите В, обозначим В*. Объектом кодирования служит как дискретная, так и непрерывная информация, которая поступает к потребителю через источник информации. Понятие ***кодировани*е** означает преобразование информации в форму, удобную для передачи по определенному каналу связи.

Обратная операция – ***декодирование*** – заключается в восстановлении принятого сообщения из закодированного вида в общепринятый, доступный для потребителя.

Правила обработки сообщений могут быть различными: поэлементное кодирование, с помощью алгоритма, кодирование с помощью различных

видов графики и др. Так как для любого кодирования должна выполняться операция декодирования, то отображение должно быть обратимым (биекция).

Если все кодовые слова имеют одинаковую длину, то код называется ***равномерным***, или ***блочным***.

Алфавитное кодирование. Пусть заданы конечный алфавит A = $\{a_1, ..., a_n\}$ и множество слов над алфавитом $A - A^*$. Назовем элементарной частью сообщения ту его минимальную часть, которой ставится в соответствие символ из множества B. Будем рассматривать простейший случай, когда элементарной частью сообщения является одна буква алфавита A.

Алфавитное, т. е. побуквенное, кодирование можно задать таблицей кодов. Фактически кодом, кодирующей функцией, будет служить некоторая подстановка σ. Тогда σ: $\langle \alpha_1 \to \sigma(\alpha_1), ..., \alpha_n \to \sigma(\alpha_n) \rangle = \langle \beta_{i_1}, ..., \beta_{i_n} \rangle$ или $\sigma = \begin{pmatrix} \alpha_1 & ... & \alpha_n \\ & & \\ \beta_1 & ... & \beta_n \end{pmatrix}$, где $\alpha_i \in A$, $\beta_i \in B^*$. Такое побуквенное кодирование обозначается $K_{\beta_i}^{\alpha_i}$. Множество кодов букв $\{\beta_i\}$ называется ***множеством элементарных кодов***. Алфавитное кодирование можно использовать для любого множества сообщений.

Таким образом, алфавитное кодирование является самым простым, его всегда можно ввести на непустых алфавитах.

Пусть, например, заданы алфавиты A и B, состоящие, соответственно, из {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} и {0, 1}. Тогда таблица кодирования может быть подстановкой:

$$\sigma = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 0 & 1 & 10 & 11 & 100 & 101 & 110 & 111 & 1000 & 1001 \end{pmatrix}.$$

Это двоично-десятичное кодирование, оно является взаимно-однозначным и потому допускает декодирование.

Однако схема $\sigma = \begin{pmatrix} 0 & 1 & 10 & 11 & 100 & 101 & 110 & 111 & 1000 & 1001 \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{pmatrix}$ не является взаимно-однозначной. Например, кортеж из шести единиц 111111 может соответствовать как слову 333, так и 77, 111111, а также 137, 3311 или 7111 плюс любая перестановка.

Пусть задана некоторая схема алфавитного кодирования $\sigma = \begin{pmatrix} \alpha_1 & ... & \alpha_n \\ & & \\ \beta_1 & ... & \beta_n \end{pmatrix}$. Тогда любая схема, $\sigma' = \begin{pmatrix} \alpha_1 & ... & \alpha_n \\ & & \\ \beta_1' & ... & \beta_n' \end{pmatrix}$, где кортеж есть перестановка кортежа $\langle \beta_1, ..., \beta_n \rangle$, также будет задавать некое кодирование. В таком случае, если длины элементарных кодов равны, то их перестановка в схеме не влияет на длину закодированного сообщения. В том случае, если длины элементарных кодов различны, то длина кода сообщения напрямую зависит и от того, какие элементарные коды каким буквам поставлены в соответствие, и от того, каков состав букв в сообщении.

Назовем схему σ ***префиксной***, если элементарный код одной буквы не является префиксом кода другой буквы. Схема σ называется ***разделимой***, если любое слово, составленное из элементарных кодов, разлагается на элементарные коды единственным образом. Алфавитное кодирование с разделимой схемой допускает декодирование.

В зависимости от целей кодирования, различают следующие его виды:

- кодирование по образцу используется всякий раз при вводе информации в компьютер для её внутреннего представления;
- криптографическое кодирование, или шифрование, используется, когда нужно защитить информацию от несанкционированного доступа;
- эффективное, или оптимальное, кодирование используется для устранения избыточности информации, т. е. снижения ее объема, например, в архиваторах;
- помехозащитное, или помехоустойчивое, кодирование используется для обеспечения заданной достоверности в случае, когда на сигнал накладывается помеха, например, при передаче информации по каналам связи.

5.2 Системы исчисления как основа разных кодов

Кодирование информации о численности чего-либо привело к созданию систем исчисления.

Десятичная система исчисления – способ кодирования натуральных чисел, причем основание системы счисления проистекало от количества пальцев на руках. История знает и другие системы – двадцатеричная – по числу пальцев на руках и ногах. С помощью букв обозначали цифры и греки, и русские, и грузины. Индийцы и арабы придумали нуль и позиционную систему исчисления.

Основной единицей кодирования информации в настоящее время является бит – один двоичный разряд, который может быть либо нулем (0), либо единицей (1). Большинство ЭВМ используют двоичное представление информации. В начале эры компьютеров битами кодировали числа и команды. Оказалось, что такое кодирование пригодно для записи, хранения и переработки практически любой информации: изображений (видеоинформации), звука, музыки, фильмов и т. д.

Группы в три бита называют триадами, группы из четырех битов – тетрадами. Наиболее часто сейчас говорят о байтах – группах из восьми битов. Два байта – это уже слово. Килобайт – это 2^{10}байт –1024 байта. Мегабайт – это 2^{20}байт. Гигабайт – это 2^{30}байт. Терабайт – это 2^{40} байт.

Двоичная система исчисления. В двоичной системе исчисления основание – 2, соответственно, используется только два символа: 0, 1. Поэтому арифметика очень простая: 0 + 0 = 0, 0+1 = 1, 1 + 0 – 1, 1 + 1 = 10 (0 и перенос в следующий разряд). Запись 11011,101 некоторого двоичного числа соответствует следующему десятичному числу: $1\cdot2^4 + 1\cdot 2^3 + 1\cdot2^2 + 1\cdot2^1 + 1\cdot2^0$, $1\cdot2^{-1} + 1\cdot 2^{-2} + 1\cdot2^{-3} = 27{,}625$.

Такое представление числа называется представлением с фиксированной запятой. Запятая фиксирована в соответствующей аппаратуре (арифметико-логическом устройстве – АЛУ). Рассмотрим, как выполняется двоичное суммирование.

Пример 5.1. Пусть необходимо сложить два двоичных числа 110 и 11 : 110 + 11 = 1001.

Пример 5.2. Двоичное вычитание, например, (10 – 5 в десятичном коде) выглядит следующим образом: 1010 – 0101 = 0101. Для выполнения такой операции вычитание заменяют сложением числа в так называемом обратном коде, когда с разрядами вычитаемого проводят операцию инвертирования: 1010 + 1010 = 1 0100. При этом образуется перенос 1, который необходимо сложить с промежуточным результатом 0100 : 0100 + 1 = 0101.Таким образом, получаем ответ 0101 (10 – 5 = 5 в десятичном коде).

Перевод чисел из десятичной в двоичную систему счисления выполняют, например, путем деления на основание системы счисления – 2 и записывают соответствующие остатки либо подбирают соответствующие степени числа 2.

Пример 5.3. Дано десятичное число 38, ближайшая степень числа 2 – это число 32, т. е. 2^5, остается еще 6, ближайшая степень числа 2 – это 4, т. е. 2^2, остается 2, это 2^1. Таким образом: $0 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 100110$.

Восьмеричная система счисления. В восьмеричном коде восемь символов: 0 (000 двоичное), 1 (001 двоичное), 2 (010 двоичное), 3 (011 двоичное), 4 (100 двоичное), 5 (101 двоичное), 6 (110 двоичное), 7 (111 двоичное).

Для перевода числа из восьмеричного кода в двоичный необходимо каждую цифру восьмеричного кода заменить на триаду (три символа) двоичного.

Пример 5.4. $354_8 = 11\ 101\ 100_2$. Здесь двоичное число представлено в байтовом формате, поэтому старшая триада неполная. Наоборот, из двоичного кода в восьмеричный: $10\ 111\ 101_2 = 275_8$.

Шестнадцатеричная система счисления. В шестнадцатеричном коде шестнадцать символов: 0 (0000 двоичное), 1 (0001 двоичное), 2 (0010 двоичное), 3 (0011 двоичное), 4 (0100 двоичное), 5 (0101 двоичное), 6 (0110 двоичное), 7 (0111 двоичное), 8 (1000 двоичное), 9 (1001 двоичное), A (1010 двоичное), B (1011 двоичное), C (1100 двоичное), D (1101 двоичное), E (1110 двоичное), F (1111 двоичное).

Для перевода числа из двоичной системы счисления в шестнадцатеричную необходимо заменить каждую тетраду (четыре символа) соответствующим шестнадцатеричным символом.

Пример 5.5. $1010\ 1101_2 = AD_{16}$. Наоборот, из шестнадцатеричного кода в двоичный: $7BAF_{16} = 111\ 1011\ 1010\ 1111_2$. В восьмеричном и шестнадцатеричном кодах можно строить соответствующие арифметики.

Пример 5.6. В шестнадцатеричном коде: $EF_{16} + AC_{16} = 19B_{16}$ (F + C = 1B, 1 переносится в следующий разряд; E + A = 18, и +1 из предыдущего разряда = 19). В восьмеричном коде: $37_8 + 21_8 = 60_8$.

5.3 Кодирование с минимальной избыточностью

Оптимальным является кодирование, при котором сообщение имеет наименьшую, из возможных, длину. Но для выявления наилучшего из вариантов необходима дополнительная информация. Например, для сообщений, представленных на естественном языке, такой дополнительной информацией может быть распределение вероятности появления букв в некотором тексте. Тогда задача построения оптимального кода приобретает точную математическую формулировку и строгое решение.

Первая теорема Шеннона о передаче информации, которая называется также основной теоремой о кодировании при отсутствии помех, формулируется следующим образом: «При отсутствии помех передачи всегда возможен такой вариант кодирования сообщения, при котором среднее число знаков кода, приходящихся на один знак кодируемого алфавита, будет сколь угодно близко к отношению средних информаций на знак первичного и вторичного алфавитов».

Используя понятие избыточности кода, можно дать более короткую формулировку теоремы: «При отсутствии помех передачи всегда возможен такой вариант кодирования сообщения, при котором избыточность кода будет сколь угодно близкой к нулю».

Эта теорема открывает принципиальную возможность оптимального кодирования. Рассматривая систему двоичных кодов ($m = 2$) и то, что при равных длительностях и вероятностях каждый элементарный сигнал (0 или 1) несёт в себе 1 бит информации ($\log_2 m = 1$), опираясь на теорему Шеннона, имеем:

$$I_1^{(A)} \le K^{(2)}, \tag{5.2}$$

где $I_1^{(A)}$ – средняя информационная емкость исходного сообщения;

$K^{(2)}$ – длина кода (длина кодовой цепочки).

Тогда первая теорема Шеннона получает следующую интерпретацию: «При отсутствии помех передачи средняя длина двоичного кода может быть сколь угодно близкой к средней информации, приходящейся на знак первичного алфавита».

В двоичной системе кодирования относительная избыточность кода:

$$Q = 1 - \frac{I_1^{(A)}}{K^{(2)}}, \tag{5.3}$$

где Q – относительная избыточность кода.

Определение количества переданной информации при двоичном кодировании сводится к простому подсчету числа импульсов (единиц) и пауз (нулей). При этом возникает проблема выделения из потока сигналов

(последовательности импульсов и пауз) отдельных кодов. Приемное устройство фиксирует интенсивность и длительность сигналов. Элементарные сигналы (0 и 1) могут иметь одинаковые или разные длительности. Их количество в коде (длина кодовой цепочки), который ставится в соответствие знаку первичного алфавита, также может быть одинаковым (в этом случае код называется равномерным) или разным (неравномерный код). Наконец, коды могут строиться для каждого знака исходного алфавита (алфавитное кодирование) или для их комбинаций (кодирование блоков, слов).

Способы кодирования / декодирования информации

Эффективное кодирование используется для уменьшения объемов информации на носителе – сигнале. Для кодирования символов исходного алфавита используют двоичные коды переменной длины: чем больше частота символа, тем короче его код.

Эффективность кода определяется средним числом двоичных разрядов для кодирования одного символа – l_{cp} (цена или длина кодирования) по формуле

$$l_{cp} = \sum_{i=1}^{k} (p_i \cdot n_i), \qquad (5.4)$$

где k – число символов исходного алфавита;

n_s – число двоичных разрядов для кодирования символа s;

p_s – вероятность появления символа s, причем $\sum_{i=1}^{k} p_i = 1$

Существуют два классических метода эффективного кодирования: метод Фано и метод Хаффмана. Входными данными для обоих методов является заданное множество исходных символов для кодирования с их частотами (вероятностями); результат – эффективные коды.

Простая идея построения кода, близкого к оптимальному, принадлежит американскому математику Р. Фано. Сформулируем алгоритм кодирования ***методом Фано***. Этот метод требует упорядочения исходного множества символов по убыванию их частот. Затем выполняются следующие шаги:

Шаг 1. Список символов делится на две части (назовем их первой и второй частями) так, чтобы суммы частот обеих частей (назовем их Σ1 и Σ2) были точно или примерно равны. В случае, когда точного равенства достичь не удается, разница между суммами должна быть минимальной;

Шаг 2. Кодовым комбинациям первой части дописывается 0, кодовым комбинациям второй части дописывается 1;

Шаг 3. Анализируют первую часть: если она содержит только один символ, работа с ней заканчивается: считается, что код для ее символов построен и выполняется переход к шагу 4 для построения кода второй части. Если символов больше одного, переходят к шагу 1 и процедура повторяется с первой частью как с самостоятельным упорядоченным списком;

Шаг 4. Анализируют вторую часть: если она содержит только один символ, работа с ней заканчивается и выполняется обращение к оставшемуся списку (шаг 5). Если символов больше одного, переходят к шагу 1 и процедура повторяется со второй частью как с самостоятельным списком;

Шаг 5. Анализируется оставшийся список: если он пуст – код построен, работа заканчивается; если нет – выполняется шаг 1.

Пример 5.5. Даны символы a, b, c, d с вероятностями p(a) = 0,45; p(b) = 0,25; p(c) = 0,2; p(d) = 0,1. Построить эффективный код методом Фано. Сведем исходные данные в таблицу, упорядочив их по убыванию вероятностей (табл. 5.1).

Таблица 5.1 – Исходные данные для кодирования по методу Фано

Исходные символы	**Вероятность символов**
a	0,45
b	0,25
c	0,2
d	0,1

Первая линия деления проходит под символом a: соответствующие суммы Σ1 и Σ2 приблизительно равны между собой и составляют 0,45 и 0,55. Тогда формируемым кодовым комбинациям дописывается 0 для верхней (первой) части и 1 для нижней (второй) части. Поскольку это первый шаг формирования кода, двоичные цифры не дописываются, а только начинают формировать код (табл. 5.2).

Таблица 5.2 – Формирование эффективного кода методом Фано (этап 1)

Исходные символы	**Вероятность символов**	**Формируемый код**
a	0,45	0
b	0,25	1
c	0,2	1
d	0,1	1

В силу того, что верхняя часть списка содержит только один элемент (символ a), работа с ней заканчивается, а эффективный код для этого символа считается сформированным (в таблице, приведенной выше, эта часть списка частот символов выделена заливкой). Второе деление выполняется под символом b: суммы частот Σ1 и Σ2 вновь равны между собой и равны 0,25. Тогда кодовой комбинации символов верхней части дописывается 0, а нижней части – 1. Таким образом, к полученным на первом шаге фрагментам кода, равным 1, добавляются новые символы (табл. 5.3).

Поскольку верхняя часть нового списка содержит только один символ (b), формирование кода для него закончено (соответствующая строка таблицы вновь выделена заливкой). Третье деление проходит между символами c и d: к кодовой комбинации символа c приписывается 0, коду символа d приписывается 1 (табл. 5.4).

Таблица 5.3 – Формирование эффективного кода методом Фано (этап 2)

Исходные символы	Вероятность символов	Формируемый код
a	0,45	0
b	0,25	10
c	0,2	11
d	0,1	11

Таблица 5.4 – Формирование эффективного кода методом Фано (этап 3)

Исходные символы	Вероятность символов	Формируемый код
a	0,45	0
b	0,25	10
c	0,2	110
d	0,1	111

Поскольку обе оставшиеся половины исходного списка содержат по одному элементу, работа со списком в целом заканчивается.

Таким образом, получили коды:

a – 0, b – 10, c – 110, d – 111.

Определим эффективность построенного кода по формуле (5.4):

$l_{ср} = 0{,}45 \cdot 1 + 0{,}25 \cdot 2 + 0{,}2 \cdot 3 + 0{,}1 \cdot 3 = 1{,}85$.

Поскольку при кодировании четырех символов кодом постоянной длины требуется два двоичных разряда, сэкономлено 0,15 двоичного разряда в среднем на один символ.

В 1952 г. американский математик Д. Хаффман предложил метод кодирования, суть которого заключалась в последовательном сжатии алфавита *A* до получения алфавита из двух букв. Оптимальная схема кодирования полученного алфавита очевидна: первую букву кодируют символом 0, вторую – символом 1. Затем последовательно расщепляют полученную схему. Предложенный метод получил название ***метод Хаффмана***. Метод Хаффмана имеет два преимущества, по сравнению с методом Фано: он устраняет неоднозначность кодирования, возникающую из-за примерного равенства сумм частот при разделении списка на две части (линия деления проводится неоднозначно) и имеет, в общем случае, большую эффективность кода. Исходное множество символов упорядочивается по убыванию вероятностей, и выполняются следующие шаги:

Шаг 1. Объединение вероятностей: две последние вероятности списка складываются, а соответствующие символы исключаются из списка; оставшийся после исключения символов список пополняется суммой вероятностей и вновь упорядочивается; предыдущие шаги повторяются до тех пор, пока ни получится единица в результате суммирования и список ни уменьшится до одного символа;

Шаг 2. Построение кодового дерева: строится двоичное кодовое дерево: корнем его является вершина, полученная в результате объединения вероятностей, равная 1; листьями – исходные вершины; остальные вершины

соответствуют либо суммарным, либо исходным вероятностям, причем для каждой вершины левая подчиненная вершина соответствует большему слагаемому, а правая – меньшему; ребра дерева связывают вершины-суммы с вершинами-слагаемыми. Структура дерева показывает, как происходило объединение вероятностей; ребра дерева кодируются: каждое левое кодируется нулём, каждое правое – единицей;

Шаг 3. Формирование кода: для получения кодов листьев (исходных кодируемых символов) продвигаются от корня к нужной вершине и «собирают» веса проходимых рёбер.

Пример 5.6. Даны символы a, b, c, d с вероятностями p(a) = 0,5; p(b) = 0,25; p(c) = 0,125; p(d) = 0,125. Построить эффективный код методом Хаффмана.

1. Объединение вероятностей (результат объединения двух последних вероятностей в списке выделен в правом соседнем столбце заливкой) (табл. 5.5):

Таблица 5.5 – Формирование эффективного кода методом Хаффмана

Исходные символы	**Вероятности p_s**	**Этапы объединения**		
		первый	**второй**	**третий**
a	0,5	0,5	0,5	1
b	0,25	0,25	0,5	
c	0,125	0,25		
d	0,125			

2. Построение кодового дерева (рис. 5.1):

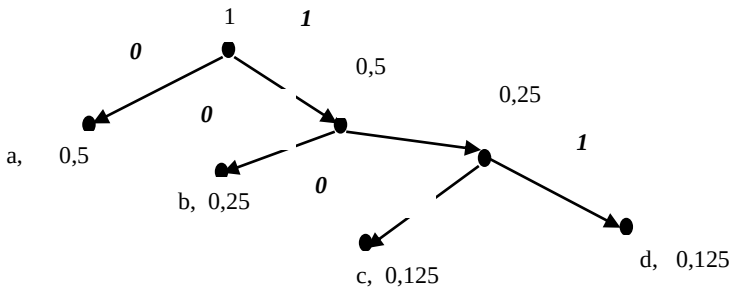


Рисунок 5.1 – Построение кодового дерева

3. Формирование кода: a – 0; b – 10; c – 110; d – 111.

Определим эффективность построенного кода по формуле (5.4): $l_{ср}$ = = 0,5 · 1 + 0,25 · 2 + 0,125 · 3 + 0,125 · 3 = 1,75.

Повышение эффективности кодирования

Повысить эффективность кодирования можно, строя код не для символа, а для блоков из n символов, причем частота блока рассчитывается как произведение частот символов, входящих в блок. Рассмотрим это на примере.

Пример 5.7. Даны символы *a* и *b* с вероятностями, соответственно, 0,9 и 0,1. Построить эффективный код методом Фано для блоков из двух символов ($n = 2$). Сформируем список возможных блоков и их вероятностей. При этом вероятность блока будем рассчитывать как произведение вероятностей символов, входящих в блок. Тогда имеем (табл. 5.6):

Таблица 5.6 – Исходные данные для построения эффективного кода методом Фано для блоков

Блоки исходных символов	**Вероятности блоков**
Aa	0,81
Ab	0,09
Ba	0,09
Bb	0,01

Построение кода сведём в таблицу (табл. 5.7).

Таблица 5.7 – Формирование эффективного кода методом Фано для блоков

Блоки исходных символов	**Вероятности блоков**	**Этапы построения кода**		
		первый	**второй**	**третий**
Aa	0,81	0	код построен	
Ab	0,09	1	0	код построен
Ba	0,09	1	1	0
Bb	0,01	1	1	1

Таким образом, получены коды:

Aa – 0; *Ab* – 10; *Ba* – 110; *Bb* – 111.

Определим эффективность построенного кода. Для этого рассчитаем сначала показатель эффективности для блока символов: $l_{ср} = 0{,}81 \cdot 1 + 0{,}09 \cdot 2 + 0{,}09 \cdot 3 + 0{,}01 \cdot 3 = 1{,}28$. Поскольку в блоке 2 символа ($n = 2$), для одного символа $l_{ср} = l_{ср}$блока / 2 = 1,28 / 2 = 0,64. При посимвольном кодировании для эффективного кода потребуется по одному двоичному разряду. Применение метода Фано даёт результат, представленный в таблице (табл. 5.8):

Таблица 5.8 – Формирование эффективного кода методом Фано для символов a и b поэлементно

Исходные символы	**Вероятности символов**	**Построение кода**
a	0,9	0
b	0,1	1

Таким образом, при блочном кодировании выигрыш составил 1 – 0,64 = 0,36 двоичных разрядов на один кодируемый символ в среднем.

Эффективность блочного кодирования тем выше, чем больше символов включается в блок.

Декодирование эффективных кодов

Особенностью эффективных кодов является переменное число двоичных разрядов в получаемых кодовых комбинациях. Это затрудняет процесс декодирования.

Рассмотрим вначале, как происходит декодирование сообщения, если использовались коды постоянной длины.

Пусть кодовая таблица имеет вид (табл. 5.9):

Таблица 5.9 – Кодовая таблица для декодирования

Исходные символы	**Двоичные коды**
a	00
b	01
c	10
d	11

Пример 5.8. Пусть закодированное сообщение – 001000011101.

Поскольку длина кода равна двум символам, в этом сообщении слева направо выделяются по два двоичных символа и сопоставляются с кодовой таблицей. Тогда имеем:

00 10 00 10 11 01

a c a b d b.

Таким образом, в исходном сообщении содержится текст *acabdb*. Декодирование выполнено.

Для декодирования кодов переменной длины рассмотренный подход не подходит. Но закодированные сообщения могут декодироваться благодаря свойству префиксности эффективных кодов: ни одна более короткая кодовая комбинация не является началом более длинной кодовой комбинации. Для раскрытия данного тезиса воспользуемся построенными ранее эффективными кодами: *a* – 0; *b* – 10; *c* – 110; *d* – 111.

Здесь самым коротким кодом является код для символа *a* со значением 0. Как видно, ни один другой код (более длинный) не имеет в начале символ 0. Второй по длине код для символа *b* имеет значение 10 и, как показывает анализ, не является началом ни для кода 110, ни для кода 111. Таким образом, данный код является префиксным. Свойство префиксности позволяет декодировать сообщения, закодированные эффективными кодами.

Пусть получено сообщение 0101101110, составленное из кодов *a* – 0; *b* – 10; *c* – 110; *d* – 111. В сообщении слева направо выделяется по одному двоичному символу и делается попытка декодирования в соответствии с заданной таблицей кодов. Если попытка успешна, двоичный символ (или символы) исключается из исходной цепочки и заменяется соответствующим исходным символом. Если попытка не удается, во входной цепочке выделяется следующий двоичный символ и уже с двумя двоичными

символами делается попытка их декодирования по таблице кодов. Если попытка и тогда неудачна, выделяют следующий третий и т.д.

Итак, имеем (направление просмотра цепочки слева направо):

0 1 0 1 1 0 1 1 1 0

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓.

Таким образом, при декодировании получили строку *abcda*.

5.4 Понятие помехоустойчивого кодирования. Код Хемминга

Итак, как было рассмотрено ранее, код представляет собой совокупность символов в представлении информации. Каждому знаку соответствует определенная комбинация нулей и единиц (бинарное кодирование).

Пусть $\{A_1, A_2, A_3, \ldots A_s\}$ – некоторое конечное подмножество попарно различных слов в алфавите I, имеющих одинаковую длину m ($s = 2^m$). Предположим, что в канале связи действует источник помех, который в словах из $\{A_1, A_2, A_3, \ldots A_s\}$, имеющих длину примерно m, может вызывать ошибки не более чем в p символах. Это значит, что двоичная последовательность, полученная на выходе канала, отличается от двоичной последовательности, поступившей на вход этого канала, не более чем в p позициях. Если передавать исходное сообщение $\alpha_1 \ldots \alpha_m$ без предварительного кодирования, то на выходе канала невозможно будет установить, какое сообщение фактически было передано. Для того чтобы однозначно восстановить код и получить исходное сообщение, существуют самокорректирующиеся коды (помехоустойчивые коды). Р. Хемминг предложил в 1950 г. коды для выявления и исправления ошибок в случае, когда источник помех может внести не более одной ошибки инвертирования в элементарный код B_i.

Рассмотрим следующую схему равномерного кодирования:

$$\begin{aligned} A_1 &\to B_1 \\ A_2 &\to B_2 \\ A_3 &\to B_3 \\ A_s &\to B_s, \end{aligned} \qquad (5.5)$$

где $B_i = \beta_1 \beta_2 \beta_3 \ldots \beta_l$ – элементарные двоичные коды.

Элементарные двоичные коды $\beta_1 \ldots \beta_l$ имеют одинаковую длину $l = m + k$. При наличии источника помех возможны следующие варианты $l+1$ способов получения кодов на выходе (правильный и l штук с инвертированием β_i на каждой позиции). Для того чтобы дополнительных разрядов в коде $\beta_1 \ldots \beta_l$ хватало для кодирования $l+1$ случаев, необходимо выполнение следующего условия:

$$2^m \le \frac{2^l}{l+1}. \qquad (5.6)$$

Из этих соображений выбираем l как наименьшее целое число, удовлетворяющее неравенству (5.6).

Дальнейшее построение кода Хемминга будет состоять из трех этапов:

1. Построение кодов Хемминга (описание алгоритма кодирования);
2. Обнаружение ошибок в кодах Хемминга;
3. Декодирование.

***Построение кодов Хемминга (описание алгоритма кодирования*)**

Разобьем множество натуральных чисел {1, 2, 3, …, l } на следующие подмножества:

подмножество {1, 3, 5, 7, 9, ….} включает все числа, у которых при переводе в двоичную запись в последнем разряде содержится 1;

подмножество {2, 3, 6, 7, 10, …} включает все числа, у которых при переводе в двоичную запись в предпоследнем разряде содержится 1;

. . . . . ;

подмножество $\{2^{k-1}, 2^{k-1}+1, \ldots.\}$ включает все числа, у которых при переводе в двоичную запись в k-ом, считая справа, разряде содержится 1.

Наименьшими членами этих подмножеств являются числа со степенями $2 : 1 = 2^0$; $2 = 2^1$; $4 = 2^2$;…., причем $2^{k-1} \le l$, а $2^{k+1} \ge l + 1$.

Члены β_i набора $\beta_1 \ldots \beta_l$, у которых индекс i принадлежит подмножеству $\{1, 2, 4, \ldots 2^{k-1}\}$, называются ***контрольными членами***, остальные – ***информационными.*** Таким образом, контрольных членов будет k, а информационных $l - k = m$. Сформулируем правило построения набора $\beta_1 \ldots \beta_l$ по набору $\alpha_1\, \alpha_2\, \alpha_3 \ldots .\alpha_m$.

Сначала определяются информационные члены:

$$\begin{aligned} \beta_3 &= \alpha_1, \\ \beta_5 &= \alpha_2, \\ \beta_6 &= \alpha_3, \\ &\ldots\ldots \end{aligned}$$

Таким образом, набор информационных членов, расположенных в естественном порядке, совпадает с набором $\alpha_1\, \alpha_2\, \alpha_3 \ldots \alpha_m$. Затем определяются контрольные члены:

$$\begin{aligned} \beta_1 &= \beta_3 + \beta_5 + \beta_7 + \ldots (\text{mod } 2), \\ \beta_2 &= \beta_3 + \beta_6 + \beta_7 + \ldots (\text{mod } 2), \\ \beta_4 &= \beta_5 + \beta_6 + \beta_7 + \ldots (\text{mod } 2), \\ &\ldots\ldots\ldots\ldots \end{aligned} \qquad (5.7)$$

В этих формулах правые части состоят из информационных элементов, определенных ранее. Контрольные элементы принимают значения 0 или 1 и помещаются на соответствующих местах в элементарных кодах B_i. Количество единиц в позициях элементарного кода, определяемых индексами, четно.

Обнаружение ошибок в кодах Хемминга. Пусть при передаче кода $\beta_1 \ldots \beta_l$ произошла ошибка в S-ом члене. Тогда на выходе канала принято слово $\beta'_1 \ldots \beta'_l$, в котором $\beta'_1 \ldots \beta'_l = \beta_1 \ldots \beta'_s \ldots \beta_1$. Пусть $S = S_k \ldots S_1$ – запись числа в двоичном исчислении. Определить номер позиции S по коду $\beta'_1 \ldots \beta'_l$ можно следующим образом. Рассмотрим число $S' = S'_k \ldots S'_1$, где

$$\begin{aligned} S'_1 &= \beta'_1 + \beta'_3 + \beta'_5 + \beta'_7 + \ldots \\ S'_2 &= \beta'_2 + \beta'_3 + \beta'_6 + \beta'_7 + \ldots \\ S'_3 &= \beta'_4 + \beta'_5 + \beta'_6 + \beta'_7 + \ldots \end{aligned} \qquad (5.8)$$

Если при передаче сообщения ошибки не произошло, то $S' = 0$. Если ошибка произошла, число S' позволят узнать номер члена сообщения, который исказился помехой. В последнем случае производят коррекцию ошибки путем инвертирования, т. е. 0 заменяют на 1 и наоборот.

Декодирование. Это шаг состоит в построении исходного сообщения $\alpha_1 \ldots \alpha_m$ по коду $\beta_1 \ldots \beta_l$. Для этого достаточно удалить все контрольные члены и взять только информационные.

Примечание. Рассмотрено построение кодов Хемминга для бинарного кодирования и числа ошибок в элементарных кодах не более одной. Существует теория для построения таких кодов для равномерного кодирования любой размерности и числе ошибок не более 2, 3,… и т. д.

Пример 5.8. Построить код Хэмминга для символа *c*(*110*) из примера 5.6. Код Хэмминга будет состоять из 6 членов (m = 6), которые включают 3 информационных члена ($l = 3$) и три контрольных ($k = 3$). Информационные члены: $\beta_3 = 1$, $\beta_5 = 1$, $\beta_6 = 0$. Воспользуемся формулами (5.7) и определим контрольные элементы: $\beta_1 = \beta_3 + \beta_5 = 1 + 1 = 0$ (*mod 2*), $\beta_2 = \beta_3 + \beta_6 = 1 + 0 = 1$ (*mod 2*), $\beta_4 = \beta_5 + \beta_6 = 1 + 0 = 1$ (*mod 2*). Таким образом, код Хэмминга составляет: 011110.

Пример 5.9. Пусть на вход канала связи поступил код 001011 (т. е. закодировано **1*01 (поз. 1, 2, 4 – контрольные члены заменены на символ «*»). На выходе получено 001001 (искажен 5-й член элементарного кода). Вычислим *S:*

$$\begin{gathered} S_1 = \beta_1 + \beta_3 + \beta_5 = 0 + 1 + 1 = 1 \ (mod\ 2), \\ S_2 = \beta_2 + \beta_3 + \beta_6 = 0 + 1 + 1 = 0 \ (mod\ 2), \\ S_3 = \beta_4 + \beta_5 + \beta_6 = 0 + 0 + 1 = 1 \ (mod\ 2). \\ S = 1\ 0\ 1 = 5 \ (в\ десятичной\ системе). \end{gathered}$$

Контрольные вопросы к разделу 5

1. Дать определение следующих понятий: кодирование, декодирование, код, префикс слова α, постфикс слова α.
2. Что такое равномерное (блочное) кодирование?
3. Что такое алфавитное кодирование?
4. Каковы достаточные условия однозначного декодирования?
5. Перечислите и охарактеризуйте основные системы исчисления.
6. Сформулируйте первую теорему Шеннона.
7. Что такое эффективность кода и эффективное кодирование?
8. Раскройте алгоритм построения эффективного кода методом Фано.
9. Раскройте алгоритм построения эффективного кода методом Хаффмана.
10. Раскройте алгоритм Хэмминга.

6 ОСНОВЫ ТЕОРИИ ФОРМАЛЬНЫХ ЯЗЫКОВ И ГРАММАТИК

6.1 Основные понятия теории формальных грамматик

Теория формальных грамматик – раздел дискретной математики, изучающий способы описания закономерностей, характеризующих всю совокупность правильных текстов того или иного языка. ***Формальные грамматики*** – это абстрактные системы, позволяющие с помощью единообразных процедур получать правильные тексты данного языка вместе с описанием их структуры. Теория формальных грамматик занимает центральное место в математической лингвистике, поскольку именно она позволяет моделировать наиболее существенный аспект функционирования языка – переработку смыслов в тексты и обратно. Вместе с тем она выделяется среди других разделов математической лингвистики большей сложностью математического аппарата (сходного с аппаратом теории алгоритмов и общей теории автоматов) и возникающими в ней математическими задачами. Формальные грамматики наиболее разработанных типов представляют собой системы, которые позволяют порождать или распознавать множества конечных последовательностей (цепочек), интерпретируемые обычно как множества правильных предложений, а также сопоставлять входящим в эти множества цепочкам описания их синтаксической структуры в терминах систем составляющих или деревьев подчинения. Рассмотрим основные понятия теории формальных грамматик.

Алфавит – это конечное множество символов.

Цепочкой символов в алфавите V называется любая конечная последовательность символов этого алфавита.

Цепочка, которая не содержит ни одного символа, называется ***пустой цепочкой.*** Для ее обозначения будем использовать греческую букву λ. Предполагается, что сама буква λ в алфавит V не входит; она лишь помогает обозначить пустую последовательность символов.

Если α и β – цепочки, то цепочка αβ (результат приписывания цепочки β в конец цепочки α) называется ***конкатенацией*** (или ***сцеплением***) цепочек α и β. Конкатенацию можно считать двуместной операцией над цепочками: $\alpha \cdot \beta = \alpha\beta$.

Пример 6.1. Если $\alpha = ab$ и $\beta = cd$, то $\alpha \cdot \beta = abcd$.

Для любой цепочки α справедливы равенства: $\alpha \cdot \lambda = \lambda \cdot \alpha = \alpha$.

Для любых цепочек α, β, γ справедливо $(\alpha \cdot \beta) \cdot \gamma = \alpha \cdot (\beta \cdot \gamma) = \alpha \cdot \beta \cdot \gamma$ (свойство ассоциативности операции конкатенации).

Обращением (или ***реверсом***) цепочки α называется цепочка, символы которой записаны в обратном порядке.

Обращение цепочки α будем обозначать α^R.

Пример 6.2. если $\alpha = abcdef$, то $\alpha^R = fedcba$.

Для пустой цепочки: $\lambda^R = \lambda$;

n-ой степенью цепочки α (будем обозначать α^n) называется конкатенация n цепочек α: $\alpha^n = \underbrace{\alpha\alpha \ldots \alpha\alpha\alpha}_{n}$.

Свойства степени: $\alpha^0 = \lambda$; $\alpha^n = \alpha\alpha^{n-1} = \alpha^{n-1}\alpha$.

Длина цепочки – это число составляющих ее символов (или длина последовательности символов).

Пример 6.3. Если $\alpha = abbcaad$, то длина α равна 7.

Длину цепочки α будем обозначать $|\alpha|$. Длина λ равна 0.

Через $|\alpha|^s$ обозначают число вхождений символа s в цепочку α.

Пример 6.4. $|babb|_a = 1$, $|babb|_b = 3$, $|babb|_c = 0$.

Обозначим через V^* множество, содержащее все цепочки в алфавите V, включая пустую цепочку λ.

Пример 6.5. Если $V = \{0, 1\}$, то $V^* = \{\lambda, 0, 1, 00, 11, 01, 10, 000, 001, 011, \ldots\}$.

Обозначим через V^+ множество, содержащее все цепочки в алфавите V, исключая пустую цепочку λ. Следовательно, $V^* = V^+ \cup \{\lambda\}$.

Вхождение слова $x \in V^*$ в слово $y \in V^*$ – это упорядоченная тройка слов (u, x, v), такая, что $y = uxv$.

При этом слово u называют ***левым***, а слово v – ***правым крылом*** указанного ***вхождения***. Слово x называют ***основой вхождения***. Слово x входит в слово y, если существует вхождение x в y. При этом также слово (цепочку) x называют ***подсловом*** (или ***подцепочкой***) слова (цепочки) y. Подцепочку x цепочки y называют ***началом*** (или ***префиксом***) цепочки y, если $y = xz$ для некоторой непустой цепочки z; если же для некоторой непустой цепочки z имеет место $y = zx$, то цепочку x называют ***концом*** (или ***постфиксом***) цепочки y.

Вхождения (u, x, v) и (s, z, t) слов x и z в одно и то же слово y ***не пересекаются***, если существуют такие (может быть, и пустые) слова p и q, что $y = uxpzt$ (и тогда $v = pzt$, а $s = uxp$) или $y = szqxv$ (и тогда $u = szq$, а $t = qxv$) (рис. 6.1). В противном случае говорят, что указанные ***вхождения пересекаются***.

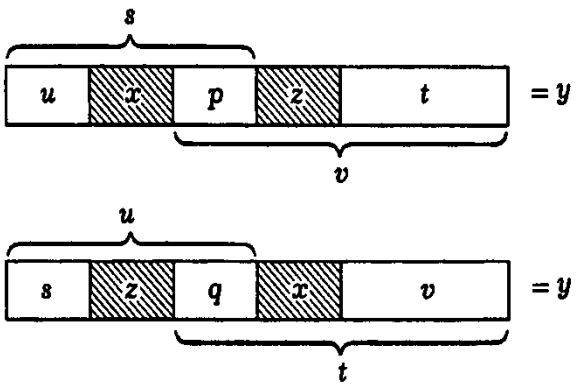


Рисунок 6.1 – Пример вхождения слов

Итерацией языка L называют объединение всех его степеней: $L^* = \bigcup_{n=0}^{\infty} L^n$. Рассматривая объединение всех степеней языка L, начиная с первой, получим позитивную итерацию $L^+ = \bigcup_{n=1}^{\infty} L^n$.

Сформулируем основное алгебраическое свойство множества всех языков в алфавите *V*.

Язык в алфавите V – это подмножество множества всех цепочек в этом алфавите. Для любого языка L справедливо $L \subseteq V^*$. Известны различные способы описания языков. Конечный язык можно описать простым перечислением его цепочек. Поскольку формальный язык может быть и бесконечным, требуются механизмы, позволяющие конечным образом представлять бесконечные языки. Можно выделить два основных подхода для такого представления: механизм распознавания и механизм порождения (генерации). Механизм распознавания (распознаватель), по сути, является процедурой специального вида, которая по заданной цепочке определяет, принадлежит ли она языку. Если принадлежит, то процедура останавливается с ответом «да», т. е. допускает цепочку; иначе – останавливается с ответом «нет» или зацикливается. Язык, определяемый распознавателем, – это множество всех цепочек, которые он допускает. Примеры распознавателей:

1. Машина Тьюринга (МТ). Язык, который можно задать с помощью МТ, называется рекурсивно перечислимым. Термин «рекурсивно перечислимый» происходит из теории рекурсивных функций, являющейся, также как НАМ и МТ, одной из формализаций понятия вычислимости (алгоритма). Вместо МТ можно использовать эквивалентные алгоритмические схемы: нормальный алгоритм Маркова (НАМ), машину Поста и др.;

2. Линейно ограниченный автомат (ЛОА). Представляет собой МТ, в которой лента не бесконечна, а ограничена длиной входного слова. ЛОА является недетерминированной машиной: в какой-то момент вычисления (во время работы ЛОА над заданной цепочкой) может возникнуть ситуация, когда есть две или более подходящие для исполнения команды, то есть выбор исполняемой команды не детерминирован. С этого момента возникают две или более копии ЛОА, соответствующие каждому возможному выбору; эти копии продолжают вычисления независимо (параллельно). Ответ «да» ЛОА дает, если хотя бы одно из вычислений успешно завершается. Известен алгоритм, позволяющий по любой недетерминированной МТ построить эквивалентную детерминированную МТ. Определяет контекстно-зависимые языки;

3. Автомат с магазинной (внешней) памятью (МП-автомат). В отличие от ЛОА, головка не может изменять входное слово и не может сдвигаться влево; имеется дополнительная бесконечная память (магазин, или стек), работающая по дисциплине LIFO. Определяет контекстно-свободные языки;

4. Конечный автомат (КА). Отличается от МП-автомата отсутствием магазина. Определяет регулярные языки.

Основной способ реализации механизма порождения – использование порождающих грамматик, которые иногда называют грамматиками Хомского. Не каждый формальный язык можно задать с помощью конечного описания. Действительно, само описание можно рассматривать как цепочку в некотором расширенном алфавите. Следовательно, множество описаний языков счётно, так как множество всех цепочек в заданном алфавите счётно. Каждый формальный язык в алфавите V является подмножеством счётного множества V^*. Из теории множеств известно, что множество всех подмножеств счётного множества является несчётным. Таким образом, мощность множества формальных языков больше мощности множества конечных описаний и, следовательно, не каждый язык представим в виде конечного описания.

Декартовым произведением $A \times B$ множеств A и B называется множество $\{ (a, b) \mid a \in A, b \in B\}$.

6.2 Формальные порождающие грамматики

Порождающая грамматика G – это четверка $\langle T, N, P, S \rangle$, где

T – терминальный алфавит, буквы этого алфавита называются терминальными символами (терминалами), из них строятся цепочки, порождаемые грамматикой, для обозначения букв терминального словаря используют строчные латинские буквы;

N – нетерминальный, вспомогательный алфавит (словарь) нетерминальных символов (нетерминалов) используется при построении цепочек. Они могут входить в промежуточные цепочки, но не должны входить в результата построения, для обозначения букв нетерминального словаря используют прописные латинские буквы, $T \cap N = \varnothing$;

P – конечное подмножество множества $(T \cup N)^+ \times (T \cup N)^*$; конечное множество **правил вывода или продукций**; элемент (α, β) множества P называется правилом вывода и записывается в виде $\alpha \rightarrow \beta$; α называется левой частью правила, β – правой частью; левая часть любого правила из P обязана содержать хотя бы один нетерминал;

S – начальный символ (цель, аксиома) грамматики, $S \in N$.

Для записи правил вывода с одинаковыми левыми частями $\alpha \rightarrow \beta_1$, $\alpha \rightarrow \beta_2$, ..., $\alpha \rightarrow \beta_n$ будем пользоваться сокращенной записью $\alpha \rightarrow \beta_1 \mid \beta_2 \mid ... \mid \beta_n$.

Каждое β_i ($i = 1, 2, ..., n$) будем называть альтернативой правила вывода из цепочки α.

Пример 6.6. Дана грамматика $G_{пример} = \langle \{0, 1\}, \{A, S\}, P, S \rangle$, где P состоит из правил:

$S \rightarrow 0A1$

$0A \rightarrow 00A1$

$A \rightarrow \lambda$.

Цепочка $\beta \in (T \cup N)^*$ ***непосредственно выводима*** из цепочки $\alpha \in (T \cup N)^+$ в грамматике $G = \langle T, N, P, S \rangle$ (обозначается $\alpha \rightarrow_G \beta$), если $\alpha = \xi_1 \gamma \xi_2$, $\beta = \xi_1 \delta \xi_2$, где $\xi_1, \xi_2, \delta \in (T \cup N)^*$, $\gamma \in (T \cup N)^+$ и правило вывода $\gamma \rightarrow \delta$ содержится в P. Индекс G в обозначении $\rightarrow_G$ обычно опускают, если понятно, о какой грамматике идет речь.

Пример 6.7. Цепочка 00A11 непосредственно выводима из 0A1 в грамматике $G_{пример}$: 0A1 → 00A11. Здесь цепочка 0A, подчеркнутая двойной чертой, играет роль подцепочки γ из определения, цепочка 00A1 играет роль подцепочки δ, $\xi 1 = \lambda$, $\xi 2 = 1$.

Языком, порождаемым грамматикой $G = \langle T, N, P, S \rangle$, называется множество $L(G) = \{\alpha \in T^* \mid S \Rightarrow \alpha\}$. Другими словами, $L(G)$ – это все цепочки в алфавите T, которые выводимы из S с помощью правил P. Например, $L(G_{пример}) = \{0^n 1^n \mid n > 0\}$.

Цепочка $\alpha \in (T \cup N)^*$, для которой $S \Rightarrow \alpha$, называется ***сентенциальной формой*** в грамматике $G = \langle T, N, P, S \rangle$.

Таким образом, язык, порождаемый грамматикой, можно определить как множество терминальных сентенциальных форм.

Грамматики G1 и G2 называются ***эквивалентными***, если $L(G1) = L(G2)$.

Пример 6.8. Грамматики $G1 = \langle \{0, 1\}, \{A, S\}, P1, S \rangle$ и $G2 = \langle \{0, 1\}, \{S\}, P2, S \rangle$ с правилами

P1:	*P2:*
$S \rightarrow 0A1$	$S \rightarrow 0S1 \mid 01$
$0A \rightarrow 00A1$	
$A \rightarrow \lambda$	

эквивалентны, т. к. обе порождают язык $L(G1) = L(G2) = \{0^n 1^n \mid n > 0\}$.

Грамматики G1 и G2 почти эквивалентны*, если* $L(G1) \cup \{\lambda\} = L(G2) \cup \{\lambda\}$. Другими словами, грамматики почти эквивалентны, если языки, ими порождаемые, отличаются не более чем на λ.

Пример 6.9. Почти эквивалентными являются следующие грамматики:

$G1 = \langle \{0,1\}, \{A, S\}, P1, S \rangle$ и $G2 = \langle \{0, 1\}, \{S\}, P2, S \rangle$ с правилами

P1:	A → λ,
S → 0A1	P2:
0A → 00A1	S →0S1 | λ,

так как $L(G1) = \{0^n 1^n \mid n > 0\}$, а $L(G2) = \{0^n 1^n \mid n \geq 0\}$, т. е. $L(G2)$ состоит из всех цепочек языка $L(G1)$ и пустой цепочки, которая в $L(G1)$ не входит.

6.3 Классификация грамматик и языков по Хомскому

Определим с помощью ограничений на вид правил вывода четыре типа грамматик: тип 0, тип 1, тип 2, тип 3. Каждому типу грамматик соответствует свой класс языков. Если язык порождается грамматикой типа i (для $i = 0, 1, 2, 3$), то он является языком типа i.

Тип 0

Любая порождающая грамматика является грамматикой типа 0. На вид правил грамматик этого типа не накладывается никаких дополнительных ограничений. Класс языков типа 0 совпадает с классом рекурсивно перечислимых языков.

Грамматики с ограничениями на вид правил вывода

Тип 1

Грамматика $G = (T, N, P, S)$ называется неукорачивающей, если правая часть каждого правила из P не короче левой части (т. е. для любого правила $\alpha \to \beta \in P$ выполняется неравенство $|\alpha| \le |\beta|$). В виде исключения в неукорачивающей грамматике допускается наличие правила $S \to \lambda$, при условии, что S (начальный символ, аксиома) не встречается в правых частях правил. Грамматикой типа 1 будем называть неукорачивающую грамматику. Тип 1 в некоторых источниках определяют с помощью так называемых контекстно-зависимых грамматик.

Грамматика G = ⟨T, N, P, S⟩ называется ***контекстно-зависимой (КЗ)***, если каждое правило из P имеет вид $\alpha \to \beta$, где $\alpha = \xi_1 A \xi_2$, $\beta = \xi_1 \gamma \xi_2$, $A \in N$, $\gamma \in (T \cup N)^+$, $\xi_1, \xi_2 \in (T \cup N)^*$.

В виде исключения в КЗ-грамматике допускается наличие правила с пустой правой частью $S \to \lambda$, при условии, что S (начальный символ) не встречается в правых частях правил. Цепочку ξ_1 называют ***левым контекстом***, цепочку ξ_2 называют ***правым контекстом***. Язык, порождаемый контекстно-зависимой грамматикой, называется контекстно-зависимым языком.

Тип 2

Грамматика G = ⟨T, N, P, S⟩ называется ***контекстно-свободной (КС)***, если каждое правило из P имеет вид $A \to \beta$, где $A \in N$, $\beta \in (T \cup N)^*$. Заметим, что в КС-грамматиках допускаются правила с пустыми правыми частями. Язык, порождаемый контекстно-свободной грамматикой, называется контекстно-свободным языком.

Грамматикой типа 2 будем называть контекстно-свободную грамматику. КС-грамматика может являться неукорачивающей, т. е. удовлетворять ограничениям неукорачивающей грамматики.

Тип 3

Грамматика G = ⟨T, N, P, S⟩ называется ***праволинейной***, если каждое правило из P имеет вид $A \to wB$ либо $A \to w$, где $A, B \in N$, $w \in T^*$.

Грамматика G = ⟨T, N, P, S⟩ называется ***леволинейной***, если каждое правило из P имеет вид $A \to Bw$ либо $A \to w$, где $A, B \in N$, $w \in T^*$.

Леволинейные и праволинейные грамматики определяют один и тот же класс языков. Такие ***языки*** называются ***регулярными***. Право- и леволинейные грамматики также будем называть ***регулярными***.

Регулярная грамматика является грамматикой типа 3.

Автоматной грамматикой называется такая праволинейная (леволинейная) грамматика, что каждое правило с непустой правой частью имеет вид: $A \to a$ либо $A \to aB$ (для леволинейной, соответственно, $A \to a$ либо $A \to Ba$), где $A, B \in N$, $a \in T$.

Автоматная грамматика является более простой формой регулярной грамматики. Существует алгоритм, позволяющий по регулярной (право- или леволинейной) грамматике построить соответствующую автоматную грамматику. Таким образом, любой регулярный язык может быть порожден автоматной грамматикой. Кроме того, существует алгоритм, позволяющий устранить из регулярной (автоматной) грамматики все λ-правила (кроме $S \to \lambda$ в случае, если пустая цепочка принадлежит языку; при этом S не будет встречаться в правых частях правил).

Для любой регулярной (автоматной) грамматики G существует такая неукорачивающая регулярная (автоматная) грамматика G', что $L(G) = L(G')$.

Иерархия Хомского

Справедливы следующие соотношения:

1) любая регулярная грамматика является КС-грамматикой;
2) любая неукорачивающая КС-грамматика является КЗ-грамматикой;
3) любая неукорачивающая грамматика является грамматикой типа 0.

Каждый регулярный язык является КС-языком, но существуют КС-языки, которые не являются регулярными, например: $L = \{a^n b^n | \ n > 0\}$; каждый КС-язык является КЗ-языком, но существуют КЗ-языки, которые не являются КС-языками, например: $L = \{a^n b^n c^n | \ n > 0\}$; каждый КЗ-язык является языком типа 0 (т. е. рекурсивно перечислимым языком), но существуют языки типа 0, которые не являются КЗ-языками, например: язык, состоящий из записей алгоритмов Маркова в некотором алфавите.

Таким образом, иерархия классов языков:

Тип 3 (Регулярные) $\subset$ Тип 2 (КС) $\subset$ Тип 1 (КЗ) $\subset$ Тип 0.

На рис. 6.1 иерархия Хомского для классов языков изображена в виде диаграммы Венна. Классы вкладываются друг в друга. Самый широкий класс языков (типа 0) содержит в себе все остальные классы.

Рисунок 6.1 – Иерархия классов языков

В примерах и задачах для краткости вместо полной четверки $G = \langle T, N, P, S \rangle$ будем иногда выписывать только так называемую «схему» грамматики – правила вывода P, считая, что большие латинские буквы обозначают нетерминальные символы, начальный символ (цель) грамматики обязательно стоит в левой части первого правила, λ – пустая цепочка, все остальные символы – терминальные.

Пример 6.10. Даны грамматики *G1* и *G2*:

G1:
S → 0A1
A → 0A0
A → λ

G2:
S → aSb | λ.

(1) Какого типа язык *L(G1)*? (2) Какого типа язык *L(G2)*?

1. Правила грамматики *G1* удовлетворяют ограничениям на правила КС-грамматик, но не удовлетворяет ограничениям регулярных грамматик, т. е. это грамматика типа 2, и поэтому *L(G1)* является языком типа 2 (следовательно, это язык и типа 1, и типа 0). Заметим, что $L(G1) = \{00^n0^n1 \mid n \geq 0\} = \{0^{(2n+1)}1 \mid n \geq 0\}$ и этот язык является также языком типа 3, поскольку описывается следующей регулярной грамматикой:

$S \rightarrow 0A$
$A \rightarrow 0B \mid 1$
$B \rightarrow 0A.$

Итак, максимальное *k*, для которого *L(G1)* является языком типа *k*, равно 3.

2. По виду правил *G2* является грамматикой типа 2 и не является грамматикой типа 3. Следовательно, *L(G2)* является языком типа 2. Язык *L(G2)* не является языком типа 3, так как не существует регулярной грамматики (т.е. грамматики типа 3), порождающей язык $L(G2) = \{a^nb^n \mid n \geq 0\}$. Итак, максимальное *k*, для которого *L(G2)* является языком типа *k*, равно 2.

Примеры грамматик и языков

Примеры грамматик и языков, иллюстрирующие классы иерархии Хомского, приведем в порядке, обратном классификации.

Регулярные

1. Грамматика $S \rightarrow aS \mid a$ является праволинейной (неукорачивающей) грамматикой. Она порождает регулярный язык $\{a^n \mid n > 0\}$. Этот язык может быть порожден и леволинейной грамматикой: $S \rightarrow Sa \mid a$. Обе эти грамматики являются автоматными.

2. Грамматика $S \rightarrow aS \mid \lambda$ является праволинейной и порождает регулярный язык $\{a^n \mid n \geq 0\}$. Для любого регулярного языка существует неукорачивающая регулярная грамматика. Неукорачивающая регулярная грамматика для данного языка:

$S \rightarrow aA \mid a \mid \lambda$
$A \rightarrow a \mid aA.$

Правило с пустой правой частью может применяться только один раз и только на первом шаге вывода; остальные правила таковы, что их правая часть не короче левой, т. е. грамматика неукорачивающая.

3. Грамматика:

$S \rightarrow A\perp \mid B\perp$
$A \rightarrow a \mid Ba$
$B \rightarrow b \mid Bb \mid Ab$ – леволинейная; она порождает регулярный язык, состоящий из всех непустых цепочек в алфавите $\{a, b\}$, заканчивающихся символом $\perp$ (маркер конца) и не содержащих подцепочку *aa*. То есть

в цепочках этого языка символ a не может встречаться два раза подряд, хотя бы один символ b обязательно присутствует между любыми двумя a. С помощью теоретико-множественной формулы данный язык описывается так: $\{\omega \perp \mid \omega \in \{a, b\}^+, aa \not\subset \omega \}$.

Контекстно-свободные

4. Грамматика:

$S \rightarrow aQb \mid accb$

$Q \rightarrow cSc$ является контекстно-свободной (неукорачивающей) и порождает КС-язык $\{(ac)^n(cb)^n \mid n > 0\}$, который, как и встречавшийся ранее язык $\{a^n b^n \mid n \geq 0\}$, не является регулярным, т. е. не может быть порожден ни одной регулярной грамматикой.

5. Грамматика:

$S \rightarrow aSa \mid bSb \mid \lambda$ порождает КС-язык $\{xx^R, x \in \{a, b\}\}^*$. Данный язык не является регулярным. Грамматика не удовлетворяет определению неукорачивающей, но для нее существует эквивалентная неукорачивающая грамматика. Приведем такую грамматику:

$S \rightarrow A \mid \lambda$

$A \rightarrow aAa \mid bAb \mid aa \mid bb.$

Неукорачивающие и контекстно-зависимые

6. Грамматика:

$S \rightarrow aSBC \mid abC$

$CB \rightarrow BC$

$bB \rightarrow bb$

$bC \rightarrow bc$

$cC \rightarrow cc$ неукорачивающая и порождает язык $\{a^n b^n c^n \mid n > 0\}$, который является языком типа 1, но не является языком типа 2.

Правило $CB \rightarrow BC$ не удовлетворяет определению КЗ-грамматики. Однако, заменив это правило тремя новыми $CB \rightarrow CD$, $CD \rightarrow BD$, $BD \rightarrow BC$ (добавляем новый символ D в множество нетерминалов N), получим эквивалентную серию контекстно-зависимых правил, которые меняют местами символы C и B. Таким образом, получаем эквивалентную КЗ-грамматику:

$S \rightarrow aSBC \mid abC$

$CB \rightarrow CD$

$CD \rightarrow BD$

$BD \rightarrow BC$

$bB \rightarrow bb$

$bC \rightarrow bc$

$cC \rightarrow cc.$

Без ограничений на вид правил (тип 0)

7. В грамматике типа 0

$S \rightarrow SS$

$SS \rightarrow \lambda$

второе правило не удовлетворяет ограничениям неукорачивающей грамматики. Существует бесконечно много выводов в данной грамматике, однако порождаемый язык конечен и состоит из единственной цепочки: $\{\lambda\}$.

Следующая грамматика также не является неукорачивающей и порождает пустой язык, так как ни одна терминальная цепочка не выводится из S (для обозначения пустого языка используется ø – знак пустого множества):

$S \rightarrow SS$

$SS \rightarrow Sa \mid S.$

Заметим, что языки $L_\lambda = \{\lambda\}$ и $L_ø = ø$ (пустой язык) могут быть описаны грамматиками типа 3. Кроме того, отметим, что $L_\lambda \neq L_ø$, так как $L_ø$ не содержит цепочек вообще, а L_λ – язык, состоящий из одной цепочки (пустая цепочка λ по определению равноправна с остальными цепочками в алфавите).

8. Содержательные примеры грамматик, порождающих языки, не принадлежащие классу контекстно-зависимых (тип 1), не приводятся из-за их громоздкости.

6.4 Деревья выводов

Цепочка в алфавите T принадлежит языку, порождаемому грамматикой $\langle T, N, P, S\rangle$, только в том случае, если существует ее вывод из начального символа S этой грамматики. Процесс построения такого вывода (а следовательно, и определения принадлежности цепочки языку) называется ***разбором*** (разбором также называют и результат этого процесса, т. е. вывод цепочки, представленный (зафиксированный) каким-нибудь способом).

Построение вывода можно осуществлять и в обратном порядке: в исходной цепочке ищем вхождение правой части некоторого правила и заменяем его на левую часть (это называется сверткой), в результате исходная цепочка «сворачивается» к некоторой сентенциальной форме, затем идет следующая свертка и т. д., пока не придем к цели грамматики – S. Процесс разбора называют также ***анализом***.

С практической точки зрения наибольший интерес представляет разбор по контекстно-свободным грамматикам**.** Их порождающей мощности достаточно для описания большей части синтаксической структуры языков программирования, для различных подклассов КС-грамматик имеются хорошо разработанные практически приемлемые способы решения задачи разбора.

Рассмотрим основные понятия и определения, связанные с разбором по КС-грамматике.

Вывод цепочки $\beta \in T^*$ из $S \in N$ в КС-грамматике $G = \langle T, N, P, S\rangle$, называется ***левым (левосторонним),*** если в этом выводе каждая очередная сентенциальная форма получается из предыдущей заменой самого левого нетерминала.

Вывод цепочки $\beta \in T^*$ из $S \in N$ в КС-грамматике $G = \langle T, N, P, S\rangle$, называется ***правым (правосторонним)***, если в этом выводе каждая очередная сентенциальная форма получается из предыдущей заменой самого правого нетерминала.

В грамматике для одной и той же цепочки может быть несколько выводов, эквивалентных в том смысле, что в них в одних и тех же местах применяются одни и те же правила вывода, но в различном порядке.

Пример 6.11. Для цепочки $a + b + a$ в грамматике:

$G = \langle \{a, b, +\}, \{S, T\}, \{S \to T \mid T + S ;\ T \to a \mid b\}, S \rangle$

можно построить выводы:

1. $S \to T + S \to T + T + S \to T + T + T \to a + T + T \to a + b + T \to a + b + a;$

2. $S \to T + S \to a + S \to a + T + S \to a + b + S \to a + b + T \to a + b + a;$

3. $S \to T + S \to T + T + S \to T + T + T \to T + T + a \to T + b + a \to a + b + a.$

Здесь 2 – левосторонний вывод, 3 – правосторонний, а 1 не является ни левосторонним, ни правосторонним, но все эти выводы являются эквивалентными в указанном выше смысле.

Для КС-грамматик можно ввести удобное графическое представление вывода, называемое ***деревом вывода***, причем для всех эквивалентных выводов деревья вывода совпадают.

Ориентированное упорядоченное дерево называется деревом вывода (или деревом разбора) в КС-грамматике $G = \langle T, N, P, S \rangle$, если выполнены следующие условия:

1) каждая вершина дерева помечена символом из множества $N \cup T \cup \{\lambda\}$, при этом корень дерева помечен символом S; листья – символами из $T \cup \{\lambda\}$;

2) если вершина дерева помечена символом A, а ее непосредственные потомки – символами $a_1, a_2, ..., a_n$, где каждое $a_i \in T \cup N$, то $A \to a_1 a_2 ... a_n$ – правило вывода в этой грамматике;

3) если вершина дерева помечена символом A, а ее непосредственный потомок помечен символом ε, то этот потомок единственный и $A \to \lambda$ – правило вывода в этой грамматике.

На рис. 6.2 изображен пример дерева для цепочки $a + b + a$ в грамматике G.

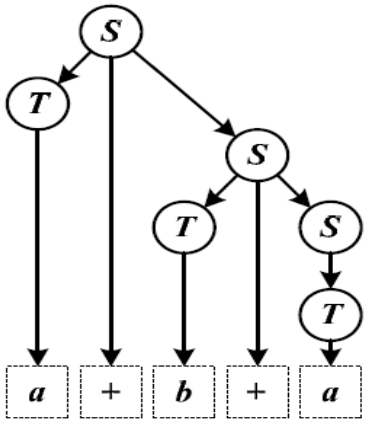


Рисунок 6.2 – Пример дерева вывода в грамматике G

КС-грамматика *G* называется ***неоднозначной***, если существует хотя бы одна цепочка $\alpha \in L(G)$, для которой может быть построено два или более различных деревьев вывода.

В противном случае грамматика называется ***однозначной***. Наличие двух или более деревьев вывода эквивалентно тому, что цепочка α имеет два или более разных левосторонних (или правосторонних) вывода.

Язык, порождаемый грамматикой, называется ***неоднозначным***, если он не может быть порожден никакой однозначной грамматикой.

Пример 6.12. Неоднозначной грамматикой является грамматика *Gif-then* = ⟨ {***if, then, else***, *a, b*}, {*S*}, *P, S* ⟩, где *P* = {*S* → ***if*** *b* ***then*** *S* ***else*** *S* | ***if*** *b* ***then*** *S* | *a*}.

В этой грамматике для цепочки ***if*** *b* ***then if*** *b* ***then*** *a* ***else*** *a* можно построить два различных дерева вывода, изображенных на рис. 6.3, а; б.

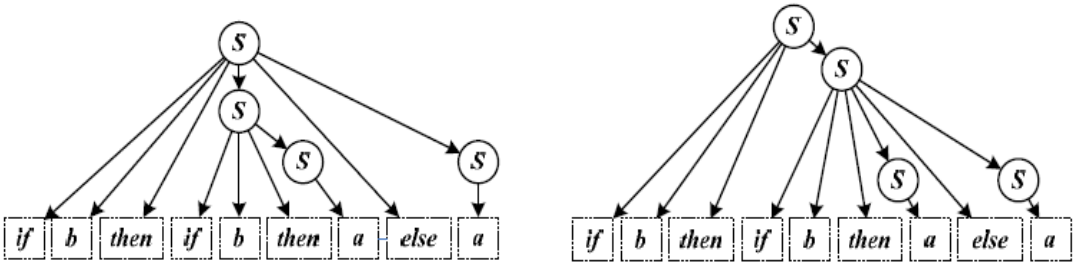
Однако это не означает, что язык $L(G_{if\text{-}then})$ обязательно неоднозначный. Обнаруженная в $G_{if\text{-}then}$ неоднозначность – это свойство грамматики, а не языка. Для некоторых неоднозначных грамматик существуют эквивалентные им однозначные грамматики.

Если грамматика используется для определения языка программирования, то она должна быть однозначной.

В приведенном выше примере разные деревья вывода предполагают соответствие *else* разным *then*. Если договориться, что *else* должно соответствовать ближайшему к нему *then*, и подправить грамматику $G_{if\text{-}then}$, то неоднозначность будет устранена:

S → ***if*** *b* ***then*** *S* | ***if*** *b* ***then*** *S′* ***else*** *S* | *a*

S′ → ***if*** *b* ***then*** *S′* ***else*** *S′* | *a*.



а) *б)*

Рисунок 6.3 – Деревья вывода для «if b if then b then a else» в грамматике $G_{if\text{-}then}$

Дерево вывода можно строить ***нисходящим*** либо ***восходящим*** способом. При ***нисходящем разборе*** дерево вывода формируется от корня к листьям; на каждом шаге для вершины, помеченной нетерминальным символом, пытаются найти такое правило вывода, чтобы имеющиеся в нем терминальные символы проецировались на символы исходной (анализируемой) цепочки.

Метод ***восходящего разбора*** основан на обратном построении вывода с помощью сверток от исходной цепочки к цели грамматики S. При этом дерево растет снизу вверх – от листьев (символов анализируемой цепочки) к корню S. Если грамматика однозначная, то при любом способе построения будет получено одно и то же дерево разбора.

6.5 Преобразования грамматик

Определение неоднозначна или однозначна заданная грамматика – это искусство поиска цепочки с двумя различными деревьями вывода или доказательство того, что таких цепочек не существует. Универсального способа решения этой задачи нет.

Однако можно указать некоторые виды правил вывода, которые приводят к неоднозначности (при условии, что эти правила не являются тупиковыми, т. е. действительно используются на каком-нибудь шаге вывода терминальной цепочки из начального символа).

В некоторых случаях КС-грамматика может содержать бесполезные символы, которые не участвуют в порождении цепочек языка и поэтому могут быть удалены из грамматики.

Символ $x \in T \cup N$ называется ***недостижимым*** в грамматике $G = \langle T, N, P, S \rangle$, если он не появляется ни в одной сентенциальной форме этой грамматики.

Если нетерминал в левой части правила P является достижимым, то достижимы все нетерминалы, стоящие в правой части этого правила.

Алгоритм удаления недостижимых символов

Дана КС-грамматика $G = \langle T, N, P, S \rangle$.

Шаг 1. Образовать одноэлементный список из начального нетерминала грамматики: $V_0 = \{S\}$; $i = 1$.

Шаг 2. Если в множестве P найдено правило, левая часть которого уже в списке, то включить в список все нетерминалы из его правой части: $V_i = V_{i-1} \cup \{x \mid x \in T \cup N, A \to \alpha x \beta \in P, A \in V_{i-1}, \alpha, \beta \in (T \cup N)^*\}$.

Шаг 3. Если $V_i \neq V_{i-1}$, то $i = i + 1$ и переходим к шагу 2, иначе получен исчерпывающий список всех достижимых нетерминалов и на предыдущем шаге список не пополняется: $N' = V_i \cap N$; $T' = V_i \cap T$; P' состоит из правил множества P, содержащих только символы из V_i; $G' = \langle T', N', P', S \rangle$.

Нетерминалы грамматики, не попавшие в список, построенный по приведенному выше алгоритму, являются недостижимыми и, не нарушая эквивалентности, из множества правил P можно удалить все правила, содержащие такие нетерминалы. В результате получаем КС-грамматику $G' = \langle T', N', P', S \rangle$, не содержащая недостижимых символов для которой $L(G) = L(G')$.

Символ $A \in N$ называется ***бесплодным*** в грамматике $G = \langle T, N, P, S \rangle$, если множество $\{\alpha \in T^* \mid A \Rightarrow \alpha\}$ пусто.

Алгоритм удаления бесплодных символов

Дана КС-грамматика $G = \langle T, N, P, S \rangle$.

Шаг 1. Составить список нетерминалов, для которых найдется хотя бы одно правило, правая часть которого состоит только из терминалов.

Шаг 2. Если найдется такое правило, что все нетерминалы, стоящие в его правой части, уже занесены в список, то добавить в список нетерминал, стоящий в его левой части.

Шаг 3. Если на предыдущем шаге список не пополняется, то получен исчерпывающий список всех продуктивных нетерминалов.

Нетерминалы грамматики, не попавшие в список, построенный по приведенному выше алгоритму, являются бесплодными и, не нарушая эквивалентности, из множества правил P можно удалить все правила, содержащие такие нетерминалы. В результате получаем КС-грамматику $G' = \langle T, N', P', S \rangle$, не содержащая бесплодных символов, для которой $L(G) = L(G')$.

КС-грамматика G называется ***приведенной***, если в ней нет недостижимых и бесплодных символов.

Пример 6.12. Дана контекстно-свободная грамматика G с правилами

$S \to U \mid VZ;$
$T \to aa \mid bb;$
$U \to aUa \mid bUb;$
$V \to aTb \mid bTa;$
$W \to YZY \mid aab;$
$X \to Xa \mid Xb \mid \lambda;$
$Y \to YY \mid aU \mid \lambda;$
$Z \to W \mid b.$

Удалив четыре правила, содержащие бесплодный символ U, получим грамматику $G1$ с правилами:

$S \to VZ;$
$T \to aa \mid bb;$
$V \to aTb \mid bTa;$
$W \to YZY \mid aab;$
$X \to Xa \mid Xb \mid \lambda;$
$Y \to YY \mid \lambda;$
$Z \to W \mid b.$

В полученной грамматике символ X является недостижимым. Удалив три правила, содержащие X, получим грамматику $G2$ с правилами:

$S \to VZ;$
$T \to aa \mid bb;$
$V \to aTb \mid bTa;$
$W \to YZY \mid aab;$
$Y \to YY \mid \lambda;$
$Z \to W \mid b.$

В результате *L (G) = L (G2)* и грамматика *G2* не содержит бесплодных и недостижимых символов.

Алгоритм приведения грамматики

Шаг 1. Обнаруживаются и удаляются все бесплодные нетерминалы.

Шаг 2. Обнаруживаются и удаляются все недостижимые символы.

Удаление символов сопровождается удалением правил вывода, содержащих эти символы (если начальный символ *S* – бесполезный в грамматике, то грамматика порождает пустой язык. Множество *P* после приведения такой грамматики будет пустым, однако *S* не следует удалять из *N*, так как алфавит *N* не может быть пустым и на последнем месте в четверке, задающей грамматику, должен стоять нетерминал – начальный символ).

Если в этом алгоритме приведения поменять местами шаги 1 и 2, то не всегда результатом будет приведенная грамматика.

Для описания синтаксиса языков программирования стараются использовать однозначные приведенные КС-грамматики.

Контрольные вопросы к разделу 6.

1. Что такое формальная грамматика? Каково практическое применение теории формальных грамматик?
2. Дать определение следующих понятий: конкатенация цепочек, пустая цепочка, обращение цепочки, длина цепочки.
3. Для алфавитов $A = \{a, b, c\}$, $B = \{0,1\}$ составить A^*, B^*, A^+, B^+.
4. Дать определение формального языка.
5. Каковы подходы к представлению формального языка?
6. Что такое распознаватели? Привести пример распознавателей.
7. Дать определение грамматики и ее компонентов.
8. Назвать типы грамматики согласно классификации Хомского. Описать каждый из типов. Привести примеры.
9. Что такое вывод в грамматике? Какие выводы называются эквивалентными?
10. Какие КС-грамматики называются приведенными?

7 ОСНОВЫ ТЕОРИИ КОНЕЧНЫХ АВТОМАТОВ

7.1 Общая характеристика автоматов

Под ***автоматом*** принято понимать абстрактную модель устройства, функционирующего в дискретном времени, которая перерабатывает последовательность входных сигналов и превращает их в последовательность выходных сигналов. В процессе работы автомата происходит последовательная смена его внутренних состояний, причем состояние автомата в определенный момент времени однозначно определяется входным и выходным сигналами. Применение автоматов позволяет моделировать любую машину, включая компоненты компьютера. Автоматы представляют собой основу всей современной вычислительной техники и всевозможных дискретных систем автоматического контроля и управления.

Используя такие вычислительные модели, как автоматы, можно исследовать вопрос решения и сложности разных задач. Как и грамматики, автоматы также используются во время исследования грамматической структуры языков.

Автомат можно изобразить в виде схемы (рис. 7.1).

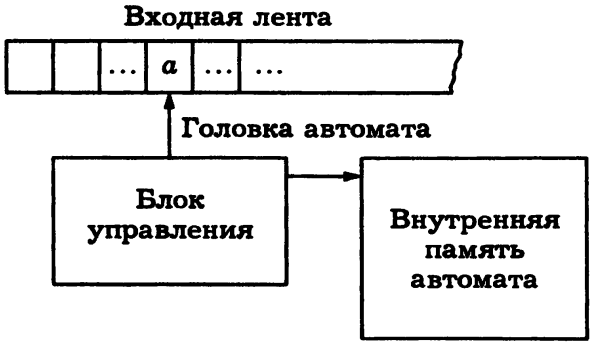


Рисунок 7.1 – Структура автомата

Автомат состоит из трех частей – ***входной ленты, управляющего устройства с конечной памятью и дополнительной или рабочей памяти***.

Входную ленту можно рассматривать как линейную последовательность клеток, или ячеек, причем каждая ячейка содержит точно один входной символ из некоторого конечного входного алфавита. Левую и правую ячейки могут занимать особенные конечные маркеры, обозначим их λ ($\perp$). Пустые ячейки обозначим ε. С помощью входной ленты изображается информация, которая поступает на вход автомата.

Входная головка в каждый момент читает одну входную ячейку. За один шаг работы автомата входная головка может переместиться только на одну ячейку влево, остаться неподвижной или переместиться на одну

ячейку вправо. Автомат, который никогда не перемещает свою входную головку влево, называется ***односторонним***. Схема входной ленты и входной головки изображена на рис. 7.2.

Рисунок 7.2 – Входная лента и входная головка

Входная головка может быть только читающей, если во время, работы автомат не образует строку – результат. Существуют такие автоматы, входная головка которых читающая и пишущая. Такие автоматы могут записывать результирующую строку или изменять символы на входной ленте.

Характерной особенностью конечного автомата является наличие памяти, которая определяет состояние автомата в зависимости от времени. ***Память автомата*** – это структура, в которой записываются, сохраняются и считываются дополнительные данные, которые используются автоматом при работе. Для каждого вида автоматов строго определенного тип памяти, функции доступа к памяти и превращению памяти. Автомат может не иметь память. Тип памяти часто определяет название автомата.

Работа автомата состоит из последовательности ***тактов***. Такт работы автомата состоит в том, что в зависимости от содержимого обозреваемой ячейки, состояния *S*, а также содержимого внутренней памяти автомат сдвигает головку вправо или влево на одну ячейку либо оставляет ее на прежнем месте, его блок управления переходит в некоторое новое состояние (возможно, что это состояние совпадет с исходным состоянием S, а содержимое внутренней памяти определенным образом модифицируется. Текущий входной символ и информация, которая вытянута из памяти, вместе с текущим состоянием управляющего устройства определяют, каким должен быть этот такт.

Таким образом, на каждом такте определяется текущая ***конфигурация*** автомата, в которую входят:

- текущее состояние управляющего устройства;
- текущее содержание входной ленты;
- текущее содержание рабочей памяти.

Конфигурация автомата изменяется на каждом такте под воздействием управляющего устройства.

Управляющее устройство состоит из конечного множества состояний $S = \{s_0, \ldots, s_n\}$ и отображений, которые в зависимости от предыдущей конфигурации позволяют определить новую конфигурацию автомата, направление перемещения входной головки (если она не односторонняя),

информацию на печати (если в автомате предусмотрена функция печати), информацию, которую заносят в память и извлекают из памяти (если автомат имеет рабочую память).

Автомат начинает работу с ***начальной конфигурации***. При начальной конфигурации управляющее устройство находится в заданном начальном состоянии, входная головка осматривает левый символ на входной ленте и память имеет предварительно установленное начальное содержание. Конфигурация, при которой автомат прекращает работу, называется ***заключительной.*** Автомат прекращает работу, перейдя в одно из состояний предварительно выделенного множества заключительных состояний или тогда, когда пересмотр входных данных завершен. Условия прекращения работы четко определяются для каждого конкретного автомата.

Управляющее устройство является ***недетерминированным,*** если для каждой конфигурации существует конечное множество возможных конфигураций (больше одной), такую, что в любую из них автомат может перейти на следующем шаге. Управляющее устройство называется ***детерминированным,*** если для каждой конфигурации существует не больше одной возможной следующей конфигурации. Соответственно автомат называется ***детерминированным*** и ***недетерминированным*** в зависимости от ***вида управляющего устройства.***

При рассмотрении грамматик было определено два способа задания языков: распознаванием строк и порождением строк. Для грамматик в основном применяется второй способ. Автоматы, которые используются для определения языка, чаще являются теми, которые распознают, их называют ***распознавателями.*** С помощью распознавателей, также как и с помощью грамматик, можно однозначно определить язык. Обычно допускается, что входная головка распознавателя только читает.

Рассмотрим некоторую строку ω. Распознаватель допускает входную строку ω, если, начиная с начальной конфигурации, в которой строка ω записана на входной ленте, распознаватель может сделать последовательность шагов, которая заканчивается завершающей конфигурацией. Следует указать, что, начиная с данной начальной конфигурации, недетерминированный распознаватель может выполнить много разных последовательностей шагов. Если хотя бы одна из этих последовательностей заканчивается завершающей конфигурацией, то начальная входная цепочка будет допущена.

Язык, который определен распознавателем**,** изображает множество входных цепочек, которые она допускает. Для каждого класса грамматик из иерархии Хомского существует класс распознавателей, что определяет тот же класс языков. Точнее языки из иерархии Хомского можно охарактеризовать так:

1. Язык регулярный (праволинейный или леволинейный) тогда и только тогда, когда он определяется конечным автоматом;

2. Язык контекстно-свободный тогда и только тогда, когда он определяется автоматом с магазинной памятью;

3. Языкконтекстно-зависимый тогда и только тогда, когда он определяется линейно ограниченным автоматом;

4. Язык определяется грамматикой общего вида тогда и только тогда, когда он определяется машиной Тьюринга.

7.2 Конечные автоматы

Как рассматривалось выше, конечные автоматы – это анализирующие модели для регулярных языков. Конечный автомат состоит только из входной ленты и управляющего устройства с конечной памятью (рис. 7.3). Конечный автомат не имеет внутренней памяти, головка движется по ленте только вправо – на одну ячейку за такт и, соответственно, является односторонней. С ленты можно только читать. Кроме того, автомат может переходить «спонтанно» из одного состояния в другое, не читая ленту и не продвигая головку. Такой такт можно рассматривать как переход из состояния в состояние по пустой цепочке. Его называют ε-тактом. Число его состояний конечно, что и определяет его название. Управляющее устройство может быть детерминированным или недетерминированным.

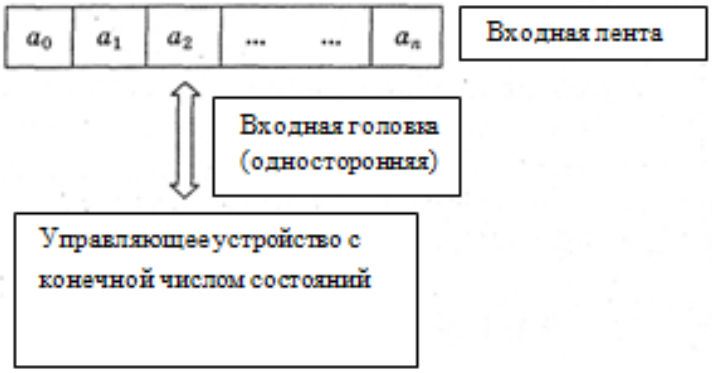


Рисунок 7.3 – Схема конечного автомата

Определим конечный автомат, задав конечное множество его состояний, допустимые входные символы, допустимые выходные символы, начальное состояние и множество завершающих состояний. Задаются также функция переходов, которая по данному текущему состоянию и текущему входному символу указывает все возможные последующие состояния, и функция печати.

Конечный автомат – это структура вида

$$M = \{S, I, O, f, g, s_0, F\}, \qquad (7.1)$$

где S – конечное множество состояний;

I – конечное множество допустимых входных символов;

O – конечное множество допустимых выходных символов;

f – отображение множества S x I во множестве S, которое определяет выбор следующего состояния (функцию f называют функцией переходов);

g – отображение множества S x I во множестве O, которое определяет выходной символ (функцию g называют функцией выходов);

$s_0 \in S$ – начальное состояние управляющего устройства;

$F \subseteq S$ – множество завершающих состояний.

Автомат может не иметь функцию g и множество O, если не предусмотрены выдачи выходной информации. В этом случае считается, что автомат без выхода. Если автомат содержит функцию g, тогда считается, что он с выходом. Множество F может быть не определенным.

Конечный автомат можно задать с помощью ***таблицы***. В этой таблице для каждой комбинации состояний автомата и входного символа указывается следующее состояние и выходной символ. Кроме того, конечный автомат можно задать с помощью ориентированного графа, который называется диаграммой состояний. В этом графе вершины изображают состояния автомата, дуги – переходы из одного состояния в другой. Каждая дуга обозначена входным и исходным символом, который отвечает этому переходу.

Пример 7.1. Построить конечный автомат, осуществляющий сложение двух натуральных чисел, используя их бинарное изображение.

Поясним процедуру поразрядного бинарного сложения. Пусть есть два числа, записанные с помощью n-разрядного двоичного кода: $x_n \ldots x_2 x_1$ и $y_n \ldots y_2 y_1$. Сначала складываются x_1 и y_1, в результате чего получается результат z_1 и переносc_1. После этого делается сложение x_2, y_2 и c_1, в итоге получается z_2 и перенос c_2. Эта процедура продолжается до шага n, на котором при добавлении x_n, y_n и c_{n-1} выходит z_n и c_n. Последняя сумма z_{n+1} равняется переносу c_n:

$$
\begin{array}{l}
c_n c_{n-1} \ \ldots\ c_3 c_2 c_1 \\
\quad x_{n\ldots} x_4 x_3 x_2 x_1 \\
\quad y_{n\ldots} y_4 y_3 y_2 y_1 \\
\hline
z_{n+1} z_{n\ldots} z_4 z_3 z_2 z_1
\end{array}
$$

Конечный автомат, который выполняет это сложение, можно построить, используя только два состояния.

Рассмотренный автомат $M = \{S, I, O, f, g, s_0\}$ имеет множество состояний $S = \{s_0, s_1\}$, множество входных символов $I = \{00, 01, 10, 11\}$, множество выходных символов $O=\{0, 1\}$, начальное состояние – $s_0 \in S$. Будем считать, что автомат прекращает работу, когда закончилось поступление входных данных, это значит, что сложение завершено.

Входными данными на одном такте является два бита – соответствующие биты слагаемых, потому возможные варианты входной информации для одного такта – 00, 01, 10, 11. Переходы построены

в соответствии с получаемой суммой, которая изображена выходным битом, и переносом, которое изображено состоянием (s_0 – перенос равняется 0, s_1 – перенос равен 1).

Отображение $f: S x I \rightarrow S$ (функция переходов) и $g: S x I \rightarrow O$ (функция выходов) показаны в таблице 7.1. В таблице изображено, что если биты, которые складываются – 00 и текущее состояние s_0 (переноса нет), то автомат остается в состоянии s_0 (переноса нет) и выдает результат 0. Если биты, которые складываются, – 00 и текущее состояние s_1 (перенос – 1), то автомат остается в состоянии s_0 (переноса нет) и выдает результат 1. Если, биты, которые складываются, – 10 и текущее состояние s_1 (перенос – 1), то автомат остается в состоянииs$_1$ (перенос 1) и выдает результат 0.

Таблица 7.1 – Функции переходов f и выходов g конечного автомата для сложения

Состояния S	f					g			
	I:	00	01	10	11	00	01	10	11
s_0		s_0	s_0	s_0	s_1	0	1	1	0
s_1		s_0	s_1	s_1	s_1	1	0	0	1

Диаграмма состояний, которая отвечает этому автомату, представлена на рис 7.4.

Около каждой стрелки перехода указана входная информация и выходной сигнал. К примеру, запись 01.0 около петли в состоянии s_1 значит, что если на вход автомата с состоянием s_1 поступает пара битов 01, то на выход необходимо выдать 0 и следующим состоянием будет s_1. Рассмотрим работу автомата. Если, например, текущее состояние – s_1 и на вход поступает 01, следующее состояние будет s_1, а выход – 0. Происходит сложение $0 + 1 + 1 = (10)_2$. Поэтому выходной сигнал – 0, а перенос 1 определяет следующее состояние s_1.

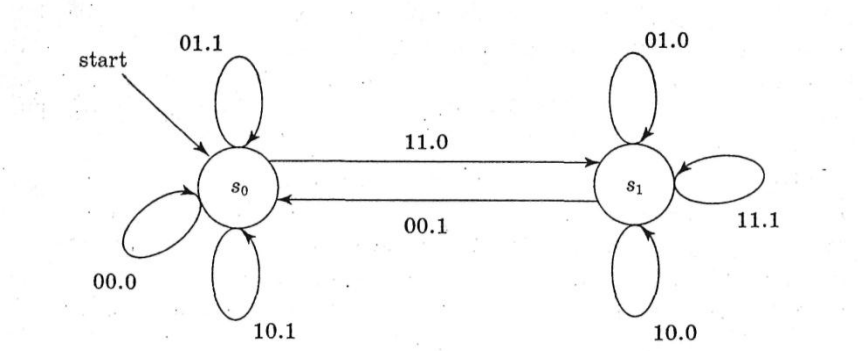


Рисунок 7.4 – Конечный автомат для сложения

В рассмотренном автомате выходная информация отвечает переходам, однако существуют автоматы, в которых выходная информация зависит только от состояния, в которое переходит автомат. В зависимости от этого различают два вида конечных автоматов.

Конечный автомат, в котором выходная информация поставлена в соответствие переходам, называется ***автоматом Мили***. Автоматы Мили – это автоматы первого рода, в которых выходной сигнал *y(t)* однозначно определяется входным сигналом *x(t)* и состоянием *q(t – 1)* автомата в предшествующий момент времени *(t – 1)*. Математической моделью таких автоматов служит система уравнений: $\begin{cases} q(t) = \delta(q(t-1),\ x(t)) \\ y(t) = \lambda(q(t-1),\ x(t)) \end{cases} t = 1, 2, \ldots$

В ***автоматах Мура*** (автоматы второго рода) выходной сигнал *y(t)* однозначно определяется входным сигналом *x(t)* и состоянием *q(t)* в данный момент времени *t*. Математической моделью таких автоматов является система: $\begin{cases} q(t) = \delta(q(t-1),\ x(t)) \\ y(t) = \lambda(q(t)) \end{cases} t = 1, 2, \ldots$

В таких автоматах функция выхода зависит только от состояний автомата в данный момент времени и не зависит от входного сигнала. Таким образом, входная строка такого автомата однократно считывается слева направо, осуществляя поочередный просмотр символов. В определенный момент времени конечный автомат находится в некотором внутреннем состоянии, которое изменяется после считывания очередного символа. Новое состояние можно охарактеризовать считанным символом и текущим состоянием.

Таким образом, в автоматах Мили функция выходов – это зависимость выходной информации от текущего состояния и текущей входной информации, а в автоматах Мура функция выходов – это зависимость выходной информации только от текущего состояния.

Конечный автомат для сложения из рассмотренного примера является автоматом Мили. Рассмотрим пример автомата Мура.

Пример 7.2. Построим автомат Мура, который на выходе печатает «1», если количество единиц, которые прочитаны автоматом во входной строке, делится на 3, и печатает «0» в противоположном случае (автомат также печатает 0, если не прочитано ни одной единицы в начальном состоянии s_0. Входной алфавит автомата $I = \{0, 1\}$. Назовем этот автомат A_3.

Определим состояния автомата: s_0 – начальное состояние, s_1 – прочитано число единиц, которое дает при делении на 3 в остатке 1, s_2 – прочитано число единиц, которое дает при делении на 3 в остатке 2, s_3 – прочитано число единиц, кратное 3. Конечно же, выходную информацию в таком автомате можно поставить в соответствие состояниям. Построим таблицу переходов и граф автомата (табл. 7.2, рис. 7.5).

Таблица 7.2 – Функции переходов f и выходов g автомата Мура A_3

I / *S*	*f*		*g*
	0	1	
s_0	s_0	s_1	0
s_1	s_1	s_2	0
s_2	s_2	s_3	0
s_3	s_3	s_1	1

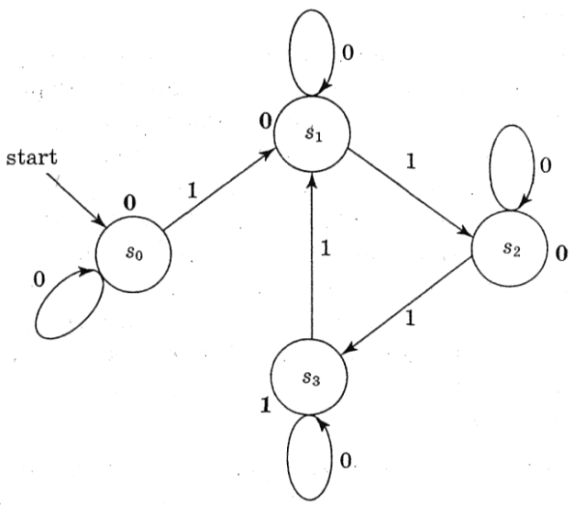


Рисунок 7.5 – Автомат Мура A_3

Пример 7.3. Допустим, на вход автомата поступила строка «0001010100». Рассмотрим, какую выходную информацию выдаст автомат. Построим соответствующую таблицу (табл. 7.3).

Таблица 7.3 – Обработка входной информации «0001010100» автоматом A_3

Прочитанная строка	***f***	***g***
0	s_0	0
00	s_0	0
000	s_0	0
0001	s_1	0
00010	s_1	0
000101	s_2	0
0001010	s_2	0
00010101	s_3	1
000101010	s_3	1
0001010100	s_3	1

После прочтения строки «0001010100» автомат печатает «1», то есть количество единиц, которые прочитал автомат во входной строке, делится на 3.

С помощью конечных автоматов можно распознавать строки языков. Автомат A_3, который построен в предыдущем примере, можно рассматривать как распознаватель языка, который состоит из строк битов, которые содержат число единиц, кратное 3. Назовем этот язык L_3. Языку L_3 принадлежат, например, строки «111», «01000110001010100», «01010101010101010» и тому подобное. Для распознавания языка обычно не нужно наличие у автомата выходных сигналов, но определяется множество конечных

состояний автомата. Если считывание входной строки автомат завершил, находясь в конечном состоянии, то строка считается распознанной. В примере множество конечных состояний состоит из единственного состояния s_3.

Язык является регулярным тогда и только тогда, когда он распознается с помощью конечного автомата.

Поскольку существует конечный автомат, который распознает язык, то, следовательно, L_3 – регулярный язык и его можно задать с помощью регулярного выражения и регулярной грамматики. Составим соответствующее регулярное выражение. В состоянии s_0 автомат находится, пропуская любое количество нулей, в начале входной строки это отвечает регулярному выражению 0^*. Состояние s_1 допускает одну единицу (при переходе из s_0 в s_1), потом может следовать любое количество нулей, автомат остается в состоянии s_1. Таким образом, состояние s_1 отвечает присоединению выражения 10^*. Аналогично переходу из s_1 в состояние s_2, из s_2 – в s_3 отвечает двукратное присоединение строки вида 10^*. Состояние s_3 является конечным. Составим выражение с помощью операции конкатенации: $0^*10^*10^*10^*$. Поскольку из состояния s_3 автомат может опять перейти в состояние s_1, прочитав единицу, и потом также последовательно пройти следующие состояния сколько угодно раз из s_1 в s_3, то в форме регулярного выражения это выглядит так: $(10^*10^*10^*)^*$. Таким образом, регулярное выражение для языка L_3 имеет вид:

L_3: $0^*10^*10^*10^*(10^*10^*10)^*$.

Можно построить грамматику для языка L_3. Это есть грамматика

$G_3 = (\{A, B, C, S\}, \{0, 1\}, P, S)$,

где P состоит из набора продукций:

1. $S \rightarrow AB$.

2. $A \rightarrow C1C1C1C$.

3. $B \rightarrow 1C1C1CB \mid \varepsilon$.

4. $C \rightarrow 0C \mid \varepsilon$.

Действительно, продукция 4 отвечает выражению 0^*, тогда продукция 3 отвечает выражению $(10^*10^*10^*)^*$, а продукция 2 – выражению 0*10*10*10*.

Один из наиболее важных результатов теории конечных автоматов состоит в том, что класс языков, определяемых недетерминированными конечными автоматами, совпадает с классом языков, определяемых детерминированными конечными автоматами.

Это означает, что для любого недетерминированного конечного автомата (НКА) всегда можно построить эквивалентный детерминированный конечный автомат (ДКА), т. е. определяющий тот же язык.

Два ***конечных автомата эквивалентны***, если эквивалентны их начальные состояния.

Алгоритм построения ДКА по НКА

Дан недетерминированный конечный автомат $M = \{S, I, f, s_0, F\}$ в результате необходимо получить детерминированный конечный автомат $M = \{S', I, f', s'_0, F'\}$, допускающий тот же язык, что и автомат М.

Шаг 1. Множество состояний S' состоит из всех подмножеств множества S. Каждое состояние из S' будем обозначать $(A_1A_2...A_n)$, где $A_i \in S$.

Шаг 2. Отображение f' определим как $f'(A_1A_2...A_n, t) = (B_1B_2...B_m)$, где для каждого $1 \le j \le m$, $f(A_i, t) = B_j$ для каких-либо $1 \le i \le n$.

Шаг 3. Начальное состояние управляющего устройства остается неизменным: $s_0 = s_0'$.

Шаг 4. Пусть $F = \{F_1, F_2, ..., F_p\}$, тогда F' – все состояния из S', имеющие вид $(...F_i...)$. $F_i \in F$ для какого-либо $1 \le i \le p$.

В множестве S' могут оказаться состояния, которые недостижимы из начального состояния, их можно исключить.

Пример 7.4. Пусть задан НКА $M = (\{H, A, B, S\}, \{0, 1\}, F, \{H\}, \{S\})$, где

$F(H, 1) = B$ $\quad$ $F(B, 0) = A$

$F(A, 1) = B$ $\quad$ $F(A, 1) = S$,

тогда соответствующий детерминированный конечный автомат будет таким:

$S' = \{H, A, B, S, HA, HB, HS, AB, AS, BS, HAB, HAS, ABS, HBS, HABS\}$

$f'(A, 1) = BS$ $\quad$ $f'(H, 1) = B$

$f'(B, 0) = A$ $\quad$ $f'(HA, 1) = BS$

$f'(HB, 1) = B$ $\quad$ $f'(HB, 0) = A$

$f'(HS, 1) = B$ $\quad$ $f'(AB, 1) = BS$

$f'(AB, 0) = A$ $\quad$ $f'(AS, 1) = BS$

$f'(BS, 0) = A$ $\quad$ $f'(HAB, 0) = A$

$f'(HAB, 1) = BS$ $\quad$ $f'(HAS, 1) = BS$

$f'(ABS, 1) = BS$ $\quad$ $f'(ABS, 0) = A$

$f'(HBS, 1) = B$ $\quad$ $f'(HBS, 0) = A$

$f'(HABS, 1) = BS$ $\quad$ $f'(HABS, 0) = A$

$F' = \{S, HS, AS, BS, HAS, ABS, HBS, HABS\}$.

Достижимыми состояниями в получившемся КА являются состояния $\{H, B, A, BS\}$, поэтому остальные состояния удаляются.

Таким образом, $M' = (\{H, B, A, BS\}, \{0, 1\}, F', H, \{BS\})$, где

$f'(A, 1) = BS$ $\quad$ $f'(H, 1) = B$

$f'(B, 0) = A$ $\quad$ $f'(BS, 0) = A$.

Для каждого конечного автомата может существовать бесконечное множество других эквивалентных ему конечных автоматов, распознающих такое же множество цепочек, однако существует единственный минимальный конечный автомат.

Минимальным конечным автоматом называется автомат, который, благодаря исключению недостижимых и эквивалентных состояний, содержит минимальное число состояний. Процесс приведения данного автомата к минимальному называется ***редукцией конечного автомата.*** Этапами редукции являются:

1) удаление недостижимых состояний.

2) замена нескольких эквивалентных состояний конечного автомата на одно.

Недостижимые состояния – это состояния, которые не достигается из начального ни для какой входной цепочки. Строки недостижимых состояний таблицы переходов можно удалить, получая эквивалентный конечный автомат.

Алгоритм поиска недостижимых состояний

Шаг 1. Найти список всех достижимых состояний:

- занести в список начальное состояние;

- для каждого состояния из списка занести в него все его состояния-преемники, которые пока отсутствуют в списке.

Шаг 2. Найти разность всего множества состояний и списка достижимых состояний.

Рассмотрим признаки эквивалентности состояний конечнх автоматов.

Состояние *s* одного конечного автомата, эквивалентно состоянию *q* другого конечного автомата, если оба эти автомата допускают одинаковые цепочки (*s* и *q* – начальные состояния соответствующего автомата).

Состояние *s* одного конечного автомата эквивалентно состоянию *q* другого конечного автомата, если для s и *q* не существует различающей цепочки, под действием которой состояние *s* переходит в допускающее состояние, а состояние *q* – в отвергающее состояние (или наоборот).

Два состояния конечного автомата эквивалентны, если выполняются следующие условия:

1) условие подобия: состояния оба допускающие или оба отвергающие;

2) условие преемственности: преемники данных состояний эквивалентны (для любых входных символов данные состояния переходят в эквивалентные состояния).

Эквивалентные состояния, принадлежащие одному конечному автомату, не нарушая эквивалентности, можно заменить одним (в таблице переходов оставить одну из строк эквивалентных состояний, удалив остальные, при этом заменить имена удаленных состояний на оставленное).

Пример 7.4. Преобразовывать НКА в эквивалентный ему ДКА (рис. 7.6).

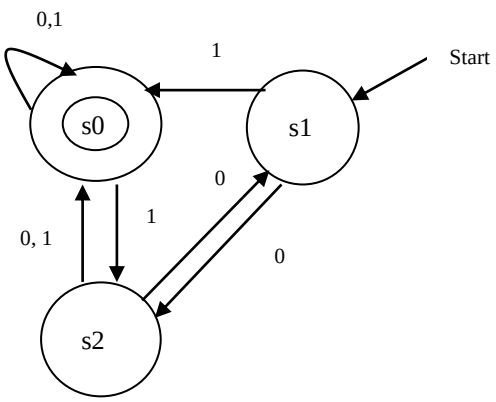


Рисунок 7.6 – Граф недетерминированного конечного автомата

Для заданного конечного автомата

$S = \{s0,s1,s2\};$

$I = \{0,1,\perp\};$

$s_0 = s1;$

$F = \{s0\}.$

Функция переходов недетерминированного конечного автомата представлена в табл. 7.4

Таблица 7.4 – Функция переходов f недетерминированного автомата

S	**0**	**1**	**⊥**
s1	*s2*	*s0*	*отверг*
s2	*s0s1*	*s0*	*отверг*
s0	*s0*	*s0s2*	*допуск*

Таблица 7.5 – Построение детерминированного конечного автомата

S	**0**	**1**	**⊥**	**S'**
s1	*s2*	*s0*	*отверг*	*a*
s2	*s0s1*	*s0*	*отверг*	*b*
s0s1	*s0s2*	*s0s2*	*допуск*	*c*
s0	*s0*	*s0s2*	*допуск*	*d*
s0s2	*s0s1*	*s0s2*	*допуск*	*e*

Таблица 7.6 – Функция переходов f' детерминированного автомата

S'	**0**	**1**	**⊥**
a	*b*	*d*	*отверг*
b	*c*	*d*	*отверг*
c	*e*	*e*	*допуск*
d	*d*	*e*	*допуск*
e	*c*	*e*	*допуск*

Таким образом, построен детерминированный конечный автомат.

Для построения минимального конечного автомата следует выделить недостижимые и эквивалентные состояния полученного ДКА. Во все состояния можно попасть из начального, следовательно, все состояния достижимы. Проверим состояния конечного автомата на эквивалентность.

По входному символу ⊥ можно выделить два подмножества:

{a, b},{c, d, e}.

По входному символу 0

{a},{b},{c, d, e}.

По входному символу 1.

Состояния *c, d, e* эквивалентны и можно оставить одно из них, например, *c*. Графическое представление минимизированного конечного автомата представлено на рис. 7.7, функция переходов *f'* принимает вид табл. 7.7.

Таблица 7.6 – Функция переходов f' минимизированного детерминированного автомата

S	***0***	***1***	⊥
a	*b*	*c*	*отверг*
b	*c*	*c*	*отверг*
c	*c*	*c*	*допуск*

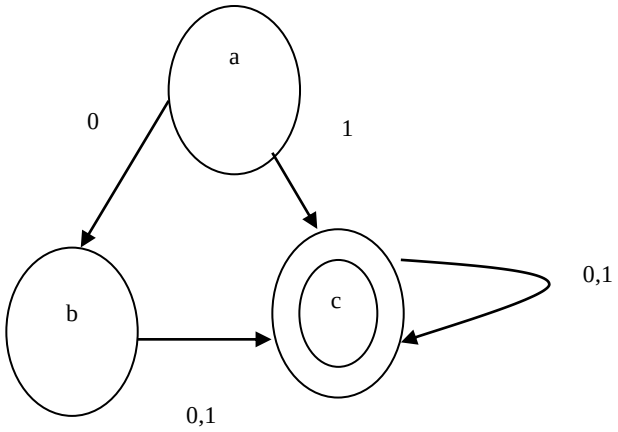


Рисунок 7.7 – Граф минимизированного конечного автомата

7.3 Автоматы с магазинной памятью

Автоматы и преобразователи с магазинной памятью играют важную роль при построении автоматно-лингвистических моделей различного назначения, связанных с использованием бесконтекстных (контекстно-свободных) языков. В частности, такие устройства используются в большинстве работающих программ для синтаксического анализа программ, написанных на различных языках программирования, которые во многих случаях можно рассматривать как бесконтекстные.

В отличие от конечных автоматов, рассмотренных ранее, автоматы и преобразователи с магазинной памятью снабжены дополнительной магазинной памятью (рабочей лентой) (рис. 7.8).

Автомат с магазинной памятью – это структура

$$M = (S, I, O, Z, f, g, s_0, z_0, F),$$

где S – конечное множество состояний;

I – конечное множественное число допустимых входных символов;

O – конечное множество допустимых выходных символов;

Z – конечное множество допустимых символов памяти;

f – функция переходов – отображение множества S x I x Z во множество S x Z, которое определяет выбор следующего состояния и символ, который заносится в память;

g – функция выходов – отображение множества S x I x Z во множество O, которое определяет выходной символ;

$s_0 \in S$ – начальное состояние управляющего устройства;

$z_0 \in Z$ – начальный символ – символ, который находится в памяти (стек) в начальный момент;

$F \subseteq S$ – множество конечных состояний.

Рисунок 7.8 – Автомат с магазинной памятью

Конечное управляющее устройство (УУ) снабжается дополнительной управляющей головкой, всегда указывающей на верхнюю ячейку магазинной памяти: за один такт работы автомата управляющая головка может произвести следующие действия:

1) стереть символ из верхней ячейки (при этом все символы, находящиеся на рабочей ленте, перемещаются на одну ячейку вверх);

2) стереть символ из верхней ячейки и записать на рабочую ленту непустую цепочку символов (при этом содержимое рабочей ленты сдвигается вниз ровно настолько, какова длина записываемой цепочки).

Таким образом, устройство магазинной памяти можно сравнить с устройством магазина боевого автомата, когда в него вкладывается патрон: те, которые уже были внутри, проталкиваются вниз; достать можно только патрон, вложенный последним.

Пример 7.5. Рассмотрим работу автомата с магазинной памятью на примере МП-распознавателя, который допускает множество $\{0^n1^n \; n = 0, 1, 2 \ldots\}$. Запись 0^n1^n означает множество строк, которые состоят из равного

количества нулей и единиц, например, «000111», «01», «0000011111». Этот язык обозначается через L_{nn} и является контекстно-свободным, но не является регулярным.

Рассмотренный автомат $M = (S, I, Z, f, s_0, z_0, F)$ имеет множество состояний $S = \{s_0,s_1,s_2\}$, множество входных символов $I = \{0,1\}$, множество символов памяти $Z = \{0,1\}$, начальное состояние s_0, конечное состояние s_0, сначала в память записан символ $x = z_0$. Будем считать, что автомат допускает входную строку $a_1, a_2, ...,a_n \in I^*$, если после прочтения этой строки (то есть последовательного обзора головкой всех символов a_i строки) автомат переходит в конечное состояние

Функция переходов f задается следующим образом:

$f (s_0, 0, x) = \{(s_1, 0x)\};$

$f (s_1, 0, 0) = \{(s_1, 00)\};$

$f (s_1,1, 0) = \{(s_2 \lambda)\};$

$f (s_2, 1, 0) = \{(s_2, \lambda)\};$

$f (s_2, \lambda, 0) = \{(s_0, \lambda)\}.$

Строка λ является пустой строкой. Переход $f (s_0, 0, x) = \{(s_1, 0x)\}$ означает, что если автомат находится в состоянии s_0, получает на вход 0 и достает из памяти текущий символ x, то он должен перейти в состояние s_0 вернуть в память символ x и поместить в стек символ 0. Переход $f (s_2, 1, 0) = \{(s_2, \lambda)\}$ значит, что если автомат находится в состоянии s_2, получает на вход символ 1 и достает из памяти текущий символ 0, то он должен остаться в состоянии s_2 и в память ничего не добавлять. Таким образом, получается, что на этом такте из стека памяти «вытолкнут» и утерян символ 0. Конечная конфигурация автомата – (s_0, λ). При работе автомат считывает в память первую половину цепочки, которая состоит из нулей, а затем удаляет из памяти по одному нулю на каждую единицу, которая поступает на вход. Кроме того, переходы состояний гарантируют, что все нули предшествуют единицам. Например, для входной цепочки 0011 автомат осуществит такую последовательность тактов: $(s_0,0011,x) \rightarrow (s_1,011,0x) \rightarrow (s_1,11,00x) \rightarrow (s_2,1,0x) \rightarrow (s_2,\lambda,x) \rightarrow (s_0, \lambda)$.

Символ «→» означает переход от такта к такту. Для каждого такта записана конфигурация, которая включает текущее состояние, часть входной цепочки, которая осталась непрочитанной, содержимое памяти. Например, для конфигурации (s_1,011,0x) текущее состояние – s_1, непрочитанная часть входной цепочки – 011, текущий входной символ (размещенный слева) – 0. Содержимое памяти – 0x, символ, которыйдостаётся (размещенный слева), – 0.

Посредством автоматов с магазинной памятью можно распознавать строки языка двумя способами. Первый – строка считается распознанной, если после его прочтения автомат переходит в завершающее состояние. Второй – если после прочтения строки стек памяти оказывается пустым.

Контрольные вопросы к разделу 7

1. Понятие автомата. Виды автоматов.
2. Как можно охарактеризовать формальные языки из иерархии Хомского исходя из теории конечных автоматов&
3. Дать определение конечного автомата. Описать его структуру.
4. Описать работу автомата Мили.
5. Описать работу автомата Мура.
6. Как преобразовать недетерминированный конечный автомат в детерминированный?
7. Какие состояния автомата называются эквивалентными и недостижимыми?
8. Алгоритм построения минимизированного конечного автомата.
9. Дать определение автомата с магазинной памятью.
10. Описать работу автомата с магазинной памятью.

8 ЛАБОРАТОРНЫЙ ПРАКТИКУМ
МОДУЛЬ 1. ТЕОРИЯ МНОЖЕСТВ, ГРАФЫ, КОМБИНАТОРНЫЙ АНАЛИЗ, БУЛЕВА ЛОГИКА

Лабораторная работа № 1 (табл. 8.1)
Элементы теории множеств

Цель работы – изучение основных элементов теории множеств.

Таблица 8.1 – Варианты заданий к лабораторной работе № 1

№ варианта	**Задание**
1	2
1	1. Фирма имеет 100 предприятий, причем каждое предприятие выпускает хотя бы одну продукцию вида A, B, C. Продукцию всех трех видов выпускают 10 предприятий, продукцию A и B – 18 предприятий, продукцию A и C – 15 предприятий, продукцию B и C – 21 предприятие. Число предприятий, выпускающих продукцию A, равно числу предприятий, выпускающих продукцию B, и равно числу предприятий, выпускающих продукцию C. Найти число всех предприятий. Составить компьютерную программу. 2. Упростить: $\overline{(A \cup B)} \cup \overline{A} \cup \overline{B}$. 3. Является ли множество A = {1, 2, 3} подмножеством множества B = {{1}, {2, 3}}? 4. Нарисовать диаграмму Эйлера – Венна для множества $\overline{(A \cup B)} \cap C$.
2	1. В группе спортсменов 30 человек, из них 20 занимаются плаванием, 18 – легкой атлетикой и 10 – лыжами. Плаванием и легкой атлетикой занимаются 11 человек, плаванием и лыжами – 8, легкой атлетикой и лыжами – 6 человек. Сколько спортсменов занимается всеми тремя видами спорта? Составить компьютерную программу. 2. Упростить: $A \cap (A \cup B)$. 3. В каком случае $A \subseteq A \cap B$? 4. Нарисовать диаграмму Эйлера – Венна для множества $\overline{A} \cup \overline{B}$.
3	1. В студенческой группе 20 человек, из них 10 имеют оценку «отлично» по английскому языку, 8 – по математике, 7 – по физике, 4 – по английскому языку и по математике, 5 – по английскому языку и по физике, 4 – по математике и по физике, 3 – по английскому языку, математике и физике. Сколько студентов в группе не имеют отличных оценок? Составить компьютерную программу. 2. Упростить: $(A \setminus B) \cup (A \setminus B)$. 3. Найти все подмножества множества A = {1, 2, 3, 4). 4. Пусть $A_n = \{0, \ 1/2^n\}$. Найти $\bigcup_{n=1}^{4} A_n$.

Продолжение таблицы 8.1

1	2
4	1. В классе 20 человек. На экзаменах по истории, математике и литературе 10 учеников не получили ни одной пятерки, 6 учеников получили 5 по истории, 5 – по математике и 4 – по литературе; 2 – по истории и математике, 2 – по истории и литературе, 1 – по математике и литературе. Сколько учеников получили 5 по всем предметам? Составить компьютерную программу. 2. Упростить: $(A \cap B) \cup (A \cap B)$. 3. Является ли множество $A = \{1, 2, 3\}$ подмножеством множества $B = \{\{1\}, \{2, 3\}\}$? 4. Нарисовать диаграмму Эйлера – Венна для множества $(A \setminus B) \cap C$.
5	1. В спортивном лагере 100 человек, занимающихся плаванием, легкой атлетикой и лыжами. Из них 10 занимаются и плаванием, и легкой атлетикой, и лыжами, 18 – плаванием и легкой атлетикой, 15 – плаванием и лыжами, 21 – легкой атлетикой и лыжами. Число спортсменов, занимающихся плаванием, равно числу спортсменов, занимающихся легкой атлетикой, и равно числу спортсменов, занимающихся лыжами. Найти это число. Составить компьютерную программу. 2. Упростить: $(A \cup B) \cup (A \cup B)$. 3. Найти все подмножества множества $A = \{1, 2, 3, 4\}$. 4. Нарисовать диаграмму Эйлера – Венна для множества $(A \setminus B) \cup C$
6	1. Группе студентов предложено три спецкурса: по мультимедиа, искусственному интеллекту и имитационному моделированию. 22 студента записались на спецкурс по мультимедиа, 18 – на спецкурс по искусственному интеллекту, 10 – на спецкурс по имитационному моделированию, 8 – на спецкурсы по мультимедиа и искусственному интеллекту, 15 – на спецкурсы по мультимедиа и имитационному моделированию, 7 – на спецкурсы по искусственному интеллекту и имитационному моделированию. 5 студентов записались на все три спецкурса. Сколько студентов в группе? Составить компьютерную программу. 2. Верно или неверно равенство: $(A \setminus B) \cup (A \cap B) = A$? 3. Привести пример множеств A, B, C, каждое из которых имеет мощность континуума, так, чтобы выполнялось равенство: $A \cup B = C$. 4. Нарисовать диаграмму Эйлера – Венна для множества $(A \setminus B) \cup (A \setminus C)$.
7	1. Во время сессии 24 студента группы должны сдать три зачета: по физике, математике и программированию. 20 студентов сдали зачет по физике, 10 – по математике, 5 – по программированию, 7 – по физике и математике, 3 – по физике и программированию, 2 – по математике и программированию. Сколько студентов сдали все три зачета? Составить компьютерную программу. 2. Упростить: $(A \cup B) \cup (A \cap B)$. 3. Доказать, что множества $A = \{(x, y): y = x^3, 1 < x < 2\}$ и $B = \{y: y = 3^x, 3 < x < \infty\}$ эквивалентны. 4. Нарисовать диаграмму Эйлера – Венна для множества $(A \setminus B) \cup C$.
8	В группе переводчиков 15 человек владеет английским языком, 19 – французским, 8 – немецким. 9 переводчиков владеют английским и французским языком, 7 – английским и немецким, 6 – французским и немецким. 4 переводчика владеют всеми тремя языками. Сколько переводчиков в группе? Составить компьютерную программу. 2. Пользуясь равносильными преобразованиями, установить, верно или неверно равенство $A \setminus (B \cup C) = (A \setminus B) \cup C$? 3. В каком случае $A \subseteq A \cap B$? 4. Нарисовать диаграмму Эйлера – Венна для множества $(\overline{A} \cup \overline{B}) \setminus (A \cup B)$.

Продолжение таблицы 8.1

1	2
9	1. Опрос группы студентов показал, что 70 % из них любят ходить в кино, 60 % – в театр, 30 % – на концерты. В кино и театр ходят 40 % студентов, в кино и на концерты – 20 %, в театр и на концерты – 10 %. Сколько студентов (в %) ходят в кино, театр и на концерты? Составить компьютерную программу. 2. Верно или неверно равенство $(A \cap B) \cap (A \cup B) = B$? 3. Являются ли множества эквивалентными $A = \{y: y = 3^x, 0<x< \infty\}$ и $B = \{y: y = 3^n, n = 1, 2, \ldots\}$? 4. Нарисовать диаграмму Эйлера – Венна для множества $A \setminus (B \cap C)$.
10	1. В группе 20 учеников. После медицинского осмотра на дополнительное обследование 14 учеников были направлены к терапевту, 6 – к окулисту, 5 – к ортопеду. К терапевту и окулисту были направлены 3 ученика, к терапевту и ортопеду –3, к окулисту и ортопеду – 2. Сколько учеников было направлено к терапевту, окулисту и ортопеду? Составить компьютерную программу. 2. Упростить $(\overline{A} \cup \overline{B}) \setminus (A \cup B)$. 3. Нарисовать диаграмму Эйлера – Венна для множества $(A \cap B) \cup (C \setminus (A \cup B))$. 4. Найти все подмножества множества $A= \{a, b, c, d\}$.
11	1. При обследовании рынка спроса инспектор указал в опросном листе следующие данные: из 1000 опрошенных 811 покупают жевательную резинку «Дирол», 752 – «Орбит» , 418 – «Стиморол», 570 – «Дирол» и «Орбит», 356 – «Дирол» и «Стиморол», 348 – «Орбит» и «Стиморол», 297 – все виды жевательной резинки. Показать, что инспектор ошибся. Составить компьютерную программу. 2. Упростить $\overline{A} \cup (B \setminus (A \cup B))$. 3. Привести пример множеств A, B, C, так, чтобы выполнялось равенство $A \cup B = C$, при этом A – конечное множество, B и C – счетные множества. 4. Нарисовать диаграмму Эйлера – Венна для множества $A \cap (B \cup C)$.
12	1. Всем участникам автопробега не повезло: 12 из них увязли в песке – пришлось толкать машину, 8 понадобилась замена колеса, у шестерых перегрелся мотор, пятеро и толкали машину, и меняли колесо, четверо толкали машину и остужали мотор, трое меняли колесо и остужали мотор. Одному пришлось испытать все виды неполадок. Сколько было участников? Составить компьютерную программу. 2. Пользуясь равносильными преобразованиями, установить, верно или неверно равенство $A \setminus (B \cap C) = (A \setminus B) \cap C$? 3. Доказать, что множество точек $A = \{y: y = 2^n, n = 1, 2, \ldots\}$ счетно. 4. Нарисовать диаграмму Эйлера – Венна для множества $(A \setminus B) \cap \square C$.
13	1. Из 10 участников ансамбля шестеро умеют играть на гитаре, пятеро – на ударных инструментах, пятеро – на духовых. Двумя инструментами владеют: гитарой и ударными – трое, ударными и духовыми – двое, гитарой и духовыми – четверо. Один человек играет на всех трех инструментах. Остальные участники ансамбля только поют. Сколько певцов в ансамбле? Составить компьютерную программу. 2. Верно или неверно равенство $\overline{(A \cup B)} \cap C) = \overline{A} \cap C \cup \overline{B} \cap C$? 3. Записать решение системы неравенств $\begin{cases} x-2>0 \\ x-5<0 \end{cases}$ в виде пересечения двух множеств. 4. Нарисовать диаграмму Эйлера – Венна для множества $\overline{A} \cap (B \cup C)$.

Продолжение таблицы 8.1

1	2
14	1. В одной студенческой группе 10 человек могут работать на Делфи, 10 – на Паскале, 6 – на Си. По два языка знают: 6 человек – Делфи и Паскаль, 4 – Паскаль и Си, 3 – Делфи и Си. Один человек знает все три языка. Сколько студентов в группе? 2. Верно или неверно соотношение $A \cap \overline{B} \cap C \subset A \cup B$? 3. Придумать пример множеств *A, B, C*, так, чтобы выполнялось равенство $A \cup B = C$, причем *A, B*, и *C* – счетные множества. 4. Нарисовать диаграмму Эйлера – Венна для множества $\overline{(A \cup B)} \cap C$.
15	1. В день авиации на аэродроме всех желающих катали на самолете, планере, дельтаплане. На самолете прокатились 30 человек, на планере – 20, на дельтаплане – 15. И на самолете, и на планере каталось 10 человек, на самолете и дельтаплане – 12, на планере и дельтаплане – 5. Два человека прокатились и на самолете, и на планере, и на дельтаплане. Сколько было желающих прокатиться? Составить компьютерную программу. 2. Верно или неверно равенство $(A \cup B) \setminus A = B \setminus A$? 3. Привести пример двух множеств *A* и *B*, таких, что мощность множества *A* больше мощности множества *B*. 4. Нарисовать диаграмму Эйлера – Венна для множества $\overline{A} \cap C \cup \overline{B} \cap C$.
16	1. Все грибники вернулись домой с полными корзинами. У десятерых из них в корзинах были белые грибы, у восемнадцати – подберезовики, у двенадцати – лисички. Белые и подберезовики были в шести корзинах, белые и лисички – в четырех, Подберезовики и лисички – в пяти. Все три вида грибов были у двух грибников. Сколько было грибников? 2. Верно или неверно равенство $(A \cup B) \setminus (A \cap B) = A \cap \overline{B} \cup \overline{A} \cap B$? 3. В каком случае $A \cup B = A \cap B$? 4. Нарисовать диаграмму Эйлера – Венна для множества $\overline{A} \cap (B \cup C)$.
17	1. Все туристы взяли в поход консервы. Шесть человек взяли «тушенку», пять – «сгущенку», восемь – кашу (с мясом). У троих в рюкзаках была «тушенка» и «сгущенка», у двоих – тушенка и каша, у троих – сгущенка и каша, и только в одном рюкзаке лежали все три вида консервов. Сколько было туристов? Составить компьютерную программу. 2. Верно или неверно равенство $\overline{(A \cup B)} \cap C = C \setminus (C \cap (A \cup B))$? 3. Пусть *A* – множество решений уравнения $x^2 - 3x + 2 = 0$. Записать это множество двумя различными способами. 4. Нарисовать диаграмму Эйлера – Венна для множества $(B \cap C) \setminus A$.
18	1. Было опрошено 70 человек. В результате опроса выяснили, что 45 человек знают английский язык, 29 – немецкий и 9 – оба языка. Сколько человек из опрошенных не знает ни английского, ни немецкого языков? 2. Верно или неверно равенство: $(A \cup B) \setminus (A \cap B) = A \cap \overline{B} \cup \overline{A} \cap B$? 3. Найти все подмножества множества $A = \{x, y, z\}$. 4. Нарисовать диаграмму Эйлера – Венна для множества $\overline{(A \cup B)} \cap C$.
19	1. В туристической группе 10 человек знают английский язык, 10 – итальянский, 6 – испанский. По два языка знают: 6 человек – английский и итальянский, 4 – английский и испанский, 3 – итальянский и испанский. Один человек знает все три языка. Сколько туристов в группе? 2. Упростить $\overline{(A \cup B)} \cap (A \cap B)$. 3. Привести пример двух множеств *A* и *B*, таких, что мощность множества *A* больше мощности множества *B*. 4. Нарисовать диаграмму Эйлера – Венна для множества $C \setminus (C \cap (A \cup B))$.

Продолжение таблицы 8.1

1	2
20	1. Предприятие объявило набор рабочих на должности токаря, слесаря и сварщика. В отдел кадров обратились 25 человек. Из них 10 человек владели профессией токаря, 15 – слесаря, 12 – сварщика. Профессией и токаря, и слесаря владели 6 человек; и токаря, и сварщика – 5 человек; и слесаря и сварщика – 3 человека. Сколько человек владеют всеми тремя профессиями? 2. Верно или неверно равенство $\overline{C} \setminus (\overline{A \cup B}) = \overline{A} \setminus (\overline{B \cup C})$? 3. Привести примеры множеств *A*, *B* и *C*, для которых одновременно выполняются равенства $A \cup B \cup C = A$ и $A \cap B \cap C = C$. 4. Нарисовать диаграмму Эйлера – Венна для множества $\overline{C} \setminus (\overline{A \cup B})$.

Для выполнения лабораторной работы № 1 необходимо изучить теоретический материал раздела 1 учебного пособия.

Пример составления компьютерной программы. В группе туристов 15 человек были раньше во Франции, 19 – в Италии, 10 – в Германии. 9 туристов были во Франции и в Италии, 7 – во Франции и в Германии, 6 – и в Италии, и в Германии. 4 туриста были во всех трех странах. Сколько туристов было хотя бы в одной из трех стран?

Программа 1

```
Program mnog;
uses crt;
 var N,A,B,C,AB,AC,BC,ABC,A1,B1,C1:integer;
begin
writeln ('Введите количество туристов, посетивших Францию');
 readln(A);
writeln ('Введите количество туристов, посетивших Италию');
 readln(B);
writeln ('Введите количество туристов, посетивших Германию');
 readln(C);
writeln ('Введите количество туристов, посетивших Францию и Италию');
 readln(AB);
writeln ('Введите количество туристов, посетивших Италию и Германию');
 readln(BC);
writeln ('Введите количество туристов, посетивших Францию и Германию');
 readln(AC);
writeln ('Введите количество туристов, посетивших и Францию, и Италию, и Германию');
 readln(ABC);
 writeln('Результат');
 N:=A+B+C–AB–AC–BC+ABC;
writeln('Общее количество туристов = ',N);
 A1:=A–AB–AC+ABC;
writeln('Количество туристов, побывавших только во Франции = ',A1);
 B1:=B–AB–BC+ABC;
writeln('Количество туристов, побывавших только в Италии = ',B1);
 C1:=C–BC–AC+ABC;
writeln('Количество туристов, побывавших только в Германии = ',C1);
 end.
```

Лабораторная работа № 2
Элементы теории графов

Цель работы – получение навыков исследования графов.

Задание (табл. 8.2)

1. Описать граф, заданный матрицей смежности, используя как можно больше характеристик. Составить матрицу инцидентности и связности (сильной связности).

2. Найти минимальный путь из x_1 в x_7 в ориентированном графе, заданном матрицей весов, используя алгоритм Форда – Беллмана. Составить программу.

3. Найти минимальное остовное дерево для графа, заданного матрицей длин ребер, используя алгоритм Краскала. Составить программу.

Таблица 8.2 – Варианты заданий к лабораторной работе № 2

№ варианта	Матрица смежности	Матрица весов	Матрица длин ребер
1	**2**	**3**	**4**
1	$\begin{pmatrix}0&1&1&0&1&1\\1&0&0&1&0&0\\1&0&0&0&1&0\\0&1&0&0&1&0\\1&0&1&1&0&1\\1&0&0&0&1&0\end{pmatrix}$	$\begin{pmatrix}\infty&4&6&12&\infty&\infty&\infty\\\infty&\infty&\infty&13&7&\infty&\infty\\\infty&\infty&\infty&5&\infty&3&\infty\\\infty&\infty&\infty&\infty&10&9&\infty\\\infty&\infty&\infty&\infty&\infty&\infty&8\\\infty&\infty&\infty&\infty&\infty&\infty&11\\\infty&\infty&\infty&\infty&\infty&\infty&\infty\end{pmatrix}$	$\begin{pmatrix}\infty&12&6&20&14\\12&\infty&2&4&6\\6&2&\infty&10&12\\20&4&10&\infty&6\\14&6&12&6&\infty\end{pmatrix}$
2	$\begin{pmatrix}0&0&0&0&0&1\\0&0&1&1&1&0\\0&0&0&0&0&0\\1&0&0&0&0&1\\1&0&1&0&0&0\\1&0&1&0&0&0\end{pmatrix}$	$\begin{pmatrix}\infty&1&3&9&\infty&\infty&\infty\\\infty&\infty&\infty&10&4&\infty&\infty\\\infty&\infty&\infty&2&\infty&1&\infty\\\infty&\infty&\infty&\infty&7&6&\infty\\\infty&\infty&\infty&\infty&\infty&\infty&5\\\infty&\infty&\infty&\infty&\infty&\infty&8\\\infty&\infty&\infty&\infty&\infty&\infty&\infty\end{pmatrix}$	$\begin{pmatrix}\infty&1&\infty&4&5\\1&\infty&2&\infty&1\\\infty&2&\infty&1&1\\4&\infty&1&\infty&3\\5&1&1&3&\infty\end{pmatrix}$
3	$\begin{pmatrix}0&1&0&1&0&0\\1&0&0&1&0&0\\0&0&0&0&1&1\\1&1&0&0&1&1\\0&0&1&1&0&0\end{pmatrix}$	$\begin{pmatrix}\infty&3&5&11&\infty&\infty&\infty\\\infty&\infty&\infty&12&6&\infty&\infty\\\infty&\infty&\infty&3&\infty&2&\infty\\\infty&\infty&\infty&\infty&9&8&\infty\\\infty&\infty&\infty&\infty&\infty&\infty&7\\\infty&\infty&\infty&\infty&\infty&\infty&10\\\infty&\infty&\infty&\infty&\infty&\infty&\infty\end{pmatrix}$	$\begin{pmatrix}\infty&6&3&10&7\\6&\infty&1&2&3\\3&1&\infty&5&6\\10&2&5&\infty&3\\7&3&6&3&\infty\end{pmatrix}$

Продолжение таблицы 8.2

1	**2**	**3**	**4**
4	$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} \infty & \infty & 5 & 4 & 2 & 2 & 9 \\ \infty & \infty & 1 & 1 & \infty & 1 & 1 \\ 2 & \infty & \infty & 1 & 1 & \infty & 3 \\ \infty & 2 & 1 & \infty & 1 & \infty & \infty \\ \infty & \infty & 2 & 2 & \infty & 1 & 6 \\ 1 & 5 & \infty & 1 & 1 & \infty & \infty \\ 2 & \infty & 1 & \infty & 1 & 2 & \infty \end{pmatrix}$	$\begin{pmatrix} \infty & 7 & 2 & 11 & 7 \\ 7 & \infty & 3 & \infty & 4 \\ 2 & 3 & \infty & 1 & 5 \\ 11 & \infty & 1 & \infty & 3 \\ 7 & 4 & 5 & 3 & \infty \end{pmatrix}$
5	$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} \infty & 3 & \infty & 2 & 2 & \infty & \infty \\ \infty & \infty & 2 & \infty & \infty & 2 & \infty \\ \infty & 4 & \infty & \infty & 3 & 1 & \infty \\ 3 & \infty & 2 & 1 & \infty & \infty & 4 \\ 1 & 1 & \infty & \infty & \infty & \infty & 1 \\ \infty & 3 & 1 & \infty & 1 & \infty & \infty \\ \infty & \infty & 2 & \infty & \infty & 1 & 5 \end{pmatrix}$	$\begin{pmatrix} \infty & 2 & \infty & 5 & 5 \\ 2 & \infty & 8 & \infty & 7 \\ \infty & 8 & \infty & 10 & 1 \\ 5 & \infty & 10 & \infty & 13 \\ 5 & 7 & 1 & 13 & \infty \end{pmatrix}$
6	$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} \infty & \infty & 9 & \infty & \infty & 2 & 12 \\ 1 & \infty & \infty & \infty & 1 & 2 & 4 \\ 2 & 1 & \infty & \infty & 1 & \infty & 2 \\ \infty & 1 & 1 & \infty & \infty & 1 & \infty \\ 1 & 2 & \infty & 2 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 1 & \infty & 8 \\ \infty & 2 & 1 & \infty & 1 & 2 & \infty \end{pmatrix}$	$\begin{pmatrix} \infty & 1 & 5 & 4 & 5 \\ 1 & \infty & 2 & 6 & 1 \\ 5 & 2 & \infty & 1 & 7 \\ 4 & 6 & 1 & \infty & 4 \\ 5 & 1 & 7 & 4 & \infty \end{pmatrix}$
7	$\begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} \infty & 3 & 4 & 9 & \infty & \infty & \infty \\ 12 & \infty & \infty & 10 & 4 & \infty & \infty \\ \infty & \infty & \infty & 2 & \infty & 1 & \infty \\ \infty & \infty & \infty & \infty & 7 & 6 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 5 \\ \infty & \infty & \infty & \infty & \infty & \infty & 8 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty \end{pmatrix}$	$\begin{pmatrix} \infty & 4 & 3 & 5 & 6 \\ 4 & \infty & 2 & \infty & 1 \\ 3 & 2 & \infty & 1 & 1 \\ 5 & \infty & 1 & \infty & 3 \\ 6 & 1 & 1 & 3 & \infty \end{pmatrix}$
8	$\begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} \infty & 2 & 5 & 8 & 9 & \infty & \infty \\ \infty & \infty & \infty & 10 & 4 & \infty & \infty \\ 5 & 3 & \infty & 2 & 1 & \infty & \infty \\ \infty & \infty & \infty & \infty & 7 & 6 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 5 \\ \infty & \infty & \infty & \infty & \infty & \infty & 9 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty \end{pmatrix}$	$\begin{pmatrix} \infty & 1 & 3 & 4 & 5 \\ 1 & \infty & 2 & 9 & 1 \\ 3 & 2 & \infty & 1 & 1 \\ 4 & 9 & 1 & \infty & 3 \\ 5 & 1 & 1 & 3 & \infty \end{pmatrix}$

Продолжение таблицы 8.2

1	2	3	4
9	$\begin{pmatrix}0&1&0&1&1&1\\1&0&0&1&0&0\\0&0&0&1&1&0\\1&1&1&0&1&0\\1&0&1&1&0&1\\1&0&0&0&1&0\end{pmatrix}$	$\begin{pmatrix}\infty&2&5&14&\infty&\infty&\infty\\11&\infty&\infty&12&6&\infty&\infty\\\infty&\infty&\infty&3&\infty&2&\infty\\\infty&\infty&\infty&\infty&9&8&\infty\\\infty&\infty&\infty&\infty&\infty&\infty&7\\\infty&\infty&\infty&\infty&\infty&\infty&10\\\infty&\infty&\infty&\infty&\infty&\infty&\infty\end{pmatrix}$	$\begin{pmatrix}\infty&5&3&10&7\\5&\infty&1&2&4\\3&1&\infty&5&6\\10&2&5&\infty&3\\7&4&6&3&\infty\end{pmatrix}$
10	$\begin{pmatrix}0&1&1&0&1&1\\1&0&0&1&1&1\\1&0&0&0&1&0\\0&1&0&0&0&1\\1&1&1&0&0&1\\1&1&0&1&1&0\end{pmatrix}$	$\begin{pmatrix}\infty&\infty&5&4&2&3&9\\\infty&\infty&1&1&\infty&1&6\\4&\infty&\infty&1&1&\infty&3\\\infty&2&1&\infty&1&\infty&\infty\\\infty&\infty&2&2&\infty&1&6\\1&5&\infty&1&1&\infty&\infty\\2&\infty&1&\infty&1&2&\infty\end{pmatrix}$	$\begin{pmatrix}\infty&7&2&11&7\\7&\infty&3&\infty&4\\2&3&\infty&1&5\\11&\infty&1&\infty&3\\7&4&5&3&\infty\end{pmatrix}$
11	$\begin{pmatrix}0&0&1&0&1&0\\0&0&0&1&0&1\\1&0&0&0&1&0\\0&1&0&0&0&1\\1&0&1&0&0&0\\0&1&0&1&0&0\end{pmatrix}$	$\begin{pmatrix}\infty&4&9&\infty&3&1&\infty\\3&\infty&2&1&\infty&\infty&4\\1&1&\infty&\infty&10&\infty&1\\\infty&3&1&\infty&1&\infty&\infty\\\infty&\infty&1&\infty&\infty&1&5\\\infty&3&\infty&1&2&\infty&\infty\\\infty&\infty&2&\infty&\infty&2&\infty\end{pmatrix}$	$\begin{pmatrix}\infty&1&\infty&4&5\\1&\infty&8&\infty&7\\\infty&8&\infty&10&1\\4&\infty&10&\infty&13\\5&7&1&13&\infty\end{pmatrix}$
12	$\begin{pmatrix}0&0&1&0&1&0\\0&0&0&1&0&1\\1&1&0&0&1&1\\0&0&0&0&0&0\\1&1&1&0&0&0\\0&1&0&1&0&0\end{pmatrix}$	$\begin{pmatrix}\infty&9&\infty&\infty&10&2&12\\1&\infty&\infty&\infty&1&2&4\\2&1&\infty&\infty&1&\infty&2\\\infty&1&1&\infty&\infty&1&15\\1&2&\infty&2&\infty&\infty&\infty\\\infty&\infty&\infty&\infty&1&\infty&8\\\infty&2&1&\infty&1&2&\infty\end{pmatrix}$	$\begin{pmatrix}\infty&1&5&4&6\\1&\infty&2&6&3\\5&2&\infty&1&7\\4&6&1&\infty&4\\6&3&7&4&\infty\end{pmatrix}$
13	$\begin{pmatrix}0&0&0&0&0&0\\1&0&0&1&0&1\\1&0&0&0&1&0\\1&1&1&0&0&0\\1&1&0&0&0&0\\0&1&0&0&1&0\end{pmatrix}$	$\begin{pmatrix}\infty&5&6&15&\infty&\infty&\infty\\\infty&\infty&\infty&13&7&\infty&\infty\\\infty&\infty&\infty&4&\infty&3&\infty\\\infty&\infty&\infty&\infty&10&9&\infty\\\infty&\infty&\infty&\infty&\infty&\infty&8\\\infty&\infty&\infty&\infty&\infty&\infty&11\\\infty&\infty&\infty&\infty&\infty&\infty&\infty\end{pmatrix}$	$\begin{pmatrix}\infty&12&6&10&4\\12&\infty&2&5&6\\6&2&\infty&10&12\\10&5&10&\infty&6\\4&6&12&6&\infty\end{pmatrix}$

Продолжение таблицы 8.2

1	**2**	**3**	**4**
14	$\begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} \infty & 2 & 3 & 9 & \infty & \infty & \infty \\ 12 & \infty & \infty & 10 & 4 & \infty & \infty \\ \infty & \infty & \infty & 2 & \infty & 1 & \infty \\ \infty & \infty & \infty & \infty & 7 & 6 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 5 \\ \infty & \infty & \infty & \infty & \infty & \infty & 8 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty \end{pmatrix}$	$\begin{pmatrix} \infty & 3 & 2 & 4 & 5 \\ 3 & \infty & 2 & \infty & 1 \\ 2 & 2 & \infty & 1 & 1 \\ 4 & \infty & 1 & \infty & 3 \\ 5 & 1 & 1 & 3 & \infty \end{pmatrix}$
15	$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} \infty & 2 & 5 & 10 & \infty & \infty & \infty \\ \infty & \infty & \infty & 12 & 6 & \infty & \infty \\ \infty & \infty & \infty & 3 & \infty & 1 & \infty \\ \infty & \infty & \infty & \infty & 9 & 8 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 7 \\ \infty & \infty & \infty & \infty & \infty & \infty & 10 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty \end{pmatrix}$	$\begin{pmatrix} \infty & 6 & 3 & 10 & 4 \\ 6 & \infty & 1 & 2 & 3 \\ 3 & 1 & \infty & 8 & 6 \\ 10 & 2 & 8 & \infty & 3 \\ 4 & 3 & 6 & 3 & \infty \end{pmatrix}$
16	$\begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} \infty & \infty & 5 & 4 & 2 & 2 & 10 \\ \infty & \infty & 2 & 1 & \infty & 2 & 1 \\ 2 & \infty & \infty & 1 & 1 & \infty & 3 \\ \infty & 2 & 1 & \infty & 1 & \infty & \infty \\ \infty & \infty & 2 & 2 & \infty & 1 & 6 \\ 1 & 5 & \infty & 1 & 1 & \infty & \infty \\ 2 & \infty & 1 & \infty & 1 & 2 & \infty \end{pmatrix}$	$\begin{pmatrix} \infty & 4 & 2 & 10 & 6 \\ 4 & \infty & 3 & \infty & 4 \\ 2 & 3 & \infty & 1 & 5 \\ 10 & \infty & 1 & \infty & 3 \\ 6 & 4 & 5 & 3 & \infty \end{pmatrix}$
17	$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} \infty & 4 & 9 & 8 & 3 & 1 & \infty \\ 3 & \infty & 2 & 1 & \infty & \infty & 4 \\ 1 & 1 & \infty & \infty & \infty & \infty & 1 \\ \infty & 3 & 1 & \infty & 1 & \infty & \infty \\ \infty & \infty & 2 & \infty & \infty & 1 & 5 \\ \infty & 3 & \infty & 2 & 2 & \infty & \infty \\ \infty & \infty & 2 & \infty & \infty & 2 & \infty \end{pmatrix}$	$\begin{pmatrix} \infty & 2 & \infty & 3 & 5 \\ 2 & \infty & 8 & \infty & 7 \\ \infty & 8 & \infty & 10 & 1 \\ 3 & \infty & 10 & \infty & 12 \\ 5 & 7 & 1 & 12 & \infty \end{pmatrix}$
18	$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} \infty & \infty & 9 & \infty & 10 & 2 & 20 \\ 1 & \infty & \infty & \infty & 1 & 2 & 4 \\ 2 & 1 & \infty & \infty & 1 & \infty & 2 \\ \infty & 1 & 1 & \infty & \infty & 1 & \infty \\ 1 & 2 & 9 & 2 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 1 & \infty & 8 \\ \infty & 2 & 1 & \infty & 1 & 2 & \infty \end{pmatrix}$	$\begin{pmatrix} \infty & 1 & 3 & 4 & 5 \\ 1 & \infty & 2 & 6 & 8 \\ 3 & 2 & \infty & 1 & 7 \\ 4 & 6 & 1 & \infty & 4 \\ 5 & 8 & 7 & 4 & \infty \end{pmatrix}$

Продолжение таблицы 8.2

1	2	3	4
19	$\begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} \infty & 3 & 5 & 12 & 20 & \infty & \infty \\ \infty & \infty & \infty & 13 & 8 & \infty & \infty \\ \infty & \infty & \infty & 5 & \infty & 3 & \infty \\ \infty & \infty & \infty & \infty & 10 & 9 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 8 \\ \infty & \infty & \infty & \infty & \infty & \infty & 11 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty \end{pmatrix}$	$\begin{pmatrix} \infty & 1 & 6 & 5 & 14 \\ 1 & \infty & 3 & 4 & 6 \\ 6 & 3 & \infty & 10 & 12 \\ 5 & 4 & 10 & \infty & 6 \\ 14 & 6 & 12 & 6 & \infty \end{pmatrix}$
20	$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} \infty & 1 & 5 & 7 & 9 & \infty & \infty \\ \infty & \infty & \infty & 10 & 4 & \infty & \infty \\ 5 & 3 & \infty & \infty & 1 & \infty & \infty \\ \infty & \infty & \infty & \infty & 7 & 6 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 4 \\ \infty & \infty & \infty & \infty & \infty & \infty & 9 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty \end{pmatrix}$	$\begin{pmatrix} \infty & 6 & 3 & 4 & 5 \\ 6 & \infty & 2 & 9 & 1 \\ 3 & 2 & \infty & 1 & 4 \\ 4 & 9 & 1 & \infty & 3 \\ 5 & 1 & 4 & 3 & \infty \end{pmatrix}$

Для выполнения лабораторной работы № 2 необходимо изучить теоретический материал раздела 2 учебного пособия.

Пример составления компьютерной программы для нахождения минимального остовного дерева, используя алгоритм Краскала, представлен в программе 2. В программе 2 приняты следующие обозначения:

- nVertex –максимальное количество вершин;
- nn – максимальная длина списка смежности;
- nRib – количество вершин.

Программа 2

```
Program tree;
uses CRT;
Const  nVertex=50;
nRib=1000; nn=100;
Type
TypeVertex=array[1..nVertex] of Integer;
TypeRib=array[1..nRib] of Integer;
TypeWeight=array[1..nVertex,1..nVertex]  of
Integer;
Var  f :Text; nX :Integer; nXo :Integer;
nUo :Integer;
X :TypeVertex;
Xo  :TypeVertex;
Uo :TypeRib;
Prev:TypeVertex ;
Dist :TypeRib;
We : TypeWeight; Wt:LongInt;
a:array[0..nn] of record
x,y,r:integer;
end;
s,r:array[1..nn] of integer;
n, m, i, p, q, ss:integer;
Procedure minDist(Var v :Integer );
Var i,d :Integer;
Begin v:=X[1]; d:=Dist[v];
for i:=2 to nX do if d>Dist[X[i]] then begin
v:=X[i]; d:=Dist[v];
        end;
        end;
   Procedure newDist(v :Integer );
   Var i :Integer;
   begin
   for i:=1 to nX do
   if Dist[X[i]]>We[X[i],v] then begin
   Dist[X[i]]:=We[X[i],v];
```

```
  Prev[X[i]]:=v;
  end; end;
Procedure Ostov;
Var i,nStop,v :Integer;
begin
nStop:=nX;
for i:=1 to nX do X[i]:=i; v:=1;
for i:=1 to nX do begin
Dist[i]:=We[i,v]; Prev[i]:=v;
end;
nXo:=0; nUo:=0; nXo:=nXo+1;
Xo[nXo]:=X[v] ;
X[v]:=X[nX]; nX:=nX-1;
Wt:=0;
while nXo<>nStop do begin
minDist(v); nXo:=nXo+1; Xo[nXo]:=v;
      for i:=1 to nX do if X[i]=v then
      begin
      X[i]:=X[nX]; break; end;
      nX:=nX-1;
      nUo:=nUo+1;
      Uo[2*nUo-1]:=Prev[v];
      Uo[2*nUo]:=v;
      Wt:=Wt+Dist[v];
      newDist(v);
      end;
      end;
      procedure sort(l,r: integer);
      var i,j: integer; x:real;
      begin
      i:=l; j:=r; x:=a[(l+r) DIV 2].r;
      repeat
      while a[i].r<x do i:=i+1;
      while x<a[j].r do j:=j-1;
      if i<=j then
      begin
      a[0]:=a[i]; a[i]:=a[j]; a[j]:=a[0];
      i:=i+1; j:=j-1;
      end;
      until i>j;
      if l<j then sort(l,j);
      if i<r then sort(i,r);
      end;
    Var  j,c ,u,uu:Integer;
    begin
    clrscr;
    ss:=0;
writeln('Введите количество вершин');
    readln(n);
writeln('Введите количество ребер');
    readln(m);
Writeln('Введите начальную и конечную вершины, вес ребер');
for i:=1 to m do readln(a[i].x, a[i].y, a[i].r);
writeln('Алгоритм Краскала');
Writeln('|-----------------|-----------------|-----|');
writeln('|Начальная вершина | Конечная вершина | Вес  |');
Writeln('|-----------------|-----------------|-----|');
    sort(1,m);
    for i:=1 to n do
    begin
    s[i]:=1;
    r[i]:=i;
    end;
    for i:=1 to m do
    begin
    p:=a[i].x;
    q:=a[i].y;
    while r[p]<>p do p:=r[p];
    while r[q]<>q do q:=r[q];
    if p<>q then  begin
writeln('|   ',a[i].x,'   |   ',a[i].y,'   |   ',a[i].r,'   |');
writeln('|-----------------|-----------------|-----|');
    ss:=ss+a[i].r;
    if s[p]<s[q] then
    begin
    r[p]:=q;
    s[q]:=s[p]+s[q];
    end
    else
    begin
    r[q]:=p;
    s[p]:=s[q]+s[p];
    end;
    end;
    end;
writeln('');
writeln('Вес остовного дерева=',ss);
writeln('');
        readln;
        end.
```

Лабораторная работа № 3 (табл. 8.3)
Комбинаторный анализ

Цель работы – изучение основных теорем и правил комбинаторики, получение практических навыков применения комбинаторных схем.

Таблица 8.3 – Варианты заданий к лабораторной работе № 3

№ варианта	Задание
1	**2**
1	В ящике 5 красных и 4 зеленых яблока. Сколькими способами можно выбрать три яблока из ящика?
2	Монету подбросили 3 раза. Сколько различных результатов подбрасываний можно ожидать?
3	Сколькими способами можно вытащить две карты пиковой масти из колоды, состоящей из 36 карт?
4	Десять человек при встрече обмениваются рукопожатиями. Сколько всего рукопожатий будет сделано?
5	Доступ к файлу открывается, только если введен правильный пароль – определенный трехзначный номер из нечетных цифр. Каково максимальное число возможных попыток угадать пароль?
6	Сколькими способами можно расположить на шахматной доске две ладьи так, чтобы одна не могла взять другую? (Одна ладья может взять другую, если она находится с ней на одной горизонтали или на одной вертикали шахматной доски).
7	Сколькими способами можно расположить на полке 10 томов энциклопедии?
8	Сколькими способами можно расположить на полке 10 томов энциклопедии так, чтобы девятый и десятый тома рядом не стояли?
9	Сколько существует шестизначных чисел, в записи которых есть хотя бы одна чётная цифра?
10	Группу из 10 человек требуется разбить на две подгруппы так, чтобы в первой группе было 6 человек, а во второй – 4 человека. Сколькими способами это можно сделать?
11	Группу из 16 человек требуется разбить на 3 подгруппы, в первой из которых должно быть 5 человек, во второй – 7 человек, в третьей – 4 человека. Сколькими способами это можно сделать?
12	Сколько существует двузначных чисел, кратных либо 2, либо 5, либо тому и другому числу одновременно?
13	Из бригады из 14 врачей ежедневно в течение 7 дней назначают двух дежурных врачей. Определить количество различных расписаний дежурства, если каждый человек дежурит один раз.
14	Сколько четырехзначных чисел, составленных из нечетных цифр, содержат цифру 3 (цифры в числах не повторяются)?
15	Шесть групп занимаются в 6 расположенных подряд аудиториях. Сколько существует вариантов расписания, при которых группы 1 и 2 находились бы в соседних аудиториях?
16	Восемь мешков постельного белья доставляются на пять этажей гостиницы. Сколькими способами можно распределить мешки по этажам? В скольких вариантах на пятый этаж доставлен один мешок? (Мешки считаем различными).

Продолжение таблицы 8.3

1	2
17	Два наборщика должны набрать 16 текстов. Сколькими способами они могут распределить эту работу между собой?
18	Поезд метро делает 16 остановок, на которых выходят пассажиры. Сколькими способами могут распределиться между этими остановками 100 пассажиров, вошедших в поезд на конечной остановке?
19	Акционерное собрание компании выбирает из 50 человек президента компании, председателя совета директоров и 10 членов совета директоров. Сколькими способами можно это сделать?
20	Из фирмы, в которой работают 10 человек, 5 сотрудников должны уехать в командировку. Сколько может быть составов этой группы, если директор фирмы, его заместитель и главный бухгалтер одновременно уезжать не должны?

Для выполнения лабораторной работы № 3 необходимо изучить теоретический материал раздела 3 учебного пособия.

Пример составления компьютерной программы. Группу из 10 человек требуется разбить на две непустые подгруппы. Сколькими способами это можно сделать? (Подгруппы считаем различными.)

Программа 3

```
Program combinator;
Uses crt;
type mas=array[1..100] of real;
Var
a:mas;
s:real;
n,i:integer;
Function rek(n:integer):integer; {Функция, реализующая факториал}
Begin
If (n<2)then rek:=1
Else rek:=n*rek(n-1);
End;
Begin       {Основная программа. Разбиение множества на группы}
ClrScr;
WriteLn('Vvedite chislo n: ');
Readln(n);
for i:=1 to n-1 do  begin
a[i]:=rek(n)/(rek(i)*rek(n-i));
WriteLn('a[',i,']: ',a[i]);  end;
s:=0;
for i:=1 to n-1 do begin
s:=s+a[i];end;
writeln('Sum = ',s:5:2);
End.
```

Лабораторная работа № 4
Исследование логических функций

Цель – изучение существующих форм представления логических функций, построение совершенных нормальных форм логических функций. Построение таблиц истинности и арифметических моделей логических функций.

Задание (табл. 8.4)

Для заданной формулы булевой функции:

1. Найти ДНФ, КНФ, СДНФ, СКНФ методом равносильных преобразований;

2. Найти СДНФ, СКНФ табличным способом (сравнить с СДНФ, СКНФ, полученными в пункте 1;

3. Разработать компьютерную программу построения таблиц истинности.

Таблица 8.4 – Варианты заданий к лабораторной работе № 4

№ варианта	Функция
1	$\overline{(x \sim y)} \to (x \vee z)$
2	$x \wedge y \to (\bar{x} \wedge z \to (x \vee y))$
3	$(y \to x) \sim (\bar{x} \to z)$
4	$\overline{(x \wedge y \to z)} \to (x \to y)$
5	$\overline{(x \to z)} \vee (y \sim z)$
6	$\overline{(x \wedge (y \to (z \sim y)))}$
7	$((x \vee y) \to \bar{z}) \vee x \wedge y \wedge z$
8	$\overline{(x \wedge (y \to (x \vee z)))}$
9	$(x \to (z \wedge (y \sim z)))$
10	$\overline{(x \sim y)} \to (x \vee z)$
11	$\overline{(x \to y)} \vee (y \to z)$
12	$(x \wedge y) \to (\overline{(x \wedge z)} \sim x)$
13	$(y \to x) \sim (x \to z)$
14	$(x \wedge \bar{y}) \to ((\bar{x} \vee z) \wedge y)$
15	$\overline{(x \wedge y)} \to ((x \vee z) \to y)$
16	$x \to (y \to (z \to y \wedge z))$
17	$((x \wedge \bar{z}) \to (x \vee \bar{z})) \to y$
18	$\overline{(x \wedge (y \to (x \sim z)))}$
19	$(x \to (x \wedge (y \vee \overline{(x \sim y)} \vee x)))$
20	$\overline{(x \to z)} \vee \overline{(y \sim z)}$

Для выполнения лабораторной работы № 4 необходимо изучить теоретический материал раздела 4 учебного пособия, а именно: алгоритмы вывода ДНФ, КНФ, СДНФ, СКНФ, используя равносильные преобразования; алгоритмы представления булевой функции, заданной таблицей, формулой в СДНФ, СКНФ.

Пример составления компьютерной программы. Построить таблицу истинности, для логических функций: $f_1 = (x_2 \wedge x_3) \vee (\bar{x}_1 \wedge \bar{x}_2) \wedge (x_3 \wedge \bar{x}_3)$, $f_2 = (x_2 \wedge x_3) \vee (\bar{x}_1 \wedge \bar{x}_2 \wedge \bar{x}_3)$

Программа 4

```
uses crt;
type  mas = array[1..8,1..5] of boolean;
var a:mas;
x1, x2, x3:boolean; i,j: byte;
a1, a2, a3: string;
begin
for i:=1 to 8 do begin
if i<=4 then a [i,1]:=false else a [i,1]:=true;
if (i<=2) or (i=5) or (i=6) then a[i,2]:=false else a [i,2]:=true;
if (i mod 2>0) then a[i,3]:=false else a[i,3]:=true; end;
for i:=1 to 8 do begin
x1:=a[i,1];
x2:=a[i,2];
x3:=a [i,3];
if (x2 and x3)or(not(x1)and not(x2)) or (x3 and not(x1)) then
a[i, 4]:=true else a[i,4]:=false;
if (x2 and x3) or (not(x1) and not(x2) and not(x3)) then a [i,5]:=true
else a [i,5]:=false;     end;
writeln('  x1       x2       x3   f1(x1,x2,x3) f2(x1,x2,x3)');
for i:=1 to 8 do begin
for j:=1 to 5 do
write(a[i,j]:7,'  ');
writeln;
end;
end.
```

Лабораторная работа № 5
Изучение методов минимизации логических функций

Цель: изучение методов минимизации логических функций для решения конкретных задач.

Задание (табл.8.5)

1. Применить метод Квайна (нечетные варианты) или метод Квайна – Мак-Класки (четные варианты) для построения сокращенной ДНФ;

2. Минимизировать функцию с помощью карт покрытия;

3. Разработать компьютерную программу, позволяющую находить минимальную ДНФ.

Таблица 8.5 – Варианты заданий к лабораторной работе № 5

№ варианта	Функция
1	$(\overline{x \sim y}) \to (x \vee z)$
2	$x \wedge y \to (\bar{x} \wedge z \to (x \vee y))$
3	$(y \to x) \sim (\bar{x} \to z)$
4	$(\overline{x \wedge y \to z}) \to (x \to y)$
5	$(\overline{x \to z}) \vee (y \sim z)$
6	$(\overline{x \wedge (y \to (z \sim y))))}$
7	$((x \vee y) \to \bar{z}) \vee x \wedge y \wedge z$
8	$(\overline{x \wedge (y \to (x \vee z)))})$
9	$(x \to (z \wedge (y \sim z)))$
10	$(\overline{x \sim y}) \to (x \vee z)$

№ варианта	Функция
11	$(\overline{x \to y}) \vee (y \to z)$
12	$(x \wedge y) \to ((\overline{x \wedge z}) \sim x)$
13	$(y \to x) \sim (x \to z)$
14	$(x \wedge \bar{y}) \to ((\bar{x} \vee z) \wedge y)$
15	$(\overline{x \wedge y}) \to ((x \vee z) \to y)$
16	$x \to (y \to (z \to y \wedge z))$
17	$((x \wedge \bar{z}) \to (x \vee \bar{z})) \to y$
18	$(\overline{x \wedge (y \to (x \sim z)))})$
19	$(x \to (x \wedge (y \vee (\overline{x \sim y}) \vee x)))$
20	$(\overline{x \to z}) \vee (\overline{y \sim z})$

Для выполнения лабораторной работы № 5 необходимо изучить теоретический материал раздела 4 учебного пособия, а именно: алгоритмы построения сокращённой ДНФ, минимальной ДНФ с использованием карты покрытия. Для реализации компьютерной программы необходимо за основу взять программу из лабораторной работы № 4.

МОДУЛЬ 2. ОСНОВЫ ТЕОРИИ КОДИРОВАНИЯ, ФОРМАЛЬНЫЕ ЯЗЫКИ И ГРАММАТИКИ, АВТОМАТЫ

Лабораторная работа № 6
Кодирование с минимальной избыточностью (алгоритмы Фано и Хаффмана)

Цель – изучение построения эффективных алгоритмов кодирования информации на основе методов Фано и Хаффмана.

Задание (табл.8.6)

1. Для заданных распределений вероятностей появления букв построить заданный код и рассчитать среднюю длину кодирования (табл. 8.6);

2. Написать программу, которая осуществляет кодирование Вашей фамилии, имени и отчества, методом Фано или Хаффмана в зависимости от варианта.

Таблица 8.6 – Варианты заданий к лабораторной работе № 6

№ варианта	Распределение вероятностей	Метод кодирования
1	P={0,1;0,6;0,09;0,08;0,07;0,06}	Метод Фано
2	P={0,5;0,3;0,1;0,03;0,02;0,01;0,04}	Метод Хаффмана
3	P={0,2;0,2;0,19;0,12;0,11;0,09;0,09}	Метод Хаффмана
4	P={0,4;0,4;0,1;0,03;0,03;0,02;0,02}	Метод Фано
5	P={0,4;0,18;0,1;0,1;0,07;0,06;0,05;0,04}	Метод Хаффмана
6	P={0,2;0,3;0,2;0,1;0,1;0,05;0,05}	Метод Фано
7	P={0,2;0,2;0,19;0,12;0,11;0,09;0,09}	Метод Фано
8	P={0,25;0,2;0,15;0,15;0,15;0,1}	Метод Хаффмана
9	P={0,4;0,18;0,1;0,1;0,07;0,06;0,05;0,04}	Метод Фано
10	P={0,6;0,1;0,06;0,08;0,07;0,09}	Метод Хаффмана
11	P={0,2;0,25;0,35;0,05;0,15}	Метод Фано
12	P={0,2;0,3;0,2;0,1;0,1;0,05;0,05}	Метод Хаффмана
13	P={0,35;0,14;0,19;0,17;0,15}	Метод Фано
14	P={0,22;0,17;0,21;0,05;0,2;0,15}	Метод Фано
15	P={0,02;0,35;0,31;0,13;0,19}	Метод Хаффмана
16	P={0,16;0,23;0,11;0,28;0,22}	Метод Хаффмана
17	P={0,38;0,08;0,27;0,18;0,09}	Метод Фано
18	P={0,14;0,36;0,19;0,24;0,07}	Метод Хаффмана
19	P={0,41;0,16;0,07;0,3;0,06}	Метод Фано
20	P={0,26;0,12;0,06;0,19;0,08;0,15;0,14}	Метод Хаффмана

Для выполнения лабораторной работы № 6 необходимо изучить теоретический материал раздела 5 учебного пособия.

Рассмотрим реализацию основных процедур алгоритмов Фано и Хаффмана в среде программирования.

Алгоритм Фано является рекурсивным алгоритмом, который строит разделимую префиксную схему алфавитного кодирования, близкого к оптимальному.

На вход в алгоритм Фано подается массив вероятностей появления букв в сообщении, упорядоченный по убыванию *p:array [1..n] of real*. Для расчета вероятности появления каждой буквы в сообщении необходимо частоту появления этой буквы разделить на общее количество букв в сообщении: *p[i]:=a[i]/kol*. В результате реализации алгоритма получается массив элементарных кодов *c:array[1..50, 1..50] of 0..1*.

Основная работа по построению элементарных кодов выполняется рекурсивной процедурой *fano(b,e,k:byte)*, где *b* – индекс начала обрабатываемой части массива *p*, *e* – индекс конца обрабатываемой части массива *p*, *k* – длина уже построенных кодов в обрабатываемой части массива *c*.

```
procedure fano(b,e,k:byte);
  var m,i:byte;
  begin
     if e>b then
      begin
          inc(k);
          m:=delen(b,e);
          for i:=b to e do
             if i>m then begin c[i,k]:=1; inc(kl[i]); end
             else begin c[i,k]:=0; inc(kl[i]); end;
          fano(b,m,k);
          fano(m+1,e,k);
      end;
  end;
```

Функция *Delen* находит середину указанной части массива p[b..e], т.е. определяет такой индекс *m(b ≤ m ≤ e)* , что сумма элементов *p[b..m]* наиболее близка к сумме элементов *p[m+1..e]*.

```
function delen(b,e:byte):byte;
  var i,m:byte;
     sb,se,d:real;
  begin
   sb:=0;
        for i:=b to e-1 do
           sb:=sb+p[i];
    se:=p[e];
    m:=e;
    repeat
        d:=sb-se;
        m:=m-1;
        sb:=sb-p[m];
        se:=se+p[m];
    until abs(sb-se)>=d;
```

```
   delen:=m;
  end;
```

Эффективность кодирования рассчитывается в процедуре *cena_kod*.

```
procedure cena_kod;
  var kod:real;
  begin
   kod:=0;
   for i:=1 to n do kod:=kod+kl[i]*p[i] ;
   writeln;
   write('cena koda: ',kod:4:3);
  end;
```

Алгоритм Хаффмана так же является рекурсивным и строит схему оптимального префиксного кодирования для заданного распределения вероятностей *p*.

```
procedure Huffmen;
 var  z,m:byte;
 begin
    z:=2;
    kod[1,1]:=0;{первый элемент}
    kod[2,1]:=1;{второй элемент}
    for i:= n-2 downto 1 do
    begin
       j:=1;
       while kod[met[i],j] <> 2  do
       begin
          kod[z+1,j] := kod[met[i],j];
          inc(j);
       end;
       for j:= met[i] to  z do
       begin
          m:=1;
          while kod[j+1,m] <> 2  do
          begin
             kod[j,m] := kod[j+1,m]; {сдвиг кода}
             inc(m); {и его длины}
          end;
       end;
       kod[j,m] := 0;
          kod[j+1,m]:=1;
       kod[j,m+1]:=2;
       inc(z);
    end;
 end;
```

Расчет эффективности кодирования происходит аналогично алгоритму Фано.

Лабораторная работа № 7
Помехоустойчивое кодирование. Код Хемминга

Цель – изучение общих методов формирования кодов, получение практических навыков по формированию корректирующих кодов на примере построения и декодирования систематического кода и изучение его свойств.

Задание (табл. 8.7)

Обнаружить и исправить ошибку в представленном коде, декодировать сообщение. Составить программу, реализующую код Хэмминга.

Таблица 8.7 – Варианты заданий к лабораторной работе № 7

№ варианта	Закодированное сообщение
1	1 1 1 0 1 0- 0 1 0 1 0 0 0 0 0-0 1 0 0 1 1 1-1 1 0 1 0 1 1 1 1- 0 0 0 0 0 1
2	1 0 1 0 0 1 1-0 0 0 1 1 1 0-1 1 1 0 1 0-1 0 0 0 0 0 1-1 1 0 0 1 0 0 0 0-1 1 0 0 1 1 0 0 1--0 1 0 1 1 0 0 0 1 0-1 1 0 1 0 1 1 1 1-0 0 1 0 0 0
3	1 1 1 1 1 1 1 0 1-1 0 0 0 1 0 1-0 1 0 0 1 0 0 1 1-1 1 1 0 0 1 1 1 1 1-0 0 1 0 0 0-0 1 1 0 1 1 1 0 0-1 1 1 0 0 1-0 1 1 0 1 0 1-0 0 0 0 0 1
4	0 1 0 0 0 1 0 1 1-0 1 1 0 0 1 1 0 1-0 1 0 1 1 1 1 1 1-1 0 1 0 0 0 1 0 1 1-0 0 1 0 0 1 0-1 1 1 0 1 0-0 0 0 0 0 1-0 1 1 1 0 1 0 0 0-0 0 0 0 1 0-1 0 1 1 0 1 1-0 1 1 0 1 1 0 1 1-1 1 1 0 0 1-1 0 1 0 0 1 1-0 0 1 0 0 0
5	1 0 1 1 1 1 0-1 0 1 0 1 0 0-0 0 1 0 1 1 1-1 0 1 1 0 0 0-1 0 1 1 1 0 0 0 0-1 1 0 0 0 0-0 0 0 1 1 1 0-1 1 1 0 0 1-0 1 0 1 0 1 1 0 0 -1 0 0 0 1 1 1-1 0 0 1 1 0 1 0 0-0 0 0 0 1 0-
6	0 1 0 1 0 1 0 0 1-0 0 0 1 0 1 1-0 0 1 0 1 1 1-0 0 1 0 0 0-1 0 0 0 0 0 1-0 0 0 0 1 0-
7	1 0 0 1 0 1 0-0 1 0 0 0 0 1-0 1 1 1 1 0 0 0 0 1-0 1 0 0 1 1 1-1 0 1 0 0 1 1 0 0 1-0 0 1 0 1 1 1-1 1 1 0 1 0-0 0 0 0 0 1
8	1 0 1 1 0 1 1-1 0 1 0 1 1 1-0 1 1 0 1 0 1-1 1 0 1 0 1 0 1 0-1 1 1 0 0 1-1 0 0 0 1 1 1-0 0 0 0 0 1
9	0 0 1 1 1 1 0 1 0 1-0 0 0 1 1 0 0 1 0 0-0 0 1 1 1 1 1-1 1 1 0 1 0-0 0 1 0 0 1 0-1 0 1 1 1 0 0 0 1 1 1-0 1 0 0 1 1 1-0 1 0 1 0 1 0 0 1
10	0 0 0 0 1 0-1 0 0 1 0 1 0-1 0 1 0 0 0 1-1 0 0 0 0 0 1-1 1 1 1 1 0 0 1 1 1-1 0 1 0 0 1 1-
11	0 0 0 1 0 1 1-0 0 1 1 1 0 0 1 1 1-1 0 0 1 0 0 0 0 0-1 1 1 0 0 1-0 0 0 1 1 0 1 1 1 0-0 0 1 0 0 0
12	0 1 1 0 1 0 1-1 0 1 1 0 0 0-0 1 1 1 1 1 0 1 1-0 0 0 1 1 1 0-0 1 1 1 1 1 1 1 0-1 1 1 1 0 1 0 1 1-0 0 1 1 1 1 1-1 0 0 1 1 1 0 0 0-0 0 1 0 0 0
13	0 1 1 0 1 1 1 0 0-0 0 1 0 0 0-1 1 1 1 0 1 1 0 0-1 0 1 0 1 1 1-0 1 0 0 0 1 0 1 1-
14	0 1 1 1 1 0 0 1 1 1 0-0 0 1 1 1 0 0 1 1 0-1 0 0 1 1 0 1 0 0-0 1 1 1 1 1 0 0 0 0-0 0 0 0 0 0 1
15	1 1 0 0 0 0-0 1 1 0 1 0 1-1 1 1 0 1 0-0 0 0 0 1 0
16	1 0 0 0 0 1 0-0 0 0 1 1 1 0-1 1 1 0 1 0-1 0 0 0 0 0 1-1 1 0 0 1 0 0 0 0-1 1 0 0 1 1 0 0 1--0 1 0 1 1 0 0 0 1 0-1 1 0 1 0 1 1 1 1-0 0 1 0 0 0
17	1 1 1 1 1 1 1 0 1-1 0 0 0 1 0 1-0 1 0 0 1 0 0 1 1-1 1 1 0 0 1 1 1 1 1-0 0 1 0 0 0-1 1 1 0 1 0- 0 1 0 1 0 0 0 0 0-0 1 0 0 1 1 1-1 1 0 1 0 1 1 1 1- 0 0 0 0 0 1
18	0 1 1 0 0 0 1 0 0-1 1 1 0 0 1-0 1 0 0 1 1 1-0 0 0 0 0 1-1 1 1 1 1 0 1 1 1 0-0 0 1 0 1 0 0-
19	0 0 0 1 1 1 0-1 1 1 0 1 0-0 0 1 0 1 1 1-1 1 0 0 1 1 1 0 0-0 0 0 0 1 0
20	1 0 1 0 0 1 1-0 0 1 0 0 0-1 0 1 1 0 1 1-1 0 1 0 1 0 0-1 0 0 1 1 0 0 0 1-0 1 1 1 1 0 1 0 0 0-0 0 0 0 1 0

Для выполнения лабораторной работы № 7 необходимо изучить теоретический материал раздела 5 учебного пособия. В табл. 8.8 содержится расшифровка кодов, полученных в результате выполнения лабораторной работы № 7.

Таблица 8.8 – Закодированные символы

Код	**Символ**	**Код**	**Символ**
000	_	100	т
0011	и	0101	о
0111	с	1010	п
1101	р	1110	е
00100	в	00101	й
01000	я	01100	н
01101	у	10110	к
11110	д	11111	а
010010	ь	010011	ы
101110	г	101111	ж
110000	м	110001	л
110010	ч	1100110	ц
1100111	з	-	-

В программе *Hemming* представлена реализация кода Хэмминга. Студенту необходимо самостоятельно разобрать пример и добавить в пример замену полученного кода на символ согласно табл. 8.8.

Программа 5

```
Program Hemming;
uses crt;
const sum1:array[1..7] of integer = (1, 3, 5, 7, 9, 11, 13);
      sum2:array[1..7] of integer = (2, 3, 6, 7, 10, 11, 14);
      sum3:array[1..5] of integer = (4, 5, 6, 7, 12);
      sum4:array[1..5] of integer = (8, 9, 10, 11, 12);
      b:array[1..4] of integer = (1,2,4,8);
type errors = record cod, nomer:integer;
end;
    kod = record code, bukv: string; end;
    mas1 = array[1..100] of errors;
    mas2 = array[1..100] of kod;
    slovo = array[1..100] of integer;
var ar1:mas1; ar2:mas2; a:slovo;
    i,j,k,n,s1,s2,s3,s4,t,t1:integer;
    st,st1,st2,st3:string; f,g:text;
    Procedure input;
var st:string; kod:integer;
begin
    assign(f,'errors.txt');
    rewrite(f);
```

```
    reset(f);
    t:=0;
    while not eof(f) do
    begin
       t:=t+1;
       readln(f,st);
       with ar1[t] do
       begin
          val((st,1,pos('',st)-1),nomer,j);
             delete(st,1,pos('',st));
             val(st,cod,j);
       end;
    end;
    close(f);
    assign(g,'kod.txt');
    rewrite(g);
    reset(g);
    t1:=0;
    while not eof(g) do
    begin
       t1:=t1+1;
       readln(g,st);
       with ar2[t1] do
       begin
          bukv:=copy(st,1,pos('',st)-1);
            delete(st,1,pos('',st));
            code:=copy(st,pos('',st)-1,15);
       end;
    end;
       close(g);
end;
begin
    input;
    clrscr;
    writeln('Chislo simvolov koda = ');
    readln(n);
    writeln('Simvol koda nomer = ');
    for  i:=1 to n do
    begin
       write(i,':');
       readln(a[i]);
    end;
    clrscr;
    writeln('Vvedennii kod: ');
    for i:=1 to n do
    write(a[i]);
    writeln;
  if n>0 then
    begin
    i:=1; s1:=0;
    while(sum1[i]<=n) do
    begin
```

```
    s1:=s1+a[sum1[i]];
    inc(i);
  end;
  if s1 mod 2=0 then s1:=0 else s1:=1;
    end;
  i:=1; s2:=0;
  while(sum2[i]<=n) do
  begin
    s2:=s2+a[sum2[i]];
    inc(i);
  end;
  if s2 mod 2=0 then s2:=0 else s2:=1;
  if n>=3 then
  begin
  i:=1; s3:=0;
  while(sum3[i]<=n) do
  begin
    s3:=s3+a[sum3[i]];
    inc(i);
  end;
  if s3 mod 2 =0 then s3:=0 else s3:=1;
  if n>=5 then
  begin
    i:=1; s4:=0;
  while(sum4[i]<=n) do
  begin
    s4:=s4+a[sum4[i]];
    inc(i);
  end;
  if s4 mod 2 =0 then s4:=0 else s4:=1;
  str(s4,st1);
  st:=st+st1;
  str(s2,st1);
  st:=st+st1;
  str(s1,st1);
  st:=st+st1;
  writeln('Kod oshibki: ',st);
  val(st,j,k);
  for i:=1 to t do
  begin
    if j=ar1[i].cod then j:=ar1[i].nomer;
  end;
if j=0 then writeln('Oshibok v kode net! ') else
  begin
    writeln('Oshibka v ',j,' simvole! ');
    if a[j]=1 then a[j]:=0 else a[j]:=1;
  end;
  writeln('Invertirovannii kod: ');
  for i:=1 to n do
  write(a[i]);
  writeln;
  for i:=1 to n do
  begin
    str(a[i],st1);
```

```
      st2:=st2+st1;
   end;
   for k:=1 to 4 do
   if b[k]<=n then st2[b[k]]:='2';
   for i:=1 to length(st2) do
   if st2[i]<>'2' then st3:=st3+st2[i];
   writeln('Iskomii kod: ',st3);
   k:=0;
   for i:=1 to t1 do
   begin
      if st3=ar2[i].code then k:=i;
   end;

   end;
End.
```

Лабораторная работа № 8
Классификация формальных грамматик

Цель **–** формирование умений и практических навыков распознавания типов формальных языков и грамматик по классификации Хомского, построение эквивалентных грамматик.

Задание (табл. 8.9)

1. Определить: к какому типу по Хомскому относится заданная грамматика, какой язык она порождает, каков тип языка.

2. Указать максимально возможный номер типа грамматики и языка.

3. Разработать программное средство, позволяющее распознать грамматику.

Таблица 8.9 – Варианты заданий к лабораторной работе № 8

№ варианта	Продукции грамматики	№ варианта	Продукции грамматики
1	***2***	***3***	***4***
1	$S \to APA$ $P \to +\mid -$ $P \to a\mid b$	6	$S \to Ab$ $A \to Aa\mid ba$
2	$S \to A\mid SA\mid SB$ $A \to a$ $B \to b$	7	$S \to 0A1\mid 01$ $0A \to 00A1$ $A \to 01$
3	$S \to 1B$ $B \to B0\mid 1$	8	$S \to AB$ $AB \to BA$ $A \to a$ $B \to b$
4	$S \to aQb\mid \varepsilon$ $Q \to cSc$	9	$S \to A\mid B$ $A \to aAb\mid 0$ $B \to aBbb\mid 1$
5	$S \to a\mid Ba$ $B \to Bb\mid b$	10	$S \to aAc$ $aA \to aaBbC\mid ab$ $Bb \to bb\mid abbbc\mid aDbbbcc$ $C \to c$ $D \to ab$

Продолжение таблицы 8.9

1	2	3	4
11	*S→0A1* *0A→0B1|0* *B1→0C11|01* *C→0D|00D1|0* *D→01*	16	*1.S→KF* *2.K→KB|CB* *3.C→CA|DA* *4.DA→aAD* *5.Aa→aA* *6.DB→bBD8* *7.Bb→bB* *8.Ab→bA* *9.DF→E* *10.BE→Eb* *11.AE→Ea* *12.bE→b*
12	*S→0A1S|ε* *A→1A|0B* *B→0S|1B*	17	*1.S→DC* *2.D→aDA|bDB|aA|bB* *3.AC→aC* *4.BC→bC* *5.Aa→aA* *6.Ba→aB* *7.Ab→bA* *8.Bb→bB* *9.C→ε*
13	*S→AB* *A→a|cA* *B→bA*	18	*S→aSL|aL* *L→Kc* *cK→Kc* *K→b*
14	*S→aBA|ε* *B→bSA* *AA→c*	19	*S→aAB* *A→Bb* *B→ε*
15	*S→Ab|c* *A→Ba* *B→cS*	20	*S→bA* *A→aB* *B→abA*

Для выполнения лабораторной работы № 8 необходимо изучить теоретический материал раздела 6 учебного пособия.

Пример составления компьютерной программы.

Программа 6

```
Program grammar;
uses crt;
const   N=['A'..'z'];
        T=['a'..'z'] ;
         l=' ';
var s1,s2:string;
f:char; i,p1,p2,p3,v:integer;
u,z:boolean;
Begin
p1:=0;p2:=0;p3:=0; v:=0;
repeat
write(' Введите левую часть правила >> ');
readln(s1);
```

```
write('Введите правую часть правила >> ');
readln(s2);
z:=false; u:=false;
if length(s1)<=length(s2) then begin
for i:=1 to length(s1) do if (s1 [i] in N) then u:=true; z:=true;
if s2[1]=l then z:=false;
if (u=true) and(z=true) then inc(p1); end;
z:=false;
u:=false;
if length(s1)=1
then if s1[1] in N then u:=true;
if (length(s2)=1) then z:=true;
if (length (s2) =0) then z:=true;
if (u=true) and(z=true) then inc(p2);
z:=false; u:=false;
if length(s1)=1
then if s1[1] in N
then u:=true;
if (length(s2)=2)  then begin
if((s2[1] in T)and(s2[2] in N))
then z:=true;
if ( (s2 [1] in N)and(s2[2] in T)) then z:=true; end;
if (length(s2)=1)  then if (s2[1]in T) then z:=true;
if (u=true) and(z=true) then inc(p3);
inc(v);
writeln('          Тип1 Тип2 Тип3 ');
writeln(s1,' ->', s2, p1:6,p2:6,p3:6);
write('Продолжить y/n: >> '); readln(f);
until(f= 'n');
if        (p1<>v) or (p1=v) and(p2<>v) or (p2=v) and(p3=v)
then     writeln ('Третий тип')
else
if  (p1<>v) or (p1=v) and(p2=v) and (p3<>v)
then writeln (' Второй тип')  else
if (p1=v) and(p2<>v) and(p3<>v) then
Writeln ('Первый тип')
else writeln('Тип 0');
End.
```

Лабораторная работа № 9
Построение конечных автоматов

Цель – изучение структуры и работы конечных автоматов, получение практических навыков построения конечного автомата и преобразования недетерминированного конечного автомата к детерминированному конечному автомату.

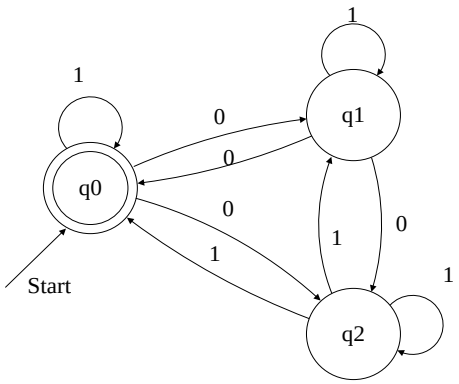
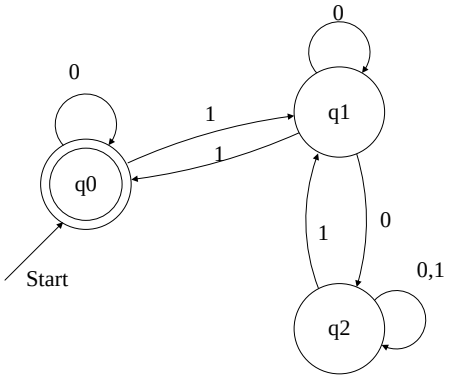
Задание (табл. 8.9)

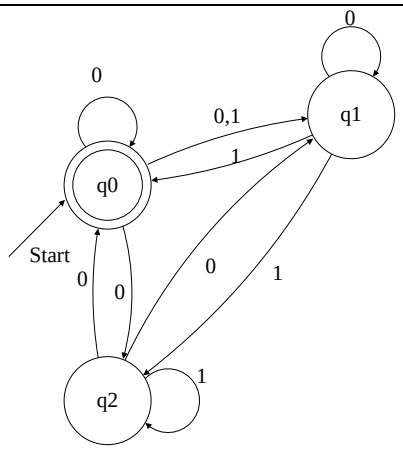
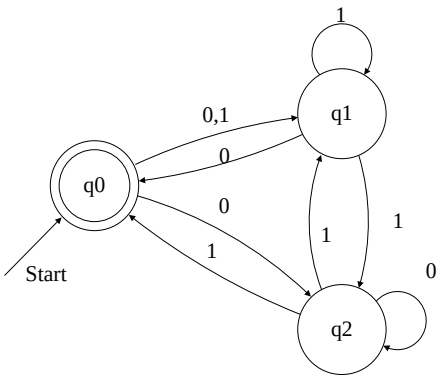
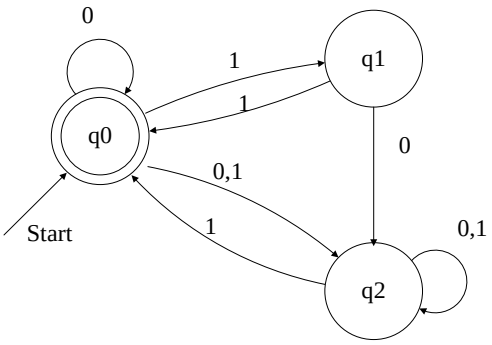
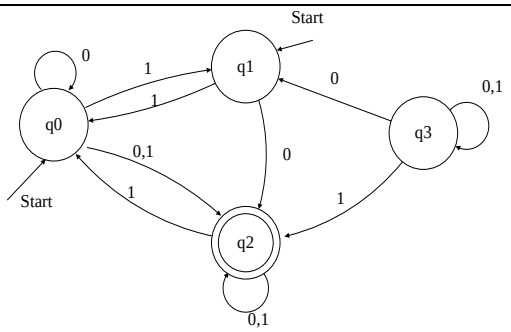
По недетерминированному конечному автомату, диаграмма которого представлена в индивидуальном задании, построить детерминированный конечный автомат, распознающий тот же язык.

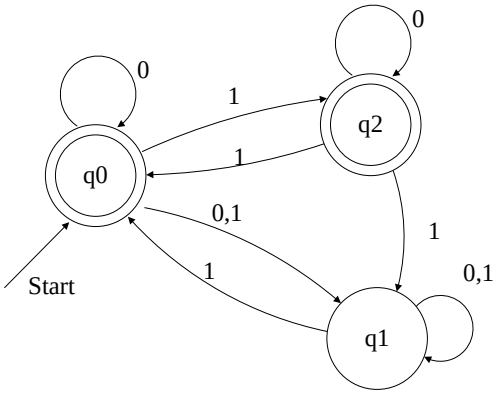
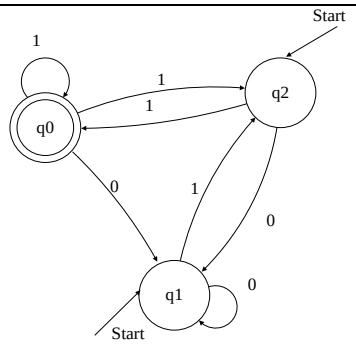
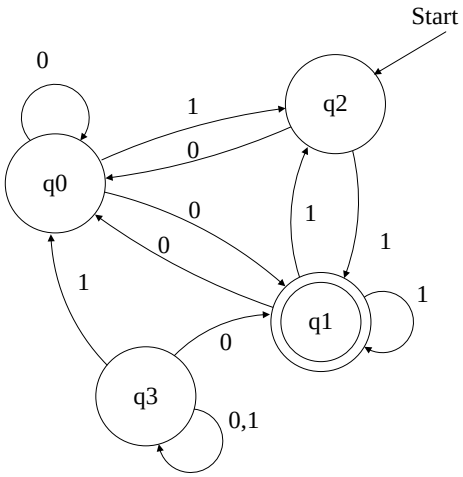
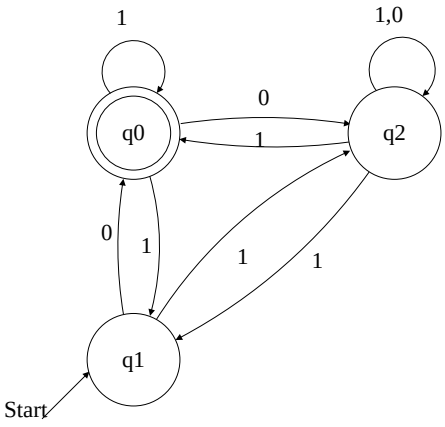
В ходе выполнения задания должны быть выполнены и описаны следующие подзадачи:

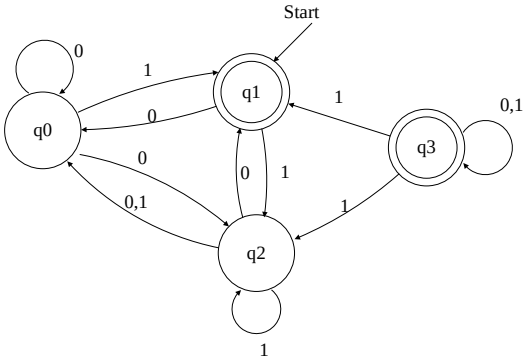
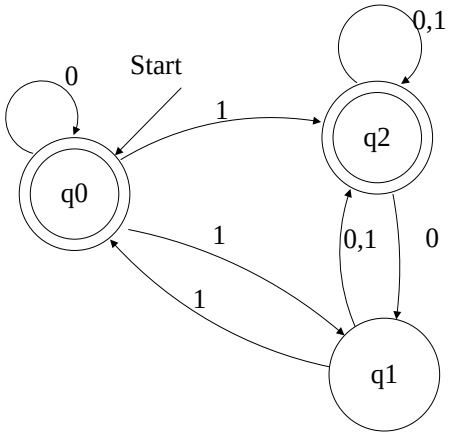
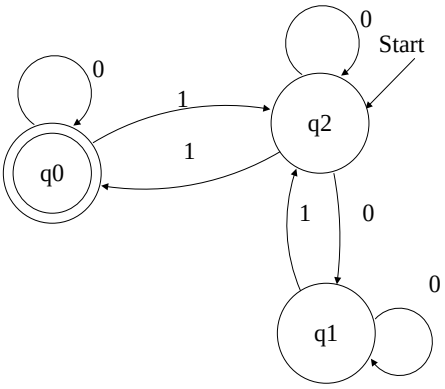
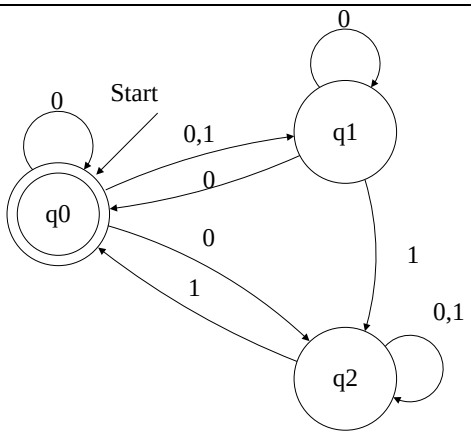
1. Записать параметры заданного НКА в виде: $M = \{S, I, O, f, g, s_0, F\};$
2. Построить таблицу переходов для заданного НКА;
4. Преобразовать НКА в ДКА;
5. Проверить состояния полученного КА на достижимость и эквивалентность;
6. Записать параметры заданного КА в виде: $M' = \{S', I', g', s_0, F'\}$;
7. Построить диаграмму состояний полученного КА;
8. Проверить, допускает ли полученный КА цепочки, которые были допущены НКА;
9. Составить программу, реализующую работу полученного КА.

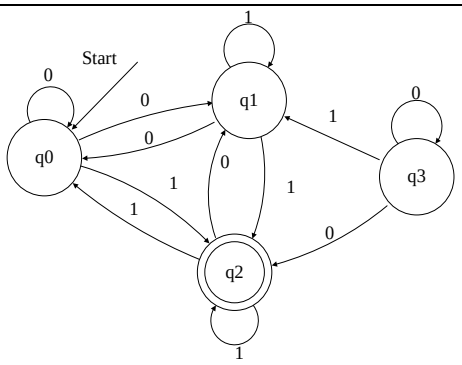
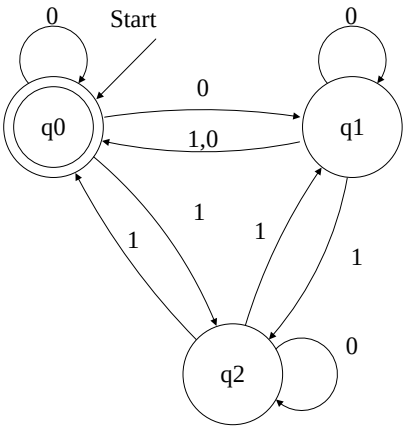
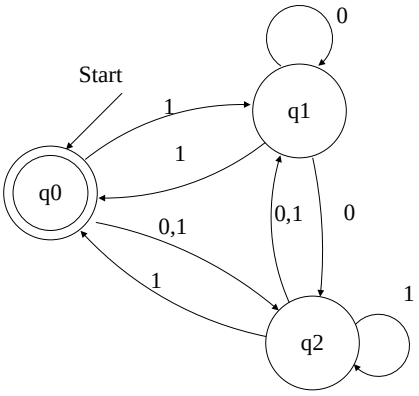
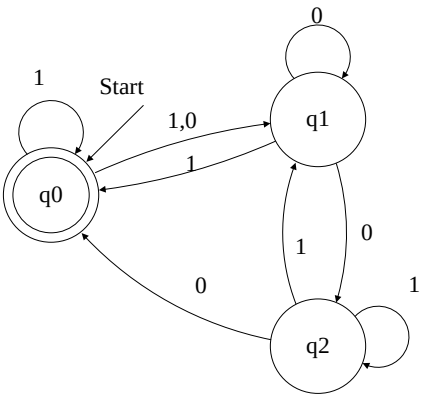
Таблица 8.10 – Варианты заданий к лабораторной работе № 9

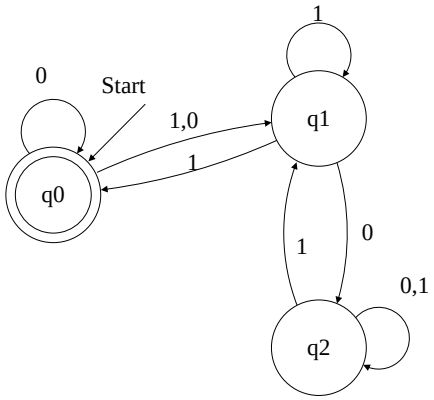
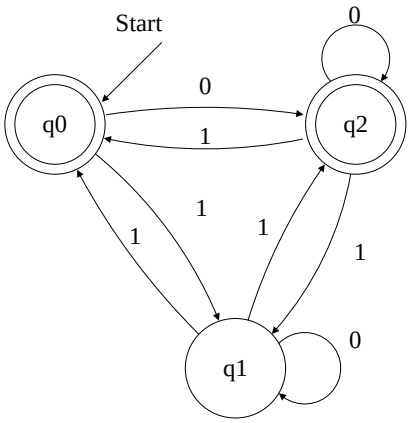
№ варианта	**Задание**
1	**2**
1	
2	

№ варианта	Задание
1	**2**
3	
4	
5	
6	

№ варианта	Задание
1	**2**
7	
8	
9	
10	

№ варианта	Задание
1	**2**
11	
12	
13	
14	

№ варианта	Задание
1	**2**
15	
16	
17	
18	

№ варианта	Задание
1	**2**
19	
20	

Для выполнения лабораторной работы № 9 необходимо изучить теоретический материал раздела 7 учебного пособия.

Для составления компьютерной программы можно использовать опережающее объявление и оператор выбора *case of*. Рассмотрим опережающее объявление. В некоторых ситуациях возникает необходимость вызвать подпрограмму, описанную далее по тексту программы. Например, эта необходимость возникает при косвенной рекурсии (подпрограмма A вызывает подпрограмму B, а та, в свою очередь, вызывает подпрограмму A). В этом случае используется опережающее объявление подпрограммы, состоящее из ее заголовка, за которым следует служебное слово *forward*. В случае построения программы для распознавания цепочки конечным автоматом состояния автомата задаются в процедурах, например для состояний *a* и *b* в блоке описания переменных укажем

```
procedure a; forward;
procedure b; forward;
```

и т. д. в зависимости от количества состояний.

В самой процедуре используется оператор выбора, например:

```
procedure a;
begin inc(i);
case s[i] of
'0': b;
'1': a;
#0: e
else k
end; end…
```

```
procedure b;
begin inc(i);
case s[i] of
'0': c;
'1': d;
#0: k
else k
end; end…
```

В процедурах *e* и *k* содержится информация: допущена или не допущена введенная цепочка.

9 МОДУЛЬНО-ТЕСТОВЫЕ ЗАДАНИЯ ДЛЯ САМОКОНТРОЛЯ

МОДУЛЬ 1. ТЕОРИЯ МНОЖЕСТВ, ГРАФЫ, КОМБИНАТОРНЫЙ АНАЛИЗ, БУЛЕВА ЛОГИКА

1. Два множества A и B равны (A = B), если:
а) мощность множества A равна мощности множества B;
б)каждый элемент A является элементом B и наоборот.

2. Если все элементы множества A входят в множество B, то A называется:
а) подмножеством множества B;
б) собственным подмножеством B.

3. Операция объединение множеств определяется как:
а) $\{x \mid x \in A$ или $x \in B\}$;
б) $\{x \mid x \in A$ и $x \in B\}$;
в) $\{x \mid x \in A$ и $x \notin B\}$;
г) $\{x \mid (x \in A$ *и* $x \notin B)$ *и* $(x \notin A$ *и* $x \in B)\}$;
д) $\{x \mid x \notin A\}$.

4. Как определяется операция разность множеств?
а) $\{x \mid x \in A$ или $x \in B\}$;
б) $\{x \mid x \in A$ и $x \in B\}$;
в) $\{x \mid x \in A$ и $x \notin B\}$;
г) $\{x \mid (x \in A$ и $x \notin B)$ *и* $(x \notin A$ *и* $x \in B)\}$;
д) $\{x \mid x \notin A\}$.

5. Как определяется операция симметрическая разность множеств?
а) $\{x \mid x \in A$ или $x \in B\}$;
б) $\{x \mid x \in A$ и $x \in B\}$;
в) $\{x \mid x \in A$ и $x \notin B\}$;
г) $\{x \mid (x \in A$ *и* $x \notin B)$ *и* $(x \notin A$ *и* $x \in B)\}$;
д) $\{x \mid x \notin A\}$.

6.Сколькими способами можно составить расписание одного учебного дня из 5 различных уроков?
а) 30;
б) 100;
в) 120;
г) 5.

7. В группе 32 студента. Сколькими способами можно сформировать команду из 4 человек для участия в математической олимпиаде?
а) 128;
б) 35960;
в) 36;
г) 46788.

8.Сколько существует различных двузначных чисел, в записи которых можно использовать цифры 1; 2; 3; 4; 5; 6, если цифры в числе должны быть различными?

а) 10;

б) 60;

в) 20;

г) 30.

9. Упростить выражение: $\frac{1}{(n+1)!} - \frac{1}{(n+2)!}$.

а) $\frac{(n+1)!}{(n+2)!}$;

б) $\frac{n+1}{(n+2)!}$;

в) $\frac{1}{(n+2)!(n+1)!}$;

г) 0.

10. Граф, имеющий как ориентированные, так и неориентированные ребра, называется:

а) мультиграфом;

б) смешанным графом;

в) нагруженным графом.

11.Сумма элементов строки матрицы смежности неориентированного графа характеризует:

а) степень вершины x_i;

б) число дуг, исходящих из x_i;

в) число дуг, входящих в x_i;

12. Прямоугольная матрица ($n \times m$), где n – число вершин, m – число ребер называется:

а) матрица смежности;

б) матрица инцидентности;

в) матрица связности.

13. Сколько существует различных булевых функций n переменных?

а) 2^n_n;

б) 2^2 ;

в) n^2;

г) $n!$.

14. Сколько существует различных наборов переменных для булевой функции n переменных?

а) 2^n_n;

б) 2^2 ;

в) n^2;

г) $n!$.

15. Верно ли утверждение: «Переменные булевой функции и сама булева функция принимают значения 0 или 1»?

а) да;

б) нет.

16. Верно ли утверждение: «Переменные булевой функции принимают значения 0 или 1, а значения самой булевой функции совпадают с множеством действительных чисел?»:

а) да;

б) нет.

17. Верно ли утверждение: «Значения переменных булевой функции совпадают с множеством действительных чисел, а сама булева функция принимает значения 0 или 1»?:

а) да;

б) нет.

18. Верно ли утверждение: Значения переменных булевой функции и значения самой функции совпадают с множеством действительных чисел;

а) да;

б) нет.

19. Сколько может быть различных ДНФ (КНФ) у булевой функции?

а) Ноль или одна;

б) ноль или бесконечно много;

в) ноль или одна;

г) одна;

д) одна или бесконечно много.

20. Сколько может быть различных СДНФ (СКНФ) у булевой функции?

а) ноль или одна;

б) ноль или бесконечно много;

в) ноль или одна;

г) одна;

д) одна или бесконечно много.

МОДУЛЬ 2. ОСНОВЫ ТЕОРИИ КОДИРОВАНИЯ, ФОРМАЛЬНЫЕ ЯЗЫКИ И ГРАММАТИКИ, АВТОМАТЫ

1. Двоичный разряд, значением которого может быть 0 или 1, называется:

а) байт;

б) бит;

в) слово;

г) цифра.

2. Осуществите перевод десятичного числа 256 в двоичное.

а) $256_{10} = 100000000_2$;

б) $256_{10} = 1000000_2$;

в) $256_{10} = 101010010_2$;

г) $256_{10} = 110100_2$.

3. Осуществите перевод десятичного числа 256 в шестнадцатеричное.

а) $256_{10} = 100_{16}$;

б) $256_{10} = 100_{16}$;

в) $256_{10} = 100_{16}$;

г) $256_{10} = 100_{16}$.

4. Результатом перевода числа 111000 из двоичного кода в шестнадцатеричный будет:

а) 36;

б)37;

в) 38.

5. Число разрядов, по которым отличаются две кодовые комбинации, называется:

а) кодовым расстоянием;

б) ошибкой;

в) синдромом ошибки.

6. При кодировании по Хэммингу в состав кодового слова входят:

а) информационные разряды и контрольные разряды;

б) информационные разряды и синдром ошибки;

в) информационные и контрольные разряды, синдром ошибки.

7. Является ли набор символов baaba словом алфавита V = {a, b, c}?

а) да;

б) нет.

8. Произвольное непустое конечное множество $V = \{a_1, \ldots, a_n\}$, элементы которого называют буквами или символами называется:

а) языком;

б) алфавитом;

в) словом (цепочкой).

9. Какой набор слов будет иметь язык L_2L_1 в результате выполнения конкатенации языков L_1 = {a, ac, cc} и L_2 = {bc, ca}?

а) L_2L_1 = {abc, aca, acbc, acca, ccbc, ccca};

б) L_2L_1 = {abc, acca, cc};

в) L_2L_1 = {bca, bcac, bccc, caa, caac, cacc};

г) L_2L_1 = {aacccbcca}.

10. Расставьте соответствия. В порождающей грамматике G (T, N, P, S):

а) T	д) начальный символ
б) N	е) терминальный алфавит
в) P	ж) нетерминальный алфавит
г) S	з) правила вывода.

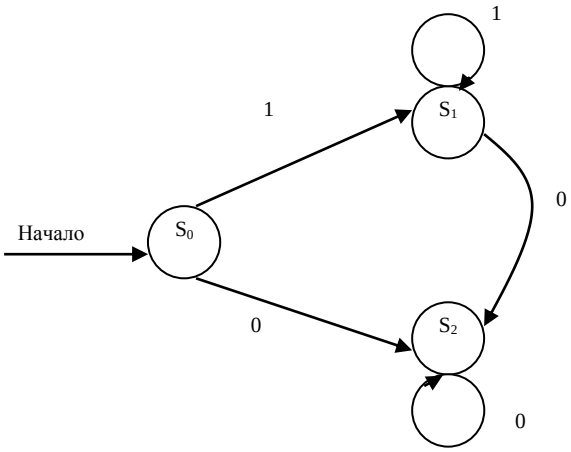
11. Листьями в дереве вывода грамматики могут быть:
а) терминальные символы;
б) нетерминальные символы;
в) символ пустой цепочки;
г) начальный символ.

12. Конечный автомат это:
а) кодер;
б) распознаватель;
в) декодер.

13. Конечный автомат К = (Q, T, δ, q_0, F) включает в себя:
а) алфавит входных символов;
б) терминальные символы;
в) нетерминальные символы;
г) начальный символ;
д) начальное состояние;
е) конечное множество состояний;
ж) множество заключительных состояний;
з) правила вывода;
и) функцию переходов.

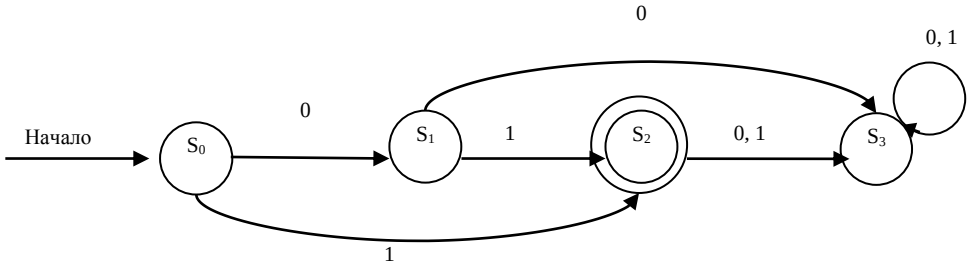
14. Наглядным способом представления автомата является:
а) диаграмма состояний;
б) дерево вывода;
в) диаграммы Эйлера.

15. Дана диаграмма состояний некоторого автомата:



Будет ли допущена цепочка 1111000 данным автоматом?
а) да;
б) нет.

16. Перечислите все цепочки, которые будут допущены конечным автоматом, заданным следующей диаграммой состояний:

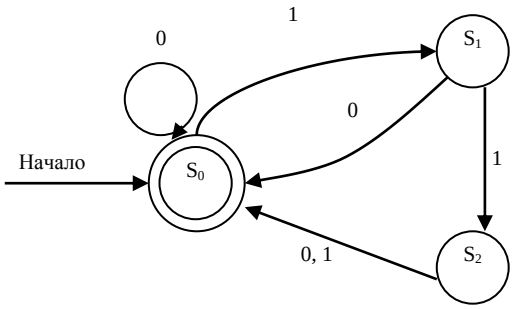


17. Детерминированным называется автомат, у которого:

а) для каждой пары «состояние – вход» существует единственное следующее состояние, заданный функцией переходов;

б) может быть несколько возможных следующих состояний для каждой пары «состояние – входной символ».

18. Автомат, диаграмма состояний которого представлена на рисунке, является:



а) детерминированным;

б) недетерминированным.

19. Язык контекстно-свободный тогда и только тогда, когда он определяется:

а) машиной Тьюринга;

б) автоматом с магазинной памятью;

в) линейно ограниченным автоматом;

г) конечным автоматом.

20. Языком автомата называется:

а) множество состояний и входных символов;

б) множества состояний, входных символов и конечных состояний;

в) множество цепочек, допускаемых данным автоматом.

ЛИТЕРАТУРА

1. **Нікольський, Ю. В.** Дискретна математика: підручник / Ю. В. Нікольський, В. В. Пасічник, Ю. М. Щербина ; за ред. М. З. Згуровського. – К. : Видавнича група BHV, 2007. – 368 с. – ISBN 966-552-201-9.

2. **Аляев, Ю. А.** Дискретная математика и математическая логика: учебник / Ю. А. Аляев, С. Ф. Тюрин. – М. : Финансы и статистика, 2006. – 368 с. – ISBN 5-279-03045-5.

3. **Бондаренко, М. Ф.** Комп'ютерна дискретна математика : підручник / М. Ф. Бондаренко, Н. В. Білоус, А. Г. Руткас. – Х. : Компанія СМІТ, 2005. – 480 с. – ISBN 966-8530-20-9.

4. **Андерсон, Д. А.** Дискретная математика и комбинаторика / Андерсон Д. А. – М. : Вильямс, 2006. – 960 с. – ISBN 5-8459-0498-6.

5. **Новиков, Ф. А.** Дискретная математика для программистов : учебник для вузов / Новиков Ф. А. – 2-е изд. – СПб. : Питер, 2008. – 364 с. – ISBN 5-272-00183-4.

6. **Белоусов, А. И.** Дискретная математика : учебник / А. И. Белоусов, С. Б. Ткачев ; под ред. В. С. Зарубина, А. П. Крищенко. – М. : Изд-во МГТУ им. Баумана, 2006. – 744 с. – ISBN 5-7038-1769-2 (5-7038-1270-4).

7. **Фомичев, В. М.** Дискретная математика и криптология : курс лекций / Фомичев В. М. ; под общ. ред. Н. Д. Подуфалова. – М. : ДИАЛОГ-МИФИ, 2005. – 400 с. – ISBN 5-86404-185-8.

8. **Сигорский, В. П.** Математический аппарат инженера : учебник для вузов / Сигорский В. П. – изд. 2-е, стереотип. – К. : Техника, 1975. – 768 с. – ISBN 000-000-000-000-0.

9. **Захаров, Н. Г.** Синтез цифровых автоматов : учебное пособие / Н. Г. Захаров, В. Н. Рогов. – Ульяновск : УлГТУ, 2008. – 135 с. – ISBN 5-89146-300-0.

10. **Гаврилов, Г.П.** Сборник задач по дискретной математике : уч. пособ. для вузов / Г. П. Гаврилов, Л. Сапоженко. – М. : Наука, 2004. – 368 с. – ISBN 5-9221-0477-2.

11. **Акимов, О. Е.** Дискретная математика: логика, группы, графы / Акимов О. Е. – 2-е изд., доп. – М. : Лаборатория Базовых знаний, 2006. – 376 с. – ISBN 5-93208-025-6.

12. **Яблонский, С. В.** Введение в дискретную математику : учебное пособие для вузов / Яблонский С. В. – 3-е изд., стер. – М. : Высш.шк., 2008. – 384 с. – ISBN 5-06-004681-8.

13. **Горбатов, В. А.** Основы дискретной математики : учеб.пособие / Горбатов, В. А. – М. : Высш.шк., 2004. – 311 с. – ISBN 5-02-015238-2.

14. **Капітонова, Ю. В.** Основи дискретної математики : підручник / Ю. В. Капітонова, С. Л. Кривий, О. А. Летичевський. – К. : Наукова думка, 2006. – 579 с. – ISBN 966-00-0622-5.

15. **Ерусалимский, Я. М.** Дискретная математика: теория чисел, задачи, приложения / Ерусалимский Я. М. – 4-е изд. – М. : Вузовская книга, 2008. – 280 с. – ISBN 5-89522-034-7.

16. **Редькин, И. П.** Дискретная математика : учебник для вузов / Редькин И. П. –СПб. : Лань, 2006. – 96 с. – ISBN 5-8114-0522-7.

17. **Бардачов Ю. М.** Дискретна математика : підручник / Ю. М. Бардачов, Н. А. Соколова, В. Є. Ходаков ; за ред. В. Є. Ходакова. – К. : Вища школа, 2006. – 287 с. – ISBN 966-642-090-2.

18. **Волкова, И. А.** Формальные грамматики и языки. Элементы теории трансляции : учебное пособие / И. А. Волкова, Т. В. Руденко. – М. : Издательский отдел факультета вычислительной математики и кибернетики МГУ им. М. В. Ломоносова, 1999. – 62 с. – ISBN 5-89407-032-5.

Навчальне видання

БЄЛЄВЦОВ Леонід Васильович,

ГУДКОВА Катерина Юріївна

ВВЕДЕННЯ У ДИСКРЕТНУ МАТЕМАТИКУ

Навчальний посібник
(*Російською мовою*)

Редагування С. П. Шнурік

Комп'ютерне верстання О. П. Ордіна

107/2012. Формат 60 х 84/16. Ум. друк. арк. __.
Обл.-вид. арк. ___. Тираж ___ пр. Зам. № __.

Видавець і виготівник
Донбаська державна машинобудівна академія
84313, м. Краматорськ, вул. Шкадінова, 72.
Свідоцтво суб'єкта видавничої справи
ДК №1633 від 24.12.2003