Министерство образования Российской Федерации

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

Е.Н. Сафьянова

ДИСКРЕТНАЯ МАТЕМАТИКА

Часть 2

Учебное пособие

2000

Сафьянова Е.Н.
Дискретная математика. Часть 2: Учебное пособие. – Томск: Томский межвузовский центр дистанционного образования, 2000. – 98 с.

Учебное пособие рассмотрено и рекомендовано к изданию методическим семинаром кафедры автоматизированных систем управления ТУСУР 17 мая 1999 г.

СОДЕРЖАНИЕ

1 ФОРМАЛЬНЫЕ ТЕОРИИ

1.1 Основные положения

Мы рассмотрели две логических системы: алгебру высказываний и логику предикатов. Это было содержательное рассмотрение. Попытаемся теперь описать их с формальной точки зрения, дав понятие формальной теории.

Построение формальной теории начинают с построения формального языка теории. А для построения формального языка нужно выделить множество знаков, составляющих его алфавит, а затем указать правила, по которым будем строить **формулы** нашей теории.

Формальная теория - это множество все формул формального языка с выделенным в нем подмножеством формул, называемых «*истинными*». Теория, в которой не все формулы истинны, называется **непротиворечивой**.

Среди формальных теорий наиболее важным является класс формальных теорий, называемых **аксиоматическими**. Их рассмотрением мы и ограничимся.

Для аксиоматических теорий характерен следующий порядок выделения «истинных» формул:

1) выделяется некоторое подмножество формул, называемых **аксиомами теории**;
2) указывается конечное множество отношений между формулами. Эти отношения называются **правилами вывода**.

Правила вывода ставят в соответствие некоторым конечным последовательностям формул – новые формулы. С помощью этих правил из аксиом получаются новые истинные формулы – теоремы.

Чтобы определить, что такое теорема, определим вначале, что такое доказательство.

Доказательством называется конечная последовательность формул $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$ такая, что каждая $\mathcal{A}_i$ есть либо аксиома теории, либо получена из предыдущих формул по одному из правил вывода.

Теоремой называется такая формула $\mathcal{A}$ теории, что существует доказательство, в котором последней формулой является формула $\mathcal{A}$.

Для аксиоматической теории введено понятие **полноты**. Аксиоматическая теория полна (в смысле Поста), если присоединение к ее аксиомам формулы, не являющейся теоремой, при сохранении неизменными правил вывода, делает теорию противоречивой.

Чтобы установить связь между формальной теорией и какой-либо конкретной содержательной теорией, нужно решить вопрос об **интерпретации** формальной теории в содержательную. Говорят, что существует интерпретация формальной теории в содержательную, если существует соответствие между формулами формальной теории и объектами – утверждениями содержательной теории. Интерпретация называется **правильной**, если каждой теореме формальной теории ставится в соответствие истинное утверждение содержательной теории. Интерпретация называется **адекватной**, если она правильная и каждому истинному утверждению содержательной теории ставится в соответствие теорема формальной теории (т.е. существует взаимно однозначное соответствие между утверждениями содержательной теории и теоремами формальной теории).

Теперь, прежде чем перейти к рассмотрению некоторых формальных логических теорий, сделаем следующее замечание. Для того чтобы ввести формальный язык, мы должны пользоваться каким-то языком, (например, русским, дополненным некоторыми символами). Этот язык мы будем называть **метаязыком**, в отличие от первого (т.е. формального языка), который называют языком-объектом.

В метаязыке доказывают некоторые утверждения формальной теории, их относят к **метатеории**. Поэтому следует различать употребление слов «доказательство» и «теорема» в формальном языке (языке-объекте) и в метаязыке.

В математике существует термин «исчисление». Он имеет два смысла.

В первом смысле исчисление – составная часть названия некоторых разделов математики, трактующих правила вычислений и оперирования с объектами того или иного типа, например, интегральное исчисление, вариационное исчисление.

Во втором смысле исчисление – дедуктивная система, т.е. способ задания того или иного множества путем указания исходных элементов (аксиом исчисления) и правил вывода, каждое из которых описывает, как строить новые элементы из исходных и уже построенных.

Построение формальных теорий основано именно на этом способе. Поэтому исчисления являются одним из основных аппаратов математической логики. В связи с этим построение формальной теории для алгебры высказываний называется **исчислением высказываний**, а для логики предикатов – **исчислением предикатов**.

1.2 Исчисление высказываний

Исходными символами или алфавитом исчисления высказываний является бесконечное число переменных высказываний

X , Y , Z , X_1 , X_2 , X_3 ... ,

четыре символа логических операций

$\wedge$, $\vee$, $\rightarrow$, $\bar{}$

и скобки (,).

Определим формулы исчисления высказываний.

1. Переменное высказывание – формула.
2. Если $\mathcal{A}$ и $\mathcal{B}$ – формулы, то $(\mathcal{A}\wedge\mathcal{B})$, $(\mathcal{A}\vee\mathcal{B})$, $(\mathcal{A}\rightarrow\mathcal{B})$, $\bar{\mathcal{A}}$ – формулы.
3. Никаких формул, кроме выделенных согласно п.п. 1 и 2 нет.

Исчисление высказываний будем базировать на бесконечном числе аксиом. Поскольку бесконечное число аксиом полностью описать нельзя, выпишем так называемые **аксиомные схемы**.

1. $(\mathcal{A}\rightarrow(\mathcal{B}\rightarrow\mathcal{A}))$.
2. $((\mathcal{A}\rightarrow(\mathcal{B}\rightarrow\mathcal{C}))\rightarrow((\mathcal{A}\rightarrow\mathcal{B})\rightarrow(\mathcal{A}\rightarrow\mathcal{C}))$.
3. $((\mathcal{A}\rightarrow\mathcal{B})\rightarrow((\mathcal{A}\rightarrow\mathcal{C})\rightarrow(\mathcal{A}\rightarrow(\mathcal{B}\wedge\mathcal{C}))))$.
4. $((\mathcal{A}\wedge\mathcal{B})\rightarrow\mathcal{A})$.
5. $((\mathcal{A}\wedge\mathcal{B})\rightarrow\mathcal{B})$.
6. $((\mathcal{A}\rightarrow\mathcal{C})\rightarrow((\mathcal{B}\rightarrow\mathcal{C})\rightarrow((\mathcal{A}\vee\mathcal{B})\rightarrow\mathcal{C}))$.
7. $(\mathcal{A}\rightarrow(\mathcal{A}\vee\mathcal{B}))$.
8. $(\mathcal{B}\rightarrow(\mathcal{A}\vee\mathcal{B}))$.
9. $((\mathcal{A}\rightarrow\mathcal{B})\rightarrow(\bar{\mathcal{B}}\rightarrow\bar{\mathcal{A}}))$.

10. $\overline{\overline{A}} \to A$.
11. $A \to \overline{\overline{A}}$.

Каждая аксиомная схема представляет собой бесконечное число аксиом после замены A, B, C на произвольные формулы исчисления высказываний.

Введем только одно правило вывода, с помощью которого из формул A и $A \to B$ получаем новую формулу B. Это правило называется **правилом заключения**.

Пользуясь аксиомами и правилом заключения, мы можем строить доказательства и получать новые формулы – теоремы.

Примеры теорем исчисления высказываний.

Пример 1. Покажем, что $(X \to X)$ – теорема. Для этой цели построим доказательство, т.е. последовательность формул, последней в которой должна быть формула $(X \to X)$:

1) $((X \to (\overline{\overline{X}} \to X)) \to ((X \to \overline{\overline{X}}) \to (X \to X)))$ (по схеме 2, где A заменено на X, B на $\overline{\overline{X}}$, C на X);
2) $(X \to (\overline{\overline{X}} \to X))$ (по схеме 1);
3) $((X \to \overline{\overline{X}}) \to (X \to X)))$ (из 2) и 1) по правилу заключения);
4) $(X \to \overline{\overline{X}})$ (по схеме 11);
5) $(X \to X)$ (из 4) и 3) по правилу заключения);

По определению доказательства и теоремы $(X \to X)$ есть теорема.

Пример 2. Покажем, что $((X \wedge Y) \to (Y \wedge X))$ – теорема.

Соответствующая последовательность формул (A заменяем на $X \wedge Y$, B- на Y, C- на X):

1) $((X \wedge Y) \to Y) \to (((X \wedge Y) \to X) \to ((X \wedge Y) \to (Y \wedge X)))$ (из схемы 3);
2) $((X \wedge Y) \to Y)$ (из схемы 5);
3) $(((X \wedge Y) \to X) \to ((X \wedge Y) \to (Y \wedge X)))$ (из 1), 2) по правилу заключения);
4) $((X \wedge Y) \to X)$ (из схемы 4);
5) $((X \wedge Y) \to (Y \wedge X))$ (из 3), 4) по правилу заключения).

Исследуем некоторые свойства построенного исчисления высказываний. Будем интерпретировать исчисление высказываний в алгебру высказываний, ставя в соответствие каждой формуле исчисления высказываний аналогичную формулу алгебры высказываний.

Можно показать, что такая интерпретация адекватна. Покажем только правильность интерпретации.

Теорема 1. *Всякая теорема исчисления высказываний является тавтологией алгебры высказываний.*

Доказательство будем проводить индукцией по длине доказательства теоремы в исчислении высказываний.

Пусть $\mathcal{A}_n$ – теорема, а $\mathcal{A}_1$, $\mathcal{A}_2$, ... , $\mathcal{A}_n$ – ее доказательство. При n = 1 теорема $\mathcal{A}_n$ есть аксиома. Из определения операций в алгебре высказываний следует, что аксиома является тавтологией. Индуктивный шаг следует из того, что правило заключения, примененное к тавтологиям, приводит к тавтологии. ■

Эта теорема доказывает правильность интерпретации.

Относительно исчисления высказываний можно показать его полноту (по Посту) и непротиворечивость.

Непротиворечивость – свойство аксиоматической теории, состоящее в том, что в этой теории нельзя получить противоречие, т.е. доказать некоторое предложение вместе с его отрицанием или доказать некоторое заведомо абсурдное утверждение.

Для широкого класса аксиоматических теорий непротиворечивость имеет место тогда и только тогда, когда существует предложение, формулируемое в данной теории и недоказуемое в ней.

Теорема 2. *Исчисление высказываний непротиворечиво.*

Доказательство. Всякая теорема исчисления высказываний является тавтологией. Отрицание теоремы тождественно ложно в алгебре высказываний и значит, не является теоремой исчисления высказываний.

Теорема 3. (Теорема о полноте исчисления высказываний)

Пусть $\mathcal{A}(X_1,..., X_n)$ – формула, не являющаяся теоремой. Теория, которая получается из исчисления высказываний добавлением в качестве аксиом всех формул, получающихся из $\mathcal{A}(X_1,..., X_n)$ заменой переменных высказываний на произвольные формулы, противоречива.

Доказательство. Формула $\mathcal{A}(X_1,..., X_n)$ не является тавтологией алгебры высказываний, поэтому существует набор

$(\alpha_1 , ... , \alpha_n)$ такой, что $\mathcal{A}(\alpha_1 , ... , \alpha_n) = 0$.

Рассмотрим формулу $\mathcal{A}$, которая получится из $\mathcal{A}(X_1,..., X_n)$ заменой каждого переменного высказывания X_i на аксиому, если

$\alpha_i = 1$ и на отрицание аксиомы, если $\alpha_i = 0$. Формула $\mathcal{A}$ тождественно ложна, следовательно, $\bar{\mathcal{A}}$ – тавтология.

В силу адекватности интерпретации исчисления высказываний в алгебру высказываний формула $\bar{\mathcal{A}}$ является теоремой исчисления высказываний. Формула же $\mathcal{A}$ – аксиома полученного исчисления.

Получили, что формулы $\mathcal{A}$ и $\bar{\mathcal{A}}$ являются теоремами исчисления высказываний. Покажем, что произвольная формула $\mathcal{B}$ является теоремой. Это следует из цепочки формул:

$(\bar{\mathcal{B}} \to \mathcal{A}) \to (\bar{\mathcal{A}} \to \bar{\bar{\mathcal{B}}})$ (схема 9);

$\mathcal{A} \to (\bar{\mathcal{B}} \to \bar{\mathcal{A}})$ (схема 1);

$\mathcal{A}$ (аксиома);

$\bar{\mathcal{B}} \to \mathcal{A}$ (по правилу заключения);

$\bar{\mathcal{A}} \to \bar{\bar{\mathcal{B}}}$ (по правилу заключения);

$\bar{\mathcal{A}}$ (теорема);

$\bar{\bar{\mathcal{B}}}$ (по правилу заключения);

$\bar{\bar{\mathcal{B}}} \to \mathcal{B}$ (схема 10);

$\mathcal{B}$ (по правилу заключения).

Полученная теория противоречива. ■

Эта теорема показывает полноту (по Посту) исчисления высказываний.

1.3 Исчисление предикатов

Алфавит исчисления предикатов состоит из предметных переменных

$$x , y , z , \dots , x_1 , x_2 , x_3 , \dots ,$$

переменных высказываний

$$X , Y , Z , \dots , X_1 , X_2 , X_3 , \dots ,$$

переменных предикатов

$$W^p , U^k , \dots , W^1{}_1 , W^1{}_2 , \dots , W^i{}_j , \dots ,$$

символов логических операций

$$\wedge , \vee , \to , \bar{} , \forall , \exists$$

и скобок (,).

Определим **формулы** исчисления предикатов.

1. Переменное высказывание – формула.
2. Если W^p – переменный предикат, то $W^p(x_1 , x_2 , \dots , x_p)$ – форму-

ла. Предметные переменные этой формулы свободны.

3. Если $\mathcal{A}$ – формула и y – свободная переменная в ней, то$\forall$ y $\mathcal{A}$ и $\exists$ y $\mathcal{A}$ – формулы. Свободными переменными этих формул являются все свободные переменные формулы $\mathcal{A}$, исключая y. Связанными переменными формул $\forall$ y $\mathcal{A}$ и $\exists$ y $\mathcal{A}$ являются все связанные переменные формулы $\mathcal{A}$ и переменная y.
4. Если $\mathcal{A}$ и $\mathcal{D}$ формулы такие, что в них нет предметных переменных, связанных в одной формуле и свободных в другой, то $(\mathcal{A} \wedge \mathcal{D})$, $(\mathcal{A} \vee \mathcal{D})$, $(\mathcal{A} \to \mathcal{D})$, $\overline{\mathcal{A}}$ – формулы. Свободными переменными этих формул являются все свободные переменные формул $\mathcal{A}$ и $\mathcal{D}$, а связанными – связанные переменные формул $\mathcal{A}$ и $\mathcal{D}$.

Выпишем **аксиомные схемы** исчисления предикатов. В качестве первых схем возьмем схемы 1 – 11 исчисления высказываний и к ним добавим следующие две схемы.

12. $\forall$ x $\mathcal{A}(x) \to \mathcal{A}(y)$.
13. $\mathcal{A}(y) \to \exists$ x $\mathcal{A}(x)$.

К **правилам вывода** исчисления предикатов кроме уже известного правила заключения отнесем следующие:

1. Пусть $(\mathcal{A}(x)) \to \mathcal{D}$ и $\mathcal{D}$ не содержит предметной переменной x, тогда $((\exists$ x $\mathcal{A}(x)) \to \mathcal{D})$.
2. Пусть $(\mathcal{D} \to \mathcal{A}(x))$ и $\mathcal{D}$ не содержит предметной переменной x, тогда $(\mathcal{D} \to (\forall$ x $\mathcal{A}(x)))$.

Пример. Покажем, что $((\forall$ x $\mathcal{A}(x)) \to (\exists$ x $\mathcal{A}(x)))$ – теорема для произвольной формулы $\mathcal{A}(x)$:

1) $(\forall$ x $\mathcal{A}(x) \to (\mathcal{A}(y) \to \exists$ x $\mathcal{A}(x))) \to ((\forall$ x $\mathcal{A}(x) \to \mathcal{A}(y)) \to (\forall$ x $\mathcal{A}(x) \to \exists$ x $\mathcal{A}(x)))$ (схема 2);
2) $((\mathcal{A}(y) \to \exists$ x $\mathcal{A}(x)) \to (\forall$ x $\mathcal{A}(x) \to (\mathcal{A}(y) \to \exists$ x $\mathcal{A}(x))))$ (схема 1);
3) $(\mathcal{A}(y) \to \exists$ x $\mathcal{A}(x))$ (схема 13);
4) $(\forall$ x $\mathcal{A}(x) \to (\mathcal{A}(y) \to \exists$ x $\mathcal{A}(x)))$ (из 2) и 3) по правилу заключения);
5) $((\forall$ x $\mathcal{A}(x) \to \mathcal{A}(y)) \to (\forall$ x $\mathcal{A}(x) \to \exists$ x $\mathcal{A}(x)))$ (из 1) и 4) по правилу заключения);

6) $(\forall x\, \mathcal{A}(x) \to \mathcal{A}(y))$ (схема 12);

7) $(\forall x\, \mathcal{A}(x) \to \exists x\, \mathcal{A}(x))$ (из 5) и 6) по правилу заключения).

Исчисление предикатов можно интерпретировать в логику предикатов. Было установлено, что эта интерпретация **адекватна**. Однако при решении этого вопроса не удается ограничиться средствами конечной метатеории ввиду того, что в постановку вопроса входит понятие тождественно истинной формулы (тавтологии). Эта формула должна содержать рассмотрения всех интерпретаций, а значит и интерпретаций с бесконечным полем.

Можно показать **непротиворечивость** рассмотренного исчисления предикатов. Доказательство его непротиворечивости можно свести к доказательству непротиворечивости исчисления высказываний, поставив в соответствие каждой теореме исчисления предикатов теорему исчисления высказываний.

Вопрос **полноты** (по Посту) в исчислении предикатов решается отрицательно. Существует формула $\exists x\, W(x) \to \forall x\, W(x)$, которая не является теоремой, и добавление которой не нарушает непротиворечивости исчисления высказываний.

2 ОСНОВЫ ТЕОРИИ АЛГОРИТМОВ

Теория алгоритмов – раздел математики, изучающий теоретические возможности эффективных процедур вычисления (алгоритмов) и их приложения. Теория алгоритмов является крупнейшим достижением науки XX века. Теория электронных вычислительных машин, теория и практика программирования не могут обойтись без нее. Кибернетика и математическая логика предъявляют на нее свои права: во многих учебниках и справочниках отмечается, что она входит крупным разделом в эти науки. Однако теория алгоритмов является самостоятельной наукой и имеет собственный предмет исследования.

Само название теории говорит о том, что ее предмет – алгоритмы. Понятие алгоритма является и очень простым, и очень сложным. Его простота в многочисленности алгоритмов, с которыми мы встречаемся повсюду, в их обыденности. Но эти же обстоятельства делают понятие алгоритма туманным, расплывчатым, трудно поддающимся строгому научному определению.

2.1 Интуитивное понятие алгоритма и проблема его уточнения

Содержательные явления, которые привели к образованию понятия «алгоритм», прослеживаются в математике в течение всего времени ее существования. Слово «алгоритм» происходит от algorithmi – латинской формы написания имени узбекского математика Хорезми (по-арабски: аль-Хорезми), который сформулировал в IX веке правила четырех арифметических действий над числами в десятичной системе счисления. В Европе совокупность этих правил стали называть «алгоризм», «алгорифм». Затем это слово переродилось в «алгоритм» и стало общим названием отдельных правил определенного вида (и не только правил арифметических действий). Длительное время этот термин употребляли только математики, обозначая им правила решения различных задач.

Только в 30-е годы XX века понятие алгоритма стало объектом математического изучения, а до этого этим понятием только пользовались.

С появлением ЭВМ понятие алгоритма получило широкую известность. Развитие ЭВМ и методов программирования способство-

вало уяснению того факта, что разработка алгоритмов является необходимым этапом автоматизации.

Сейчас слово «алгоритм» вышло за пределы математики. Его стали применять в самых различных областях. Под ним понимают точно сформулированное правило, назначение которого – быть руководством для достижения необходимого результата. Иными словами, алгоритм – точно определенное правило действий (предписание, программа), для которого задано указание, как и в какой последовательности необходимо применять это правило к исходным данным задачи, чтобы получить ее решение. Перечислим характерные свойства алгоритма.

1. **Дискретность**. Алгоритм описывает процесс последовательного построения величин, идущий в дискретном времени. Необходимый для вычисления интервал времени разбит на малые отрезки – такты. Система величин в конце каждого такта получается в результате осуществления элементарного шага алгоритма из системы величин, имеющейся к началу такта.
2. **Определенность** (детерминированность). Программа преобразований в каждом такте однозначно определена.
3. **Результативность** (направленность). Алгоритм направлен на получение определенного результата. В частности, если вычисляемая функция в данном такте не определена, совокупность правил определяет, что нужно считать результатом применения алгоритма.
4. **Массовость**. Исходные величины могут варьироваться в известных пределах. Алгоритм служит для решения не одной конкретной задачи, а целого класса задач.

Эти свойства алгоритмов являются эмпирическими, подмеченными для всех известных алгоритмов. Рассмотренное понятие алгоритма, даже подкрепленное перечислением данных свойств, нельзя считать математически строгим. Его называют непосредственным, или интуитивным, понятием алгоритма, оно лишь объясняет смысл слова «алгоритм», но не определяет, что следует понимать под «правилом действия».

На протяжении длительного времени интуитивное понятие алгоритма в своей основе не изменялось, хотя и приобретало все большую выразительность. Математики удовлетворялись его содержательным пониманием, поскольку этот термин рассматривался

только в связи с построением конкретных алгоритмов и в положительных высказываниях, типа «для решения таких-то задач имеется алгоритм и вот в чем он состоит». Теоремы о несуществовании алгоритма не могли быть доказаны в силу нечеткости интуитивного определения. Как уже отмечалось, только в 30-х годах XX века возникла необходимость в рассмотрении общих способов формализации задач и процессов их решения, в уточнении понятия алгоритма как объекта математической теории. Эта необходимость возникла в связи с вопросом обоснования математики и с развитием вычислительной математики и вычислительной техники.

К настоящему времени разработаны теории, ведущие к уточнению понятия алгоритма. Основанием для одного из уточнений служит теория рекурсивных функций, другие уточнения связаны с понятиями машин Тьюринга и нормального алгоритма Маркова. Эти теории иногда называют традиционными теориями алгоритмов и не без основания, так как в связи с бурным развитием теории алгоритмов упомянутые теории уже стали классикой. Далее будут последовательно изложены основы данных теорий, каждая из которых уточняет понятие алгоритма. Но предварительно заметим, что уточнение распространяется не на все алгоритмы, а лишь на узкое их семейство. Можно сказать, что каждая теория является теорией некоторых избранных алгоритмов. Избранные алгоритмы каждого вида пригодны для решения ряда теоретических вопросов.

Доказано, что в теоретическом отношении все рассмотренные теории эквивалентны, т.е. научные результаты, полученные с помощью алгоритмов, изученных в какой-либо из этих теорий, могут быть также получены с помощью алгоритмов, изученных в любой другой. Тем не менее, отказаться от одной из них в пользу другой теории нельзя, поскольку в одних случаях легче получить результат с помощью алгоритмов одного класса, а в других – с помощью алгоритмов другого класса. Связь каждой теории избранных алгоритмов со всеми остальными алгоритмами осуществляется с помощью основных тезисов теорий. Но основной тезис позволяет выявлять случаи невозможности алгоритмов, однако ничего не дает нам, если требуется получить хороший, удобный для практики алгоритм. Кроме того, нужно иметь в виду, что основной тезис каждой теории избранных алгоритмов является лишь очень вероятной гипотезой, а не строгой теоремой.

2.2 Элементы теории рекурсивных функций

В алгоритмических проблемах речь обычно идет о существовании алгоритма для вычисления целочисленных значений функции, зависящей от целочисленных аргументов, т.е. **числовой функции**. К алгоритму вычисления числовой функции может быть сведен алгоритм вычисления любой имеющей практическое значение функции. Приведем пример. В математике рассматриваются функции, определенные на континуальных множествах, например на множестве точек некоторого отрезка. Однако на практике любая величина может быть измерена лишь с ограниченной точностью, причем это ограничение точности является принципиальным: для осуществления измерения с нулевой погрешностью требуется затратить бесконечно большое количество энергии. Таким образом, на практике всегда существует некоторый «порог различимости», определяемый точностью измерений, из-за этого множество значений любой физической величины оказывается счетным. Путем нумерации элементы счетного множества превращаются в целочисленные номера, следовательно, алгоритм вычисления функции, определенной на отрезке, сводится к алгоритму вычисления числовой функции.

Числовые функции, значения которых можно вычислять с помощью некоторого (единого для данной функции) алгоритма, называются **вычислимыми** функциями. В этом определении используется интуитивное понятие алгоритма, поэтому и понятие вычислимой функции оказывается интуитивным. Тем не менее, при переходе от алгоритмов к вычислимым функциям возникает одно очень важное обстоятельство. Совокупность процессов, удовлетворяющих интуитивному понятию алгоритма, весьма обширна и трудно обозрима. В то же время совокупность вычислимых функций для самых разных пониманий этих процессов, оказалась одной и той же и притом легко описываемой в обычных математических терминах. Эта точно описанная совокупность, совпадающая с совокупностью всех вычислимых функций при самом широком понимании алгоритма, носит название рекурсивных функций.

2.2.1 Основные понятия

Рассмотрим два каких-либо множества:

$$X = \{x_1, x_2, ...\}, Y = \{y_1, y_2, ...\}.$$

Если некоторым элементам множества X поставлены в соответствие однозначно определенные элементы множества Y, то говорят, что задана **одноместная частичная функция** из X в Y. Совокупность тех элементов множества X, у которых есть соответствующие элементы в Y, называется **областью определения** функции, а совокупность тех элементов множества Y, которые соответствуют некоторым элементам множества X, называется **совокупностью значений** (областью значений) функции. Если область определения функции из X в Y совпадает с множеством X, то функция называется **всюду определенной**, или просто функцией.

Функцию можно определить и как подмножество $F \subset X \times Y$, если для каждого элемента $x \in X$, найдется не более одного элемента $y \in Y$ так, что пара $(x, y) \in F$. При этом если для каждого элемента x имеется элемент y, образующий с x пару $(x, y) \in F$, то функция является всюду определенной, в противном случае она называется частично определенной или **частичной** функцией.

Сопоставим с декартовым произведением двух множеств прямоугольную решетку, узлы которой взаимно однозначно соответствуют элементам декартова произведения. Приведем пример, поясняющий введенные понятия для множеств $X = \{x_1, x_2, x_3, x_4\}$ и $Y = \{y_1, y_2, y_3\}$(рис. 2.1). На рис. 2.1,а изображено подмножество декартова произведения множеств X и Y, не являющееся функцией; на рис. 2.1,б – являющееся всюду определенной функцией; на рис. 2.1,в – частичной функцией. Частичная функция из $X_1 \times X_2 \times \ldots \times X_n$ в Y называется частичной функцией от n переменных, или **n-местной частичной функцией**. Обозначим через N множество всех натуральных чисел. Частичная функция из $N^{(k)} = N \times N \times \ldots \times N$ в N называется **k-местной числовой частичной функцией**.

Математика не накладывает никаких ограничений на соответствие (закон) применяеемый для определения функции. Допускается любой мыслимый закон. Таким законом может быть некоторый ал-

горитм. В этом случае функцию называют вычислимой, так как известен способ получения ее значений.

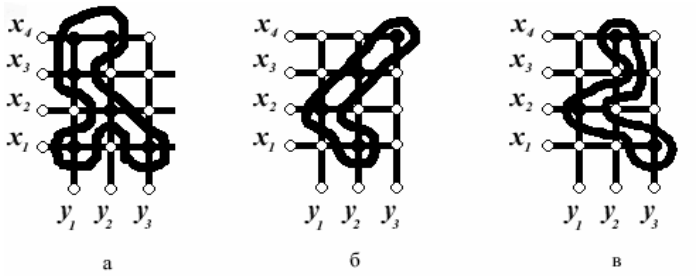

Рис. 2.1 – Примеры подмножеств декартова произведения X × У

Рекурсивными называют один частный класс вычислимых функций. Алгоритмы, являющиеся законами их задания, называются алгоритмами, сопутствующими рекурсивным функциям.

Для построения четко выделенного класса вычислимых функций нужны некоторые упрощающие предположения. Ранее уже отмечалась возможность выразить разные математические понятия с помощью целых неотрицательных чисел. Поэтому ограничимся случаем, когда и независимые переменные, и функции могут принимать только целые неотрицательные значения (натуральные числа).

Следующие числовые функции называются **простейшими**:

- функция следования $s(x) = x' = x + 1$;
- функция-константа $C_a^n(x_1, \ldots, x_n) = a$;
- функция тождества $I_m^n(x_1, \ldots, x_n) = x_m$ $(1 \le m \le n,$ $n = 1, 2, \ldots)$.

Сопутствующие этим функциям алгоритмы будут наиболее простыми, «одношаговыми».

Для функции следования (иначе – получение последователя) сопутствующий алгоритм гласит: если функциональный знак имеет вид s, то значением функции считать число, непосредственно следующее за числом, являющимся значением аргумента.

Сопутствующий алгоритм для функции-константы гласит: если функциональный знак имеет вид C_a^n, то любой совокупности значений аргументов данной функции ставится в соответствие ее значение a. Например:

$$C_0^1(2) = 0,\ C_1^3(4, 6, 7) = 1,\ C_5^n(7, 8, \ldots, 110) = 5.$$

Для функции тождества сопутствующий алгоритм гласит: если функциональный знак имеет вид I, то значением функции считать значение m-го (считая в функциональной записи слева-направо) независимого переменного. Например,

$$I_2^3 (5, 8, 6) = 8, \quad I_1^1 (3) = 3.$$

А вот запись $I_4^3 (x, y, z)$ не имеет смысла, так как в ней n = 3, m = 4, следовательно, не выполнено условие $1 \le m \le n$.

2.2.2 Преобразования функций

Преобразования функций называются **операторами**. Рассмотрим основные операторы, с помощью которых, исходя из рекурсивных функций, можно подучить новые функции, которые по определению тоже будем считать рекурсивными. Эти операторы, по сути, будут алгоритмами, на их основе можно получать новые алгоритмы.

Оператор подстановки (суперпозиции)

Пусть задано n каких-либо m-местных частичных функций f_1, …, f_n из A в B и пусть задана частичная n-местная функция f из B в C. Введем частичную функцию g из A в B такую, что

$$g (x_1, \ldots, x_m) = f (f_1 (x_1, \ldots, x_m), \ldots, f_n (x_1, \ldots, x_m))$$

для любых $x_1, \ldots, x_m$ из A.

Преобразование, с помощью которого получена функция g из f_1, …, f_n, называется **оператором подстановки** или суперпозиции и обозначается S^{n+1}, где (n+1) – число функций.

Алгоритм, сопутствующий этому оператору, гласит: «Значения функций f_1, …, f_n принять за значения аргументов функции и вычислить ее значение».

Оператор подстановки определен для функций f_1, …, f_n с одинаковым числом переменных. Затруднение при подстановке функций с разным числом переменных преодолевается введением фиктивных переменных с помощью функций тождества. Например,

$$\varphi (x_1, x_2) = \varphi (I_1^3 (x_1, x_2, x_3), I_2^3 (x_1, x_2, x_3)) = \psi (x_1, x_2, x_3).$$

Здесь переменная x_3 является фиктивной.

Обозначим через F^n множество всех частичных n-местных числовых функций. Оператор S^{n+1} является всюду определенной (n+1)-местной функцией из $F^n \times F^m \times \ldots F^m$ в F^m.

Если обозначить через F множество всех частичных числовых функций от произвольного числа переменных, то оператор S^{n+1}

можно рассматривать как частичную (n+1)-местную функцию из $F^{(n+1)}$ в F.

Оператор примитивной рекурсии

Пусть заданы какие-либо частичные числовые функции: n-местная g и (n + 2)-местная h. (n + 1)-местная частичная функция *f* возникает из функций g и h с помощью **оператора примитивной рекурсии** (или просто примитивной рекурсией), если для натуральных значений $x_1,\ldots, x_n$, y

$f(x_1,\ldots, x_n, 0) = g(x_1,\ldots, x_n)$,

$f(x_1,\ldots, x_n, y+1) = h(x_1,\ldots, x_n, y, f(x_1,\ldots, x_n, 0))$.

Этот оператор обозначим через R: $f = R(g, h)$. Найдем последовательно значения *f*.

$f(x_1,\ldots, x_n, 0) = g(x_1,\ldots, x_n)$,

$f(x_1,\ldots, x_n, 1) = h(x_1,\ldots, x_n, 0, g(x_1,\ldots, x_n))$,

$f(x_1,\ldots, x_n, m+1) = h(x_1,\ldots, x_n, m, f(x_1,\ldots, x_n, m))$.

Совокупность этих равенств для любых функций g и h однозначно определяет значения функции *f*. Итак, для каждых двух частичных числовых функций g от n переменных и h от (n+2) переменных существует одна и только одна функция *f* от (n+1) переменной, возникающая примитивной рекурсией (по данной переменной x_{n+1}).

При описании сути оператора примитивной рекурсии удобно не указывать аргументов из заданных функций ни в его функциональной записи, ни в записях двух других функций, подразумевая эти аргументы. Тогда можно сказать, что оператор примитивной рекурсии задает функцию с помощью двух условий, в которые входят функции g и h:

$f(0) = g$,

$f(i') = h(i, f(i))$.

Для удобства формулировки алгоритма условимся, что один из дополнительных аргументов, вошедший вместе с аргументами первой функции в число аргументов вновь получаемой функции, называется главным дополнительным аргументом, а другой аргумент, играющий вспомогательную роль при выполнении оператора, – вспомогательным аргументом. Тогда алгоритм, сопутствующий оператору примитивной рекурсии, гласит: «Значением получаемой функции для нулевого значения главного дополнительного аргумента считать значение исходной функции n-го аргумента. Значением

определяемой функции для каждого последующего значения главного аргумента считать значение второй из заданных функций при предыдущем значении главного аргумента и при значении вспомогательного аргумента, совпадающем с предыдущим значением определяемой функции».

Если для некоторых систем чисел $x_1,\ldots, x_n, y_1$ значение f не определено, то неопределенными будут и значения $f(x_1,\ldots, x_n, y)$ для всех $y > y_1$.

Если функции g и h всюду определены, то и $f = R(g, h)$ – всюду определенная функция.

Пример. Пусть $g = 0$, $h = 2x + y$, тогда

$f(0) = 0$,

$f(1) = h(0, g) = 0$,

$f(2) = h(1, f(1)) = 2$,

$f(3) = h(2, f(2)) = 2 \cdot 2 + 2 = 6$,

$f(4) = 2 \cdot 3 + 2 \cdot 2 + 2 = 12$,

$f(x + 1) = 2x + 2(x - 1) + \ldots + 2 \cdot 2 + 2 = x(x+1)$,

то есть функция $f(x + 1) = x(x + 1)$ возникает примитивной рекурсией из постоянной $g = 0$ и функции $h = 2x + y$.

При задании номера переменной, по которой осуществляется рекурсия, функция f однозначно определяется видом функций g и h. Переход к рекурсии по другой переменной в общем случае приводит к изменению функции.

Оператор минимизации

Этот оператор часто называют оператором наименьшего числа, а иногда – оператором построения по первому нулю. Последнее название лучше отражает суть рассматриваемого оператора и сопутствующего ему алгоритма.

Рассмотрим какую-нибудь n-местную ($n \geq 1$) частичную числовую функцию f. Допустим, что существует алгоритм для вычисления значений этой функции во всей области определения. Зафиксируем значения $x_1,\ldots, x_{n-1}$ первых (n-1) аргументов функции и рассмотрим уравнение

$$f(x_1,\ldots, x_{n-1}, y) = x_n.$$

Чтобы найти целочисленное решение y этого уравнения, будем по имеющемуся алгоритму вычислять значения $f(x_1,\ldots, x_{n-1}, y)$ последовательно для y = 0, 1, 2, … . Найденное в процессе такого вычисления наименьшее значение a, для которого

$$f(x_1,\ldots, x_{n-1}, a) = x_n,$$

обозначим через $\mu_y (f(x_1,\ldots, x_{n-1}, y) = x_n)$.

Данный процесс нахождения a не дает результатов, т.е. будет продолжаться бесконечно, в следующих случаях:

1) значение $f(x_1,\ldots, x_{n-1}, 0)$ не определено;
2) значения $f(x_1,\ldots, x_{n-1}, y)$ при y = 0, 1, … k определены, но не равны x_n , а значение $f(x_1,\ldots, x_{n-1}, k+1)$ не определено;
3) значения $f(x_1,\ldots, x_{n-1}, y)$ определены для всех y = 0, 1, 2,…, но отличны от x_n.

В этих случаях значение μ_y считается неопределенным. μ_y является частичной функцией от n переменных $x_1,\ldots, x_n$. Обозначим ее через Mf. Здесь M – символ **оператора минимизации**, преобразующего функцию f в Mf.

Иными словами этот оператор по заданной функции, зависящей от n аргументов, строит новую функцию от (n-1) аргументов. Исчезающий аргумент является вспомогательным и используется при выполнении оператора. Алгоритм, сопутствующий оператору минимизации гласит: «Придавать вспомогательному аргументу последовательные значения, начиная с нуля, до тех пор, пока не окажется, что функция f стала (в первый раз) равной нулю. Полученное значение вспомогательного аргумента принять за значение определяемой функции, соответствующее тем значениям основных аргументов, при которых осуществлялся описанный процесс».

Пример 1. Пусть $f(y) = 2^y$, тогда $Mf(x) = \mu_y (2^y = x)$. Функция $Mf(x)$ определена для значений $x = 2^n$, где n = 0, 1, 2,…. В этих случаях $Mf(x) = \log_2 x = n$. В частности,

$Mf(1) = 0$,

$Mf(2) = 1$.

Значения функции Mf при x = 0 и x = 3 не определены.

Пример 2. Пусть $f(x, y) = y - x$, тогда $Mf(x, z) = \mu_y(y - x = z)$. Значение этого выражения не определено, так как уже при y = 0 (x > 0) функция $f(x, y) = y - x$ не определена. Но, с другой стороны, уравнение y - x = z имеет решение y = z + x.

Этот пример показывает, что значение функции Mf не является в общем случае решением уравнения f (x_1,…, x_{n-1}, y) = x_n. Значение Mf совпадает с минимальным решением уравнения f (x_1,…, x_{n-1}, y) = =x_n, если функция f (x_1,…, x_{n-1}, y) для данного набора значений x_1,…, x_{n-1} определена при всех y $\leq$ Mf (x_1,…, x_n).

Если функция f одноместная, то функция Mf называется обращением функции f, или обратной функцией, ее обозначают часто через f^{-1}:

$$f^{-1}(x) = \mu_y (f(y) = x).$$

2.2.3 Примитивно-рекурсивные функции

Функция f называется **примитивно-рекурсивной**, если она является одной из простейших функций s, C_0^1, I_m^n или может быть получена из простейших функций с помощью конечного числа операторов подстановки и примитивной рекурсии. Это определение индуктивное: определены исходные, базовые примитивно-рекурсивные функции s, C_0^1, I_m^n и даны правила построения любых других примитивно-рекурсивных функций.

Операторы подстановки и примитивной рекурсии, примененные ко всюду определенным функциям, дают также всюду определенные функции, поэтому все примитивно-рекурсивные функции – всюду определенные.

Пример 1. Показать, что n-местная функция-константа C_a^n является примитивно рекурсивной. Действительно,

$$C_a^n(x_1,\ldots, x_n) = s(s(\ldots s(C_0^1(I_1^n(x_1,\ldots, x_n)))\ldots)),$$

т.е. функция C_a^n выражается с помощью подстановок через примитивно-рекурсивные функции s, C_0^1, I_1^n и, следовательно, является примитивно-рекурсивной.

Пример 2. Функция f (x, y) = x + y может быть построена по схеме примитивной рекурсии:

f (x, 0) = x = I_1^1(x),

f (x, y+1) = s (f (x, y)) = h (x, y, f (x, y)),

т.е. f (x, y) возникает примитивной рекурсией из примитивно-рекурсивных функций g = I_1^1(x) и h (x, y, z) = s (z).

Пример 3. Назовем сложение, умножение и возведение в степень соответственно действиями нулевой, первой и второй ступеней и обозначим соответственно

$$P_0(x, y) = x + y,\ P_1(x, y) = xy,\ P_2(x, y) = x^y.$$

Нетрудно заметить, что при n = 1, 2

$$P_n(x, 0) = g(x),$$

где g (x) = 0 при n = 1 и g (x) = 1 при n = 2,

$$P_n(x, y + 1) = P_{n-1}(x, P_n(x, y)).$$

Отсюда видно, что функция $P_n(x, y)$ (n = 1,2) возникает примитивной рекурсией из функций g (x) и h (x, y, z) = P_{n-1} (x, z). Следовательно, функции P_1 и P_2 – примитивно-рекурсивные.

Представление примитивно-рекурсивной функции с помощью операторов подстановки и примитивной рекурсии, примененных к функциям s, C_0^1, I_m^n, по существу, определяет способ вычисления значений функции. Поэтому доказательство примитивной рекурсивности какой-либо функции является одновременно доказательством существования алгоритма вычисления этой функции.

2.2.4 Частично рекурсивные функции

Функция *f* называется **частично рекурсивной**, если она может быть получена из s, C_0^1, I_m^n с помощью конечного числа операторов подстановки, примитивной рекурсии и минимизации. Из этого определения вытекают следующие свойства частично рекурсивных функций:

- каждая примитивно-рекурсивная функция является также и частично рекурсивной;

- класс частично рекурсивных функций шире класса примитивно-рекурсивных функций, так как все примитивно-рекурсивные функции всюду определены, а среди частично рекурсивных функций есть функции, не всюду определенные.

Примеры частично рекурсивных функций.

1. Частичная функция (x - y) может быть представлена в виде

   $$x - y = \mu_z (y + z = x),$$

   функция (y + z) является примитивно-рекурсивной, следовательно, функция (x - y) – частично рекурсивная.
2. Функция $x / y = \mu_z (yz = x)$, yz – примитивно-рекурсивная функция, поэтому x / y – частично-рекурсивная функция.

Понятие частично рекурсивной функции является одним из основных понятий теории алгоритмов. Какие бы классы алгоритмов до сих пор не строились, во всех случаях оказывалось, что числовые функции, вычисляемые посредством этих алгоритмов, были частично рекурсивными.

Американский логик и математик Алонзо Черч в 1936 г. высказал гипотезу о том, что понятием рекурсивной функции исчерпывается понятие вычислимой функции. Аналогичную гипотезу выдвинул независимо от Черча Стивен Клини (профессор Висконсинского университета). Эта гипотеза подтверждается всем предыдущим опытом математиков. Поэтому ныне общепринятой является следующая истинно научная гипотеза, обычно формулируемая как **тезис Черча**: *класс алгоритмически вычислимых частичных числовых функций совпадает с классом всех частично рекурсивных функций.*

Для каждой частично рекурсивной функции f существует вычислительный процесс, посредством которого любое натуральное число перерабатывается в значение f (x) функции f. Этот процесс не дает определенного результата в том и только том случае, если значение функции f в точке x не определено.

Рассмотрим способ получения всюду определенных частично рекурсивных функций. Введем оператор слабой минимизации:

$M^l f = Mf$, если функция Mf всюду определена;

$M^l f$ не определена, т.е. не существует, если функция Mf определена не всюду.

Функции, которые можно получить из простейших функций s, C_0^1, I_m^n применением конечного числа операторов подстановки, примитивной рекурсии и слабой минимизации, называются **общерекурсивными**.

Так как операторы R, M^l, S^i , примененные ко всюду определенным функциям, либо ничего не дают, либо дают снова функции всюду определенные, то все общерекурсивные функции – всюду определенные.

Если результат применения оператора слабой минимизации определен, то он совпадает с результатом применения оператора обычной минимизации. Поэтому общерекурсивные функции являются всюду определенными частично рекурсивными функциями. Справедливо и обратное: каждая всюду определенная частично рекурсивная функция общерекурсивна.

Перенося тезис Черча на алгоритмы, сопутствующие рекурсивным функциям, можно сформулировать следующую гипотезу: каков бы ни был алгоритм, перерабатывающий набор целых неотрицательных чисел в целые неотрицательные числа, существует эквивалентный ему алгоритм, сопутствующий рекурсивной функции.

Опираясь на эту гипотезу, можно сделать вывод о том, что выполнение алгоритма в определенном смысле эквивалентно вычислению значения рекурсивной функции, а невозможность рекурсивной функции означает и невозможность алгоритма.

Подчеркнем еще раз, что как тезис Черча, так и следующие из него гипотезы не имеют доказательства и принципиально не могут быть доказаны, поскольку в них речь идет об алгоритмах в интуитивном смысле, не являющихся математическими объектами.

2.3 Машины Тьюринга

Из интуитивного понятия алгоритма следует, что вычисления в соответствии с некоторым алгоритмом выполняются механически и их можно поручить машине. Процесс, происходящий в такой машине, должен обладать всеми свойствами алгоритмического процесса: дискретностью, результативностью, детерминированностью. Английский математик А. Тьюринг (1912-1954 г.г.) в 1936 г. выполнил, а в 1937 г. опубликовал работу, в которой уточнил понятие алгоритма, прибегая к воображаемой вычислительной машине, известной теперь под названием **машины Тьюринга**. Аналогичную концепцию машины позднее и независимо от Тьюринга ввел американский математик Э. Пост (1897-1954), поэтому упомянутую машину иногда называют машиной Тьюринга-Поста. Идея Тьюринга возникла еще до появления ЭВМ и потому, по-видимому, не зависит от них. Итак, машина Тьюринга – это математическое понятие, введенное как формальное уточнение интуитивного понятия алгоритма.

В каждой машине Тьюринга (рис 2.2) присутствуют три части: 1) внешняя память; 2) считывающая и записывающая головка; 3) управляющее устройство.

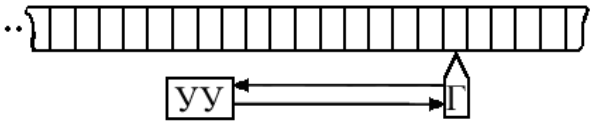

Рис. 2.2 – Состав машины Тьюринга

Внешняя память представляет собой неограниченную в обе стороны ленту, разделенную на ячейки одинаковой длины. Известны варианты машины Тьюринга с неограниченной лентой в правом направлении, а также с конечной лентой. В последнем случае предполагается, что справа могут пристраиваться новые ячейки, так что здесь лента тоже может считаться потенциально неограниченной в правом направлении. В каждой ячейке может быть один из символов a_1, ..., a_m . Ячейка, не содержащая ни одного из этих символов, называется пустой. Для обозначения пустой ячейки используют специальный символ a_0 (или 0). Если в ячейке записан символ a_i , то говорят, что ячейка находится в состоянии a_i. Множество символов $A = \{a_0, a_1,..., a_m\}$ называется **внешним алфавитом** машины.

Считывающая и записывающая головка – устройство, которое в каждый определенный момент времени располагается напротив некоторой ячейки ленты, которая называется **обозреваемой** или **воспринимаемой** ячейкой. По командам из управляющего устройства головка может стереть символ, записанный в обозреваемой ячейке, и записать новый, а также передвинуться вправо или влево вдоль ленты на одну ячейку.

Управляющее устройство вырабатывает команды, поступающие на головку. В свою очередь с головки на вход управляющего устройства поступает символ обозреваемой ячейки. Управляющее устройство может находиться в одном из конечного числа состояний, совокупность которых обозначается через $Q = \{q_0, q_1, ..., q_n\}$ и образует **внутренний алфавит** машины (или алфавит внутренних состояний), а состояние управляющего устройства называется **состоянием внутренней памяти**. Одно из внутренних состояний называется **заключительным** (обычно это q_0). В это состояние внутренняя память приходит по окончании работы машины; символ, обозначающий заключительное состояние, называется **стоп-символом**. Управляющее устройство работает в дискретном време-

ни, при этом команды, вырабатываемые им в i-м такте работы, определяются символом обозреваемой ячейки и состоянием внутренней памяти. В результате выполнения команд к началу (i+1)-го такта работы может быть изменен символ обозреваемой ячейки, записывающая головка может быть передвинута вправо или влево, а также может быть изменено состояние внутренней памяти. **Состоянием машины Тьюринга**, или ее **конфигурацией**, называется совокупность, образованная последовательностью символов состояний ячеек ленты a_{i1}, a_{i2}, ..., a_{ir}, символом внутреннего состояния q_j и номером k воспринимаемой ячейки. Эти данные записываются в виде машинного слова

$$a_{i1}, a_{i2}, ..., a_{ik-1}, q_j, a_{ik}, ..., a_{ir}.$$

Каждое машинное слово содержит лишь одно вхождение символа q_j (он записывается слева от обозреваемой ячейки). Работу машины Тьюринга можно представить как последовательное преобразование машинных слов в соответствии с некоторой программой. Цель работы машины Тьюринга – преобразование слова, записанного на ленте. Чтобы «запустить» машину Тьюринга, надо поместить ее головку напротив нужной ячейки и привести внутреннюю память машины Тьюринга в исходное состояние.

Работа машины Тьюринга состоит из тактов. В каждом из них выполняется преобразование конфигурации, в которой машина Тьюринга находится в данный момент времени i (i = 1, 2, ...), в конфигурацию, в которой машина Тьюринга будет находиться в момент i+1. Это преобразование включает в себя:

1) изменение состояния q_j в некоторое состояние q_l;
2) замену буквы a_{ik}, записанной в обозреваемой ячейке, некоторой буквой a_{pk};
3) сдвиг головки на одну ячейку влево или вправо (головка может и не сдвинуться).

Такое преобразование называется **командой** машины Тьюринга. Символически команда записывается в виде

$$q_i\, a_{ik} \rightarrow q_l\, a_{pk}\, R,$$

где R – одна из букв Л, П, С (влево, вправо, сохранение положения головки).

Совокупность всех команд, которые выполняет машина Тьюринга, называется ее **программой**.

Для каждой буквы $a_i \in A$ и состояния $q_j \in Q$ программа содержит в точности одну команду с левой частью $q_j\, a_i$. Поэтому работа машины Тьюринга определится однозначно, если фиксировать конфигурацию K_1, с которой она начинает работать. А именно: в первом такте K_1 преобразуется в K_2 выполнением единственной, применимой к K_1 команды; во втором такте K_2 преобразуется таким же образом в K_3 и т.д. Работа машины Тьюринга продолжается неограниченно, с какой бы конфигурации она не начиналась, однако можно ввести некоторые правила остановки этого процесса. Например, можно использовать понятие заключительных состояний, придя в которые, машина Тьюринга останавливается. Конфигурация, в которой машина Тьюринга останавливается, называется **заключительной конфигурацией**. Или, например, можно считать, что работа машины Тьюринга прекращается на некотором такте, если в этом такте (а, следовательно, и во всех дальнейших) изменения конфигурации не происходит.

В дальнейшем ограничимся рассмотрением машин Тьюринга с внешним алфавитом $A = \{0, 1\}$. Здесь 0 используется для обозначения пустых ячеек. Произвольное натуральное число x записывается на ленте в виде последовательности x+1 единиц: $11...1 = 1^{x+1}$.

При такой записи число 0 обозначается одной единицей, а пустые ячейки используются для разделения чисел на ленте. Так как любой алгоритм может быть сведен к алгоритму вычисления целочисленной функции, то двоичный внешний алфавит позволяет построить машину Тьюринга, реализующую любой алгоритм.

Пример 1. Машина, увеличивающая данное число на единицу – машина «1». Фактически она вычисляет известную из п.2.2.1 функцию следования. В начале работы машины головка воспринимает заполненную ячейку (ячейку, содержащую символ 1), и внутренняя память машины находится в состоянии q_1. В процессе работы машина находит справа пустую ячейку, печатает в ней единицу и останавливается. Условимся, что стрелка ➔ обозначает преобразование, осуществляемое машиной в результате выполнения программы. В данном случае имеем

$$01^x\, q_1 1^y\, 0\, 0 \rightarrow 01^{x+y}\, q_0 10.$$

Запишем совокупность команд, составляющих это программу:

$q_1 1 \rightarrow q_1 1 П$ – после выполнения раз этой команды машина придет в состояние $01^{x+y}\, q_1\, 0\, 0$;

$q_1 0 \rightarrow q_0 1C$ – после выполнения этой команды машина приходит в состояние $01^{x+y} q_0 10$.

Пример работы машины при x=3, y=2 приведен в табл. 2.1.

Состояние машины представлено в таблице словом, записанным на ленте с символом внутренней памяти, расположенным под воспринимаемой ячейкой.

Таблица 2.1 – Пример работы машины «1»

Номер такта	Состояние машины							
0	0	1	1	1	1 q_1	1	0	0
1	0	1	1	1	1	1 q_1	0	0
2	0	1	1	1	1	1	0 q_1	0
3	0	1	1	1	1	1	1 q_0	0

Программу машины Тьюринга можно представить в виде табл. 2.2, содержащей необходимые команды.

Таблица 2.2 – Программа машины «1»

q \ a	0	1
q_1	$q_0 1C$	$q_1 1П$

В этой таблице a – символ обозреваемой ячейки; q – символ состояния внутренней памяти.

Пример 2. Машина, «сдвигающая» головку вправо, на следующий массив единиц – машина «C^+». Необходимо осуществить преобразование

$$01^x q_1 1^y 0^z 1^w 0 \rightarrow 01^{x+y} 0^z 1^{w-1} q_0 10.$$

Здесь 0^z означает последовательность из z нулей. Машина находит следующий справа массив единиц и останавливается, воспринимая самую правую заполненную ячейку.

Программа содержит команды:

$q_1 1 \rightarrow q_1 1П$ (через y тактов команда дает слово $01^{x+y} q_1 0^z 1^w 0$),

$q_1 0 \rightarrow q_2 0П$,

$q_2 0 \to q_2 0П$ (выполнение этих команд приводит к слову $01^{x+y}0^z q_2\, 1^w 0$),

$q_2 1 \to q_3 1П$,

$q_3 1 \to q_3 1П$ (получается слово $01^{x+y}\, 0^z\ 1^w\, q_3\, 0$),

$q_3 0 \to q_0 0Л$ (получается заключительное слово).

Эта программа приведена в табл. 2.3.

Таблица 2.3 – Программа машины «C^+»

q \ a	0	1
q_1	$q_2 0П$	$q_1 1П$
q_2	$q_2 0П$	$q_3 1П$
q_3	$q_0 0Л$	$q_3 1П$

Композиция машин Тьюринга

Это понятие играет важную роль при синтезе машин Тьюринга, осуществляющих требуемые преобразования машинных слов.

Пусть заданы машины Тьюринга P и Q с общим внешним алфавитом $\{a_0, a_1, \ldots, a_m\}$ и внутренними алфавитами $\{q_0, q_1, \ldots, q_n\}$ и $\{q_0, q_1, \ldots, q_k\}$. Пусть машина Тьюринга P осуществляет преобразование машинного слова A в B, а машина Q – преобразование слова B в слово C. Требуется построить машину Тьюринга R, которая преобразовывала бы слово A в слово C. Такое преобразование будет осуществляться, если R сначала выполнит преобразование по программе машины Тьюринга P, а затем – машины Q. Очевидно, что программу машины R можно получить, объединив программы машин P и Q, причем в программе машины P стоп-символ q_0 заменить на символ q_{n+1}, а в программе машины Q символы q_i (i = 1,…, k) – символами q_{i+n}, оставив неизменным символ q_0. Тогда внутренняя память машины R после выполнения преобразований по программе машины P приходит в состояние q_{n+1}, после чего машина R начинает работу по программе машины Q. Машина R с внешним алфавитом $\{a_0, a_1, \ldots, a_m\}$, внутренним алфавитом $\{q_0, q_1, \ldots, q_{n+k}\}$ и программой, полученной из программ P и Q указанным способом, называется **композицией машин** P и Q, или **произведением** машины P на машину Q (обозначается: R = PQ).

Заметим, что программа машины R может быть получена подбором нужных команд. При этом найденная программа может ока-

заться короче программы, полученной в результате композиции машин. Однако понятие композиции дает общий метод синтеза машин, который облегчает построение сложных программ. Существенно то, что ряд машин Тьюринга, последовательно выполняющих свои программы, всегда можно заменить одной машиной, являющейся их композицией.

Произведение машин Тьюринга – некоммутативная операция, т.е. в общем случае $PQ \neq QP$. Произведение одинаковых машин Тьюринга называется **возведением в степень** и обозначается как P^n.

Пример 3. Используя понятие композиции машин Тьюринга, построим программу машины Т, сдвигающей головку вправо на следующий массив единиц, изображающий некоторое число w, увеличивающей это число на единицу и останавливающейся:

$$01^x q_1 1^y 0^z 1^{w+1} 0 \rightarrow 01^{x+y} 0^z 1^{w+1} q_0 10.$$

Данная машина может быть представлена композицией машины «C^+» (пример 2) и машины «1» (пример 1). Для получения таблицы команд машины Т заменим символ q_0 в табл. 2.3 и символ q_1 в табл.2.2 символом q_4. Полученная программа машины Т представлена в табл. 2.4.

Непосредственный подбор команд может привести к более короткой программе машины Т (табл. 2.5).

Таблица 2.4 – Программа машины «C^+1»

q \ a	0	1
q_1	q_2 0П	q_1 1П
q_2	q_2 0П	q_3 1П
q_3	q_4 0Л	q_3 1П
q_4	q_0 1С	q_4 1П

Таблица 2.5 – Более короткая программа машины Т

q \ a	0	1
q_1	q_2 0П	q_1 1П
q_2	q_2 0П	q_3 1П
q_3	q_0 1С	q_3 1П

Итерация машин Тьюринга

Кроме операции композиции при синтезе программ машин Тьюринга используется операция итерации. Рассмотрим машину T_1, имеющую k заключительных состояний внутренней памяти q_{01}, …, q_{0k}. Отождествим одно из этих состояний, (например, q_{0i}) с начальным состоянием машины. Операция отождествления одного из заключительных состояний внутренней памяти машины с ее исходным состоянием называется **операцией итерации**. Машину T, полученную в результате итерации машины T_1, обозначим следующим образом:

$$T = \dot{T}_1 \begin{cases} (q_{01}), \\ \dots \\ (\dot{q}_{0i}), \\ \dots \\ (q_{0k}). \end{cases}$$

Точки над символами указывают на отождествление i–го заключительного состояния с начальным состоянием машины T_1. В результате итерации получим машину, внутренняя память которой во время работы совершает один или несколько замкнутых циклов. Если при выполнении программы внутренняя память машины должна прийти в состояние q_{0i}, то она возвращается в начальное состояние и машина вновь начинает работу. Это продолжается до тех пор, пока ее внутренняя память не придет в какое-либо из остальных заключительных состояний. Если машина T_1 имеет одно заключительное состояние, то в результате итерации получаем машину, не имеющую заключительных состояний. Такая машина работает по замкнутому циклу бесконечно или останавливается, если в процессе работы получено машинное слово, к которому программа неприменима.

Итерация может объединиться с композицией машин. Например, в машине

$$T = \dot{T}_1 \begin{cases} (q_{01}), \\ \dots \\ (\dot{q}_{0i}), \dot{T}_2 \\ \dots \\ (q_{0k}). \end{cases}$$

после того, как внутренняя память приходит в состояние q_{0i}, выпол-

няется программа T_2, после чего внутренняя память приходит в начальное состояние и вновь выполняется программа T_1.

Использование операций композиции и итерации позволяет построить из простых машин более сложные. При этом сложные машины удается описать компактным выражением.

Теорема Тьюринга

Рассмотрим вычисление с помощью машин Тьюринга значений какой-либо функции $f(x_1, \ldots, x_n)$. Определим стандартную форму записи исходных данных и полученного результата и стандартное положение головки в начале и в конце вычисления. Число x записывается совокупностью x + 1 единиц. Считаем, что два числа расположены рядом (подряд), если между записями чисел имеется одна пустая ячейка. Для простоты условимся, что внешняя память представляет собой ленту, неограниченную в правом направлении. Оставим пустой самую левую ячейку ленты и запишем далее подряд числа $x_1, \ldots, x_n$. Будем говорить, что головка воспринимает систему чисел $< x_1, \ldots, x_n >$ в стандартном положении, если эти числа записаны подряд и головка воспринимает самую правую заполненную ячейку в представлении последнего из чисел x_n.

Пусть в начале работы машина воспринимает в стандартном положении систему чисел $< x_1, \ldots, x_n >$. Будем говорить, что машина вычисляет частичную функцию $f(x_1, \ldots, x_n)$, если:

- для всех систем чисел $< x_1, \ldots, x_n >$, для которых функция $f(x_1, \ldots, x_n)$ определена, по окончании работы машина воспринимает в стандартном положении систему чисел $< x_1, \ldots, x_n, f(x_1, \ldots, x_n) >$ и внутренняя память находится в заключительном состоянии, т.е.

$$01^{x1+1} \ldots 0\ 1^{xn+1} q_1 10 \rightarrow 01^{x1+1} \ldots 0\ 1^{xn+1}\ 0\ 1^{f(x1,\ \ldots,xn)} q_0 1\ 0$$

- для всех систем чисел $< x_1, \ldots, x_n >$, для которых функция не определена, машина, начав работать, не приходит в заключительное состояние.

Функция называется вычислимой на машине Тьюринга, если существует машина Тьюринга, вычисляющая эту функцию.

Теорема. *Класс функций, вычислимых на машинах Тьюринга, совпадает с классом частично рекурсивных функций.*

Эта теорема доказывает эквивалентность уточнений понятия алгоритма с помощью машин Тьюринга и с помощью теории рекурсивных функций.

2.4 Нормальные алгорифмы Маркова

Советский математик А.А. Марков в 1947 г. разработал другой путь уточнения алгоритма. Заметим, что термины «алгоритм» и «алгорифм» в то время существовали равноправно, и лишь гораздо позднее термин «алгоритм» получил более широкое распространение. А.А. Марковым разработана строгая теория класса алгоритмов, которые он назвал **нормальными алгорифмами**. Наряду с рекурсивными функциями и машинами Тьюринга нормальные алгорифмы получили известность в качестве одного из наиболее удобных уточнений общего интуитивного представления об алгоритме.

Как и машины Тьюринга, нормальные алгоритмы в качестве исходных данных и искомых результатов имеют строки символов (букв) – **слова**. Предположим, что, как и в случае машин Тьюринга, заранее определен некоторый алфавит. Обозначим его через **А**. Букву, одинаковую с одной из букв, входящих в алфавит **А**, называют **буквой в А**. Слово, состоящее из букв в **А**, называют **словом в А**. Для удобства рассуждений допускаются и пустые слова, которые не имеют в своем составе ни одной буквы.

Если **А** и **В –** два алфавита, причем каждая буква алфавита **А** является буквой в **В**, а хотя бы одна из букв алфавита **В** не является буквой в **А**, то **В** называется расширением **А**.

Например, если

А = {а, б, в, г, д}, **В** = {1, а, б, в, г, д, е},

то **В** является расширением **А**, поскольку содержит две буквы («1» и «е»), не являющиеся буквами в **А**, тогда как все буквы алфавита **А** являются буквами в **В**.

Рассмотрим какое-либо конкретное слово для определенности в алфавите русских букв, например слово «система». Из него можно вырезать «подслова», например «сис», «ист», «тема» и т.п., наконец однобуквенное слово, например «а». Про такие «подслова» говорят, что они **входят** в рассматриваемое слово или являются **вхождениями** в него. Отметим, что в наше слово «система» входит и пустое слово, причем несколько раз. Оно входит перед первой буквой, между каждыми двумя буквами и, наконец, после последней буквы, т.е., в нашем случае, 8 раз.

Условимся обозначать слова заглавными латинскими буквами, при этом они не должны быть буквами в применяемом алфавите.

Если задано некоторое слово и нами выбрана буква, являющаяся его обозначением (именем), то будем ставить между ними знак равенства «=». Обращаясь к нашему примеру, мы можем написать: для слова R = «система» слово P = «тема» является вхождением.

Марковской подстановкой называется операция над словами, задаваемая с помощью пары слов (P, Q) и заключающаяся в следующем. Если задано исходное слово R, то в нем находят первое вхождение P (если таковое имеется) и, не изменяя остальных частей слова R, заменяют в нем это вхождение словом Q. Полученное слово и есть результат применения марковской подстановки (P, Q) к слову R. Если же нет первого вхождения слова P в слово R (при этом нет вообще ни одного вхождения P в R), то считается, что марковская подстановка неприменима к слову R.

Частными случаями марковских подстановок являются (, Q), (P,) и (,). В первом из этих примеров P, во втором Q, а в третьем и P, и Q являются пустыми словами. В табл. 2.6 приведены некоторые примеры преобразования слов с помощью марковских подстановок.

Таблица 2.6 – Примеры марковских подстановок

N п/п	Преобразуемое слово	Марковская подстановка	Результат
1	192375923	(923, 0000)	1000075923
2	Функция	(,)	Функция
3	Паровоз	(овоз,)	Пар
4	Слово	(,)	Слово
5	Слово	(ра, да)	(результата нет)

Будем рассматривать слова в некотором алфавите **А**. Предположим, что символы «→» и «→•» не являются буквами в **А**. Записи P→Q и P→•Q будем называть записями марковской подстановки (P, Q), причем первую из них будем называть **подстановкой**, а вторую – **заключительной подстановкой**. Эти записи будем называть **формулами**, различая в них левую часть P и правую часть Q.

Записью нормального алгорифма в алфавите А называется столбец формул, левые и правые части которых являются словами в **А**. Выполнение нормального алгорифма $\mathcal{A}$ применительно к исходному данному R, являющемуся словом в **А**, заключается в следующем.

Двигаясь по столбцу формул, ищут первую формулу, левая часть которой входит в преобразуемое слово. Если такой формулы не найдется, то процесс окончен. Если же она найдется, то выполняют марковскую подстановку, соответствующую данной формуле. Затем смотрят, является ли выполненная подстановка заключительной. Если да, то процесс окончен, и результат переработки слова R посредством алгорифма $\mathcal{A}$ обозначается через $\mathcal{A}$ (R). Если нет, то весь процесс повторяют с самого начала.

Пусть **B** является расширением алфавита **A**. Нормальный алгорифм в **B**, который слова в **A**, если он к ним применим, перерабатывает в результаты, являющиеся словами в **A**, называется нормальным алгорифмом над **A**. В некоторых случаях построение алгорифма над **A** гораздо легче, чем построение нормального алгорифма в **A**.

Рассмотрим пример нормального алгорифма. Мы уже показывали, как можно для функции

$$s(x) = x + 1$$

построить машину Тьюринга. В качестве алфавита **A** возьмем перечень арабских цифр, то есть

$$\mathbf{A} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}.$$

Нормальный алгорифм будем строить не в **A**, а над **A**, добавив к **A** еще две буквы: x и y. Для экономии места столбец формул нашего искомого нормального алгорифма запишем в виде четырех подстолбцов:

0y →• 1	**8y →• 9**	**x5 → 5x**	**3x → 3y**
1y →• 2	**9y → y0**	**x6 → 6x**	**4x → 4y**
2y →• 3	**y →• 1**	**x7 → 7x**	**5x → 5y**
3y →• 4	**x0 → 0x**	**x8 → 8x**	**6x → 6y**
4y →• 5	**x1 → 1x**	**x9 → 9x**	**7x → 7y**
5y →• 6	**x2 → 2x**	**0x → 0y**	**8x → 8y**
6y →• 7	**x3 → 3x**	**1x → 1y**	**9x → 9y**
7y →• 8	**x4 → 4x**	**2x → 2y**	**→ x**

Если бы этот алгорифм мы применили к исходному слову R – пустому, то получили бы слова: x, xx, xxx, ... , т.е. бесконечный процесс. Это означает, что к пустому слову данный алгорифм неприме-

ним.

Рассмотрим его применение к слову R = 299.

Применение алгорифма к этому слову даст промежуточные результаты: x299 (в силу последней строки), 2x99, 29x9, 299x (в силу строк, расположенных в конце второго и начале третьего подстолбцов), 299y (в силу предпоследней формулы), 29y0, 2y00 (в результате двукратного применения второй формулы второго подстолбца), и приведет к искомому результату 300 (в силу третьей формулы алгорифма). Итак, мы получили S(299) = 300, что и требуется.

Мы определили понятие нормального алгорифма в алфавите **A** и нормального алгорифма над **A**. Одноместная частичная словарная функция F (R), заданная в алфавите **A**, называется **нормально вычислимой**, если существует нормальный алгорифм $\mathcal{A}$ над алфавитом **A** такой, что для каждого слова R в алфавите **A** выполнено равенство F(R) = $\mathcal{A}$(R).

В частности, алгорифм со схемой $\Lambda \to\bullet\ \Lambda$ вычисляет функцию F(R)=R, а алгорифм со схемой $\Lambda \to \Lambda$ вычисляет нигде не определенную функцию.

Доказана следующая общая

Теорема. *Класс нормально вычислимых частичных функций, заданных в произвольном алфавите* ***A****, совпадает с классом всех одноместных частично рекурсивных словарных функций в алфавите* ***A***.

Аналогом тезиса Черча для нормальных алгорифмов является следующий **принцип нормализации А.А. Маркова**: *всякий алгоритм в алфавите* ***A*** *вполне эквивалентен относительно* ***A*** *некоторому нормальному алгорифму над* ***A***.

Нормальные алгорифмы оказались удобным рабочим аппаратом во многих исследованиях, требующих точного понятия алгоритма, особенно тогда, когда основные объекты рассмотрения имеют неарифметическую природу и допускают удобное представление в виде слов в некоторых алфавитах.

2.5 Задачи и упражнения

1. Если значения примитивно-рекурсивной, общерекурсивной или частично рекурсивной функции изменить лишь на конечном

множестве точек, то будет ли новая функция снова соответственно примитивно-рекурсивной или частично рекурсивной?

2. Покажите, что примитивно-рекурсивна каждая
   а) конечная совокупность чисел;
   б) совокупность чисел вида $a * n + b$, $n = 0, 1, 2, ...$;
   в) совокупность чисел вида $a * b^n$, $n = 0, 1, 2, ...$.
3. Составьте программу машины Тьюринга, складывающей любые два натуральных числа n_1 и n_2 , представленные на ленте двумя сериями из $n_1 + 1$ и $n_2 + 1$ единиц, разделенных ячейкой, в которой записан символ 0. (Результатом вычисления считается число единиц в выражении, написанном на ленте после окончания вычисления.) В начальном состоянии головка считывает первый символ 1.
4. Составьте программу машины Тьюринга, проверяющей истинность равенства $x = 0$.
5. Составьте программу машины Тьюринга, сдвигающей головку влево на следующий массив чисел.
6. Составьте программу машины Тьюринга, уменьшающей данное число на единицу.
7. Используя понятие композиции машин Тьюринга и программы машин, решающих задачи 5 и 6, постройте программу машины, сдвигающей головку влево на следующий массив единиц, изображающий некоторое число, и уменьшающей это число на единицу.
8. Постройте нормальные алгорифмы Маркова для вычисления простейших числовых функций: функции - константы, функции тождества.

3 ЭЛЕМЕНТЫ КОМБИНАТОРНОГО АНАЛИЗА

Целый ряд математических моделей процессов управления представляет собой дискретные модели комбинаторного типа. Так, например, комбинаторные методы используются для решения транспортных задач, в частности задач по составлению расписаний; для составления планов производства и реализации продукции и т.д. Исследование таких моделей и методов их решения относится к области комбинаторного анализа.

Что же изучают в комбинаторном анализе и какие типы задач решают? Рассмотрим для начала несколько задач.

1. Поступающий в ТУСУР должен сдать три экзамена при пятибальной системе оценок. Для поступления достаточно набрать 13 баллов. Сколькими способами он может сдать экзамены (разумеется, не получив ни одной двойки)?

2. Как отыскать кратчайший путь для почтальона, обязанного обслуживать заданное число населенных пунктов? Расстояние между каждой парой пунктов известно.

3. Сколько ферзей или других шахматных фигур достаточно, чтобы они держали под боем все клетки шахматной доски? Сколькими способами они могут быть расставлены?

4. На сколько частей делят пространство n плоскостей, из которых никакие четыре не проходят через одну и ту же точку, никакие три не проходят через одну и ту же прямую и никакие две не параллельны, а любые три плоскости имеют общую точку?

Общим во всех этих задачах является то, что в них рассматриваются дискретные (составленные из отдельных, обособленных элементов) множества. Эти множества в большинстве случаев конечны, но могут быть и бесконечными, составленными из неограниченно большого числа элементов.

Задачи комбинаторного анализа можно разбить на три класса:

а) задачи о количестве решений, то есть о числе дискретных построений, удовлетворяющих поставленным условиям, или перечислительные задачи;

б) задачи, решающие вопрос о существовании или несуществовании конфигурации, удовлетворяющей условиям, то есть о наличии или отсутствии решения;

в) задачи отыскания алгоритма получения конфигурации, а также выделения из заданной совокупности конфигураций таких, которые обладали бы избранным свойством в наибольшей или наименьшей степени, или задачи оптимизации.

Элементы комбинаторных суждений появились еще в глубокой древности, на заре формирования математической науки. В ходе истории они развивались совместно с другими разделами математики, Нетрудно увидеть, что в ряде областей современной математики: теории чисел, алгебре, геометрии, математической логике и др. многие основные понятия и методы имеют дискретную природу и обладают устойчивыми связями. Это позволяет рассматривать задачи комбинаторного анализа в различных интерпретациях, исследовать проблемы различной природы с единой точки зрения, В наше время в связи с развитием вычислительной техники возможности дискретных методов исследования и их значение резко возросли. Наряду с исторически сложившейся комбинаторной теорией (комбинаторным анализом) в современной математике существуют: теория графов, геометрия чисел, дискретный и конечный анализы, исследование операций.

3.1 Постановка задач комбинаторного программирования

Изучение основ комбинаторного анализа полезно начать с рассмотрения нескольких примеров, содержащих постановки задач оптимизации, типичных для АСУ и относящихся к области комбинаторного программирования. Комбинаторное программирование – это подраздел математического программирования, включающий чисто дискретные задачи и специфические методы их исследования. Началом развития комбинаторного программирования считают 40-50-е годы, первыми объектами исследования явились простые идеализированные модели, такие как задача коммивояжера, задача обработки деталей на двух станках и т.д.

Задачи комбинаторного программирования возникают во многих областях практической деятельности: организация и размещение производства, планирование транспортных перевозок, распределение ресурсов и др. Задачи этого типа можно интерпретировать как задачи оптимизации функций, определенных на заданном множест-

ве выборок (комбинаций) из конечного числа элементов.

При рассмотрении примеров особое внимание уделим формализации задач, т.к. уже этот этап часто вызывает трудности на практике.

3.1.1 Задачи определения очередности выполнения заданий

Проблема. Для заданного множества заданий (работ, операций) исполнителю требуется выбрать наилучшую последовательность их выполнения. Понятие «наилучшей» последовательности зависит от конкретных условий и чаще всего определяется директивными сроками окончания отдельных заданий.

Введем следующие обозначения:

$I_N = \{1,2,...,N\}$ – множество номеров заданий, число которых равно N;

$\pi(\pi_1, \pi_2, ..., \pi_N)$ – перестановка элементов множества I_N, определяющая очередность выполнения заданий (например, $\pi_5 = 3$ означает, что третье задание выполняется пятым по порядку);

$\tau(j)$ – время выполнения j-го задания;

$T(j)$ – директивный срок окончания j-го задания;

$a(j)$ – штраф, налагаемый на исполнителя за невыполнение j-го задания к сроку $T(j)$;

$Q(x) = \begin{cases} 0, x \le 0 \\ 1, x > 0 \end{cases}$ – функция скалярной переменной.

Множество возможных решений рассматриваемой задачи составляют всевозможные перестановки π из элементов множества I_N. Обозначим это множество перестановок Π_N. На множестве Π_N ищется наилучшее в некотором смысле, т.е. **оптимальное**, решение. Формализация понятия оптимальности осуществляется в задачах математического программирования заданием на множестве возможных решений так называемой **целевой функции**. Для любой

пары возможных решений лучшим считается то, для которого значение целевой функции «лучше». Если смысл выбранного критерия таков, что «лучше» означает больше (или меньше), то задача сводится к отысканию среди возможных решений такого, на котором целевая функция принимает наибольшее (наименьшее) значение.

В нашей задаче цель – **возможно меньшее** (с учетом штрафов) **отклонение от директивных сроков**. Поэтому целевой функцией может быть функция $F_1(\pi)$:

$$F_1(\pi) = \sum_{j \in I_N} a(\pi j) Q[\sum_{K=1}^{j} \tau(\pi_k) - T(\pi_j)]. \qquad (1)$$

Для каждой очередности π функция (1) определяет величину суммарного штрафа за нарушение директивных сроков.

Таким образом, задача сводится к нахождению минимума функции $F_1(\pi)$ на множестве Π_N. Рассмотрим конкретный вариант решения данной задачи. Пусть $a(j) = 1,\ j \in I_N$. В этом случае функция $F_1(\pi)$ определяет количество невыполненных к сроку заданий для данной очередности π. Исходные данные к задаче приведены в табл. 3.1.

Таблица 3.1 – К решению задачи о заданиях

j	1	2	3	4	5	6	7	8
$\tau(j)$	10	6	3	1	4	8	7	6
T(j)	35	20	11	8	6	25	28	9

Пусть π' =(1,2,3,4,5,6,7,8), π'' =(5,4,3,2,7,1,8,6).

Соответствующие значения целевой функции: $F_1(\pi') = 6$, $F_1(\pi'') = 2$, т.е. значения F_1 существенно зависят от выбранной очередности π. Общее же число возможных решений равно 8! = 40320.

3.1.2 Задачи определения порядка обработки деталей

Проблема. При заданных временах и последовательностях об-

работки деталей найти такой порядок их запуска, при котором суммарное время обработки минимально.

Пусть

I_N – множество номеров деталей, подлежащих обработке

π – перестановка из элементов множества I_N, задающая порядок запуска деталей в производство;

$I_M = \{1,2,...,M\}$ – множество номеров станков.

Сделаем следующие, естественные во многих случаях предположения:

1) каждая из деталей должна быть обработана на каждом из M станков;
2) начавшись, обработка любой детали идет без перерывов;
3) последовательность прохождения деталей по станкам должна быть одинакова для всех деталей (для определенности примем, что нумерация станков соответствует последовательности обработки деталей;
4) заданы времена $\tau(i, j)$ обработки j-й детали на i-м станке $(i \in I_M, j \in I_N)$.

Определим время $T_k(\pi_j, \pi)$ завершения обработки детали на первых k станках (для данной перестановки π):

$$T_1(\pi_j, \pi) = \sum_{i=1}^{j} \tau(1, \pi_i), \qquad (2)$$

$$T_k(\pi_1, \pi) = \sum_{i=1}^{k} \tau(i, \pi_1) \qquad (3)$$

Время $T_k(\pi_j, \pi)$ получим, воспользовавшись следующими соображениями. Обработка детали π_j на k-м станке может начаться только после окончания ее обработки на $(k-1)$-м станке и по окончании обработки на k-м станке детали π_{j-1}. Тогда для определения $T_k(\pi_j, \pi)$ имеем рекуррентное соотношение

$$T_k(\pi_j, \pi) = \tau(k, \pi_j) + \max\{T_{k-1}(\pi_j, \pi), T_k(\pi_{j-1}, \pi)\}. \qquad (4)$$

Критерий оптимальности в задачах такого типа – время обработки всех деталей на всех станках, которое стремятся уменьшить. Поэтому в качестве целевой функции $F_2(\pi)$ можно выбрать время окончания обработки детали π_N на станке M:

$$F_2(\pi) = T_M(\pi_N, \pi). \qquad (5)$$

Множеством возможных решений является множество Π_N, на котором с помощью соотношений (2) – (5) определена целевая функция $F_2(\pi)$.

Формальная постановка задачи.

На множестве π_N найти перестановку, минимизирующую функцию $F_2(\pi)$.

Проиллюстрируем постановку на конкретном примере. Пусть $N = 5$, $M = 2$. Значения $\tau(i, j)$ приведены в табл. 3.2.

Таблица 3.2 – К задаче о деталях

i \ j	1	2	3	4	5
1	4	3	1	5	2
2	1	2	5	2	4

Рассмотрим очередности $\pi' = (1,2,3,4,5)$ и $\pi'' = (3,5,4,2,1)$. Соответствующие значения целевой функции $F_2(\pi') = 20$, $F_2(\pi'') = 16$ можно получить из соотношений (6.2 – 6.5) или построив диаграмму обработки деталей для данных перестановок.

3.1.3 Задачи распределения заданий

Проблема. Для заданного множества заданий и заданного множества исполнителей распределить задания наиболее рационально (с точки зрения затрат ресурсов или времени на их выполне-

ние).

Пусть

I_N – множество номеров заданий;

I_M – множество номеров исполнителей;

$t(i, j)$ – затраты на выполнение j-го задания i-м исполнителем;

$r(r_1, r_2, ..., r_M)$ – разбиение множества I_N на M подмножеств (подмножества r_i не имеют общих элементов, $\bigcup_{i=1}^{M} r_i = I_N$, некоторые r_i могут быть равны $\varnothing$);

$R(N, M)$ – множество всех разбиений указанного типа. Проиллюстрируем понятие $R(N, M)$. Пусть $N = 5, M = 3$. Тогда одним из элементов $R(N, M)$ является $r' = \{(2,5), (1,3,4), \varnothing\}$. Т.е. имеются три исполнителя и пять заданий. Первому исполнителю назначаются задания с номерами 2, 5, второму с номерами 1, 3, 4, третьему задания не назначаются. Множество $R(N, M)$ есть множество возможных решений задачи.

Задачи этого типа допускают два критерия оптимальности:

1) критерий наибольшей суммарной эффективности;
2) критерий равномерной загрузки исполнителей.

Соответствующие целевые функции

$$F_1(r) = \sum_{i \in i_M} \sum_{j \in r_i} t(i, j), \qquad (6)$$

$$F_2(r) = \max_{i \in I_M} \sum_{j \in r_i} t(i, j). \qquad (7)$$

Функция $F_1(r)$ определяет для заданного распределения r суммарные затраты (например, временные) на выполнение всех заданий. Функция $F_2(r)$ определяет для заданного r максимальную загрузку исполнителей.

Таким образом, задача распределения заданий по критерию

минимума общих затрат сводится к минимизации функции $F_1(r)$ на множестве $R(N,M)$. Та же задача, но с критерием максимально равномерной загрузки исполнителей сводится к минимизации функции $F_2(r)$ на множестве $R(N,M)$.

В задачах распределения заданий могут присутствовать **ограничения**, делающие неприемлемыми некоторые из возможных решений. Те из возможных решений, которые удовлетворяют ограничениям, называются **допустимыми**. Ограничения в задачах распределения заданий обычно носят ресурсный характер. Это означает, что на выполнение любого задания каждый исполнитель затрачивает некоторое количество ресурса определенного типа (время, фонд заработной платы, количество людей, станков и т.д.). Поскольку суммарное количество ресурса всякого типа ограничено, не всякое разбиение является допустимым.

Формализуем ограничения:

$S(i,j,k)$ – количество ресурса k-го типа, используемое i-м исполнителем для выполнения j-го задания $(i \in I_M, j \in I_N, k = \{1,2,...,K\})$;

$S_2(i,k)$ – наличное количество этого ресурса у i-го исполнителя;

$S_1(i,k)$ – обязательный уровень его использования. Тогда для допустимого разбиения r должно выполняться соотношение

$$S_1(i,k) \le \sum_{j \in r_i} S(i,j,k) \le S_2(i,k), i \in I_M, k \in I_k. \quad (8)$$

Формальная постановка задач распределения заданий. Минимизировать функции $F_1(r)$ или $F_2(r)$ на множестве $R(N,M)$ при ограничениях (8).

В задачах этого типа впервые появилось понятие множества допустимых решений, широко используемое в математическом программировании. Таким образом, в оптимизационных комбинаторных задачах оперируют с тремя вложенными друг в друга множествами: множеством возможных решений, множеством допустимых решений, множеством оптимальных решений.

3.1.4 Задача о назначениях

Задача о назначениях является частным случаем задачи распределения заданий при следующих условиях:

$$N = M, K = 1, S_1(i,1) = S_2(i,1) = S(i, j,1) = 1$$

Она состоит, таким образом, в назначении каждому исполнителю ровно одного задания. Поэтому возможна постановка этой задачи на множестве перестановок $\pi \in \Pi_N$, в которой $\pi_j = i(j \in I_M)$.

Функции $F_1(r) u F_2(r)$ для множества Π_N легко пересчитываются в функции

$$F_3(\pi) = \sum_{j \in I_N} (\pi_j, j),$$

$$F_4(\pi) = \max_{j \in I_N} \{t(\pi_j, j)\}.$$

Таким образом, задача назначения состоит в выборе в матрице $\|t_{ij}\|$ размера $N \times N$ по одному элементу в каждой строке и столбце так, чтобы сумма выбранных элементов (функция $F_3(\pi)$) или максимальный из них (функция $F_4(\pi)$) были минимальными.

3.2 Основные понятия и операции комбинаторики

Переходя непосредственно к изучению основ комбинаторного анализа, выделим в комбинаторных исследованиях два основных вида операций:

1) отбор подмножеств;
2) упорядочение элементов.

С первой операцией связано **понятие выборки.** При этом под выборкой понимают как осуществление операции отбора, так и ее результат – само выбранное подмножество. Поэтому если из множества A, состоящего из n элементов, выбрано r – подмножество, то будем называть его r **-выборкой**, где r – **объем выборки**.

Если r -выборки рассматриваются с учетом порядка элементов в них, то они называются r **-перестановками.** Если этот порядок не принимается во внимание, то соответствующие выборки называются r **-сочетаниями**.

Пополним полученные ранее знания о множествах следующими логическими правилами.

Правило суммы. Если из множества S подмножества A можно выбрать m способами, а подмножество B, отличное от A, n способами и при этом выборы A и B таковы, что взаимно исключают друг друга и не могут быть получены одновременно, то выбор из S множества $A \bigcup B$ можно осуществить $m+n$ способами.

Это правило можно сформулировать в терминах теории множеств следующим образом. Если даны m - множество P и n - множество Q (элементами их являются в нашем случае выбранные подмножества), то при $P \bigcap Q = \varnothing$ $P \bigcup Q$ будет ($(m+n)$ - множеством.

Обобщенное правило суммы. Если дано разбиение множества M

$M = T_1 \bigcup T_2 \bigcup ... \bigcup T_r$,

где $T_i \bigcap T_j = \varnothing, i \neq j, j = 1,2,...,r$, и если T_i есть n_i - множество $(i = 1,2,...,r)$, то множество M есть $(\sum_{i=1}^{r} n_i)$ - множество.

Правило произведения. Если из множества S подмножество A может быть выбрано m способами, а после каждого такого выбора можно n способами выбрать подмножество B, то выбор A и B в указанном порядке можно осуществить $m \times n$ способами.

В терминах теории множеств это правило соответствует правилу декартова произведения множеств: если P является m - множеством, а Q есть n -множество, то $P \times Q$ окажется $(m \cdot n)$ - множеством.

Обобщенное правило произведения. Пусть T_i есть n_i -

множество $(i = 1,...,r)$. Построим множества $M_1 = T_1$, $M_2 = M_1 \times T_2$ (оно будет $(n_1 n_2)$- множеством), $M_3 = M_2 T_3, ..., M_r = M_{r-1} \times T_r$. Тогда M_r будет $(n_1 n_2 ... n_r)$- множеством.

В комбинаторном анализе будем рассматривать множества всюду упорядоченные, т.е. такие, для любой пары которых можно установить отношение порядка.

3.3 Выборки и упорядочения

Пусть имеется n- множество A. Две r-выборки $a = (a_1, a_2, ..., a_r)$ и $b = (b_1, b_2, ...b_r)$ называются упорядоченными, если $a = b$ лишь при условии, что $a_i = b_i, i = 1,2,...,r,$ и неупорядоченными, если $a = b$ лишь при условии, что каждое a_i равно некоторому b_j, $j = 1,2,...,r.$ Уточним, исходя из этого, некоторые определения.

Упорядоченные r-выборки из n-множества A называются r- **перестановками**, если все r элементов различны, и r- перестановками с повторениями, если среди r элементов имеются одинаковые (равные).

Неупорядоченные r-выборки множества A называются r- **сочетаниями**, если все r элементов различны, и r-сочетаниями с повторениями, при наличии одинаковых элементов.

Так, например, две выборки (2,3,4,6,1,1) и (1,3,4,1,6,2) из множества (1,2,3,4,5) являются одинаковыми 6 – сочетаниями с повторениями и разными 6 – перестановками с повторениями.

Очевидно, что задача выделения r-выборки не имеет в общем случае единственного решения. Поэтому подсчет числа r-выборок заданного типа из n-множеств исторически явился одной из первых задач комбинаторики.

Найдем **число всех возможных r-перестановок** (без повторений) из n-множеств. Обозначим это число через $P(n,r)$. Будем рассуждать следующим образом. В n-множестве имеется n возможностей для выбора первого элемента перестановки. Как только

такой выбор сделан, остаются $(n-1)$ возможностей для выбора второго элемента, затем $(n-2)$ возможностей для выбора третьего и т.д. Для выбора r-го элемента будет $(n-r+1)$ возможностей. Применяя правило произведения, получим

$$P = (n,r) = n(n-1)...(n-r+1).$$

Отсюда, в частности, следует, что $P(n,n) = n!$, т.е. n различных предметов, расположенных в n различных местах, можно представить $n!$ способами. Для полноты результата полагают

$$P(n,0) = 0! = 1$$

Подсчитаем теперь **число P возможных r-перестановок с повторениями.** В этом случае после выбора любого элемента r-перестановки остаются все те же n возможностей для выбора следующего элемента. Таким образом, число r-перестановок с повторениями из n-множества равно

$$P = n^r.$$

Приведенные рассуждения хорошо иллюстрируются урновой схемой широко используемой в теории вероятностей: из урны, в которую помещены n шаров, поочередно вынимают r шаров, при этом вынутый шар либо возвращают (выбор с возвращениями), либо не возвращают (выбор без возвращений).

Определим **число $C(n,r)$ r-сочетаний** (без повторений). Так как r-сочетания - это неупорядоченные r-выборки, а число способов упорядочения r-выборки равно $r!$, то $C(n,r)$ будет в $r!$ раз меньше, чем число r-перестановок из элементов n-множества. Следовательно,

$$C(n,r) = \frac{P(n,r)}{r!} = \frac{n(n-1)...(n-r+1)}{r!} = \frac{n!}{r!(n-r)!}$$

Отсюда следует, что

$$C(n,n-r) = C(n,r),$$

$$C(n,n) = C(n,0) = 1.$$

Найдем **число $H(n,r)$ r- сочетаний с повторениями**. Чтобы глубже понять специфику комбинаторных рассуждений, рас-

смотрим три способа определения этого числа.

1. Пусть множество S находится во взаимно однозначном соответствии с множеством первых n натуральных чисел. Тогда вместо r-выборок из S можно рассматривать соответствующие им r- выборки из $T = \{1,2,...,n\}$. Любая r-выборка с повторениями из T может быть представлена как $a = (a_1, a_2, ..., a_r)$, где $a_1 \le a_2 \le ... \le a_r$. Поставим ей в соответствие выборку $a' = (a_1 + 0, a_2 + 1, ..., a_r + r - 1)$, в которой все элементы различны. Так как число всех возможных выборок типа a из множества T равно числу всех возможных выборок типа a' из множества $T' = \{1,2,..., n, n+1, ..., n+r-1\}$, то искомое число $H(n,r) = C(n+r-1, r)$.

2. К r-сочетанию с повторениями из множества S припишем все элементы множества S и полученные $n+r$ элементов разделим $(n-1)$ черточкой, так как промежутков между n различными элементами имеется $n-1$. Таким образом, задача сводится к отысканию числа способов, которыми можно расположить $(n-1)$ черточку в $(n+r-1)$ промежутке между $n+r$ элементами. Это число равно $C(n+r-1, n-1) = C(n+r-1, r)$, откуда $H(n,r) = C(n+r-1, r)$.

3. Получим реккурентную последовательность для $H(n,r)$. Очевидно, что $H(n,1) = n, H(1,r) = 1$.

Зафиксируем в множестве S некоторый элемент. Тогда каждое r-сочетание либо содержит этот элемент, либо нет. В первом случае остальные $(r-1)$ элементы этого r-сочетания можно выбрать $H(n, r-1)$ способами. Во втором случае r-сочетания выбираются из $(n-1)$ элемента, поэтому число таких сочетаний равно $H(n-1, r)$. Используя правило суммы, получим

$$H(n,r) = H(n, r-1) + H(n-1, r).$$

Теперь последовательно находим

$$H(n,2) = H(n,1) + H(n-1,2) = H(n,1) + H(n-1,1) +$$

$$+ H(n-2,2) = n + (n-1) + (n-2) + \ldots + 1 = \frac{n(n+1)}{2} =$$

$$= C(n+1,2),$$

$$H(n,3) = H(n,2) + H(n-1,3) = H(n,2) + H(n-1,2) +$$

$$+ H(n-2,3) = H(n,2) + H(n-1,2) + H(n-2,2) + \ldots +$$

$$H(1,3) = C(n+1,2) + C(n,2) + \ldots + 1 = \frac{(n+1)n}{2} +$$

$$+ \frac{n(n-1)}{2} + \ldots + 1 = \frac{(n+2)(n+1)n}{2 \cdot 3} = C(n+2,3)$$

Легко убедиться, что

$$H(n,r) = C(n+r-1,r).$$

3.4. Разложение на циклы

Рассмотрим еще одно понятие, связанное с операцией упорядочения – **перестановку**. Она может рассматриваться с двух позиций:

1) как упорядоченная совокупность элементов данного множества;
2) как нарушение стандартного порядка, называемого обычно естественным (например, алфавитным или цифровым).

Первый случай приводит к уже рассмотренным r-перестановкам, где $r \le n$.

Второй случай приводит к n-перестановкам, называемым просто **перестановками** или **подстановками**.

Пусть, например, имеется перестановка

P = (4 3 7 5 6 9 2 8 1 12 11 10),

являющаяся нарушением естественного порядка первых двенадцати чисел натурального ряда. Эта запись показывает, что при перестановке P элемент 1 перешел в 4, 2 – в 3, 3 – в 7 и т.д. Перестановку P можно записать иначе:

P = (1 4 5 6 9) (2 3 7) (10 12) (8) (11), (11)

где каждая скобка есть перестановка, действующая только на элементы, заключенные в ней.

Представление перестановки в виде (11) называется **разложением на циклы.** Любая перестановка может быть разложена на циклы.

Если циклические перестановки внутри циклов считать тождественными, как например, (2 3 7) (3 7 2) (7 2 3), **то разложение будет единственным.**

Пусть некоторая перестановка содержит K_1 циклов, состоящих из одного элемента, или 1-циклов, затем K_2 2-циклов, K_3 3-циклов и т.д. Тогда она может быть записана как $(K_1, K_2, ..., K_n)$ - перестановка, или

$$1^{K_1} \cdot 2^{K_2} \cdot 3^{K_3} ... r^{K_r}. \qquad (12)$$

Очевидно, что $\sum_{i=1}^{r} i \cdot k_i = n.$

Теорема. *Число перестановок вида (12) равно*

$$P(K_1, K_2, ..., K_r) = \frac{n!}{1^{K_1} \cdot K_1! 2^{K_2} K_2! ... r^{K_r} K_r!}.$$

Доказательство. Среди всех возможных $n!$ перестановок из n элементов имеются тождественные перестановки типа (12).

При этом возможны две ситуации:

1) перестановки содержат все циклы с одними и теми же элементами в одном и том же циклическом порядке;
2) перестановки отличаются лишь относительным расположением циклов, что несущественно.

Так как r -цикл может начинаться с любого из r элементов, то можно получить r дубликатов цикла, соответствующего первой ситуации. Общее число дубликатов составит

$$1^{K_1} \cdot 2^{K_2} \cdot 3^{K_3} ... r^{K_r}.$$

Далее, если число r -циклов равно K_r, то их можно представить $K_r!$ способами, поэтому дубликатов, соответствующих второй ситуации, окажется

$K_1!K_2!...K_r!$

Следовательно,

$$P(K_1, K_2,..., K_r) = \frac{n!}{1^{K_1} K_1!2^{K_2} K_2!...r^{K_r} K_r}.$$

3.5.Размещения и заполнения

Во многих задачах требуется некоторую совокупность элементов (зерен, болтов и т.д.) распределить на некоторое множество ячеек (коробок, ящиков и т.п.). Фактически это задачи о разбиении некоторой совокупности элементов на множество классов. Оба понятия (размещение и заполнение) используются для обозначения как операций, так и их результата – полученной ситуации.

Задачи этого класса существуют с давних пор и имеют разработанную методику решения. Постановки их различны: разбиения множеств, рассечения графов, группировки станков и т.д.

Задачей на размещение будем называть задачу о числе размещений элементов по ячейкам.

Задачей на заполнение будем называть задачу на размещение, в которой существенны число элементов либо порядок их в заданных или произвольно выбранных ячейках.

Для подсчета числа размещений важно знать, являются ли элементы данного множества и ячейки различными. В соответствии с этим признаком выделяют четыре класса задач.

1. Элементы множества и ячейки различимы между собой.
2. Элементы множества неразличимы, ячейки различимы.
3. Элементы множества различимы, ячейки неразличимы.
4. Как элементы множества, так и ячейки неразличимы между собой.

Внутри каждого из этих классов задачи могут различаться видом отображений, задаваемых конкретными условиями. Рассмотрим задачи, относящиеся к первым двум классам.

Число способов размещения n различных объектов по различным ячейкам. Эквивалентами являются: образование слов длины r из алфавита, содержащего n букв; последовательный выбор r шаров с немедленным из возвращением; образование r - перестановок с повторениями из n элементов. Очевидно, что число

таких способов размещения равно $P = n^r$.

В частном случае, когда каждая ячейка может содержать только один элемент,

$$P = n(n-1)...(n-r+1) = \frac{n!}{n-r!}.$$

Число способов размещения n одинаковых объектов по r различным ячейкам. Возможны две разновидности задач этого класса:

а) Ни одна ячейка не пуста (рис. 3.1). Задача состоит в нахождении числа способов провести $r-1$ линий в $n-1$ промежутках. Это число равно $C(n-1, r-1)$.

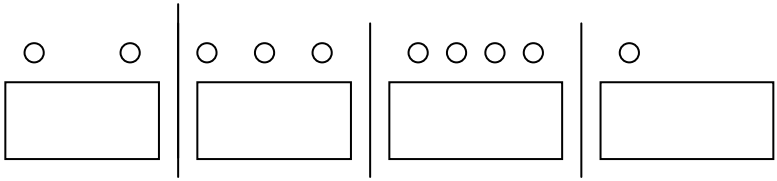

Рис.3.1 – Задача о размещениях

Варианты задачи: число способов окрашивания r цветами n одинаковых объектов; число r - сочетаний с повторениями, в которых каждый элемент использован.

б) Ячейки могут быть пустыми. В этом случае к множеству элементов присоединяют r символических «пустых элементов». Задача сводится к определению числа способов провести $r-1$ линий в $n+r-1$ промежутках между элементами. Число таких способов равно $C(n, n+r-1)$. К этой разновидности относится, например, задача нахождения числа решений уравнения

$$x_1 + x_2 + ... + x_r = n$$

в неотрицательных числах.

Решение задач третьего и четвертого классов представляет значительно большие трудности, однако возможно в ряде простейших случаев.

3.6. Производящие функции

До сих пор мы занимались прямыми, или «элементарными», методами подсчета количества комбинаторных конфигураций. Производящие функции представляют собой аппарат для реализации косвенных методов подсчета количества комбинаторных конфигураций.

Метод производящих функций является одним из самых развитых и самых сильных в приложениях методов комбинаторного анализа. Основные идеи этого метода впервые были высказаны в конце XVIII века в работах Лапласа по теории вероятностей.

Поясним идею метода на примерах производящих функций для сочетаний и перестановок.

Производящие функции для сочетаний. Для примера рассмотрим три объекта, обозначив их x_1, x_2, x_3, образуем произведение

$$(1+x_1t)(1+x_2t)(1+x_3t).$$

Раскрыв скобки и расположив полином по степеням t, получим

$$1+(x_1+x_2+x_3)t+(x_1x_2+x_1x_3+x_2x_3)t^2+x_1x_2x_3t$$

или

$$1+a_1t+a_2t^2+a_3t^3=\sum_{r=0}^{3}a_r(x_1,x_2,x_3)t^r,$$

где a_1, a_2, a_3- элементарные симметрические функции трех переменных x_1, x_2, x_3. Очевидно, что число слагаемых каждого коэффициента $a_r(r=1,2,3)$ равно $C(3,r)$. Следовательно, при $x_i=1(i=1,2,3)$ получим

$$(1+t)^3=\sum_{r=0}^{3}a_r(1,1,1)t^r=\sum_{r=0}^{3}C(3,r)t^r$$

Для n различных объектов $x_1, x_2, ..., x_n$ будем иметь

$$\prod_{r=1}^{n}(1+x_r t)=\sum_{r=0}^{n}a_r(x_1,...,x_n)t^r,$$

$$(1+t)^n=\sum_{r=0}^{n}a_r(1,...,1)t^r=\sum_{r=0}^{n}C(n,r)t^r. \qquad (13)$$

Функцию $f(t)=(1+t)^n$ называют (обычной) **производящей функцией сочетаний** из n различных объектов.

Придавая в (13) различные частные значения переменной t, получим некоторые интересные соотношения:

$$t=1:2^n=\sum_{r=0}^{n}C(n,r)=1+n+C(n,2)+C(n,3)+...+C(n,n),$$

$$t=-1:0=\sum_{r=0}^{n}(-1)^r C(n,r)=1-n+C(n,2)-C(n,3)+...+$$

$$+(-1)^n C(n,n).$$

Почленное сложение и вычитание этих равенств дает

$$\sum_{r=0}^{n}C(n,2r)=\sum_{r-0}^{n}C(n,2r+1)=2^{n-1},$$

а простая разбивка сомножителей

$$(1+t)^n=(1+t)^{n-m}(1+t)^m$$

и приравнивание коэффициентов при t^r приводят к равенству

$$C(n,r)=C(n-m,0)C(m,r)+C(n-m,1)C(m,r-1)+...$$

$$+C(n-m,r)C(m,0)$$

или

$$C(n,r)=\sum_{K=0}^{r}C(n-m,k)C(m,r-k).$$

Пример 1. Найти производящую функцию для r-сочетаний с ограниченным числом повторений из n элементов.

Здесь нельзя воспользоваться произведением биномов вида $1+x_k t$, как для r-сочетаний без повторений, так как всякий такой

бином отражает лишь две возможности: элемент x_k множества либо не появляется в r-сочетании, либо появляется в нем один раз. Пусть x_k появляется в r-сочетаниях с повторениями $0,1,2,...,j$ раз. Тогда ровно i появлениям элемента x_k соответствует одночлен $x_k^i t_k^i$, а по обобщенному правилу суммы появлениям элемента x_k "либо 0, либо 1,…, либо j раз" соответствует полином

$$1 + x_k t + x_k^2 t^2 + ... + x_k^j t^j$$

и значит производящая функция в этом случае имеет вид

$$F(t) = \prod_{K=1}^{n} (1 + x_K t + x_K^2 t^2 + ... + x_K^j t^j).$$

Если в этой задаче важен не вид производящей функции, а число соответствующих r-сочетаний, то принимаем $x_1 = x_2 = ... = x_n = 1$ и представляем производящую функцию в виде

$$(1 + t + t^2 + ... + t^j)^n = \sum_{r=0}^{n} a_r t^r.$$

Пример 2. Найти производящую функцию для r-сочетаний с неограниченным числом повторений из n-элементов.

Построим эту функцию на основе предыдущего примера при $j = 0,1,2,...$:

$$f(t) = (1 + t + t^2 + ...)^n = \left(\frac{1}{1-t}\right)^n = (1-n)^{-n} =$$

$$\sum_{r=0}^{\infty} C(-n, r)(-t)^r = \sum_{r=0}^{\infty} \frac{(-n)(-n-1)...(-n-r+1)}{r!}(-1)^r t^r =$$

$$\sum_{r=0}^{\infty} \frac{n(n+1)...(n+r-1)}{r!} t^r = \sum_{r=0}^{\infty} C(n+r-1, r) t^r,$$

откуда имеем последовательность $\{a_0, a_1 ...\}, a_r = C(n+r-1,2), r = 0,1,...$ чисел r-сочетаний с

повторениями из n элементов. Этот результат согласуется с полученным ранее.

Производящие функции для перестановок. По определению в случае n различных элементов число сочетаний равно

$$C(n,r)=\frac{P(n,r)}{r!}.$$

Поэтому из (13) следует, что

$$(1+t)^n=\sum_{r=0}^{n}P(n,r)\frac{t^r}{r!}, \qquad (14)$$

т.е. число перестановок $P(n,r)$ есть коэффициент при $\frac{t^r}{r!}$. Это и есть (экспоненциальная) производящая функция для числа r-перестановок.

При возможности неограниченного повторения элементов необходимо в левой части равенства (14) бином $1+t$ заменить на ряд вида

$$1+\frac{t}{1!}+\frac{t^2}{2!}+\frac{t^3}{3!}+\dots .$$

В случае, если соответствующий элемент допускает $K_1,K_2,\dots K_l$ повторений в выборках в выборках, бином следует заменить выражением

$$\frac{t^{K_1}}{K_1!}+\frac{t^{K_2}}{K_2!}+\dots+\frac{t^{Kl}}{K_l!}.$$

Представление соответствующего произведения в виде полинома по степеням t дает в качестве коэффициентов при $\frac{t^r}{r!}$ числа r-перестановок, допускающих указанные повторения.

Производящие функции для упорядоченных выборок называются **экспоненциальными** в силу того, что для r-перестановок с неограниченным числом повторений из n элементов производящая функция имеет вид степени ряда для функции e^t:

$$(1+\frac{t}{1!}+\frac{t^2}{2!}+...)^n = e^{nt} = \sum_{r=0}^{\infty} n^r \frac{e^r}{r!}.$$

При дополнительном условии, что каждый элемент входит в перестановку не менее одного раза, производящая функция примет вид

$$(\frac{t}{1!}+\frac{t^2}{2!}+...)^n = (e^t - 1)^n.$$

Понятия производящих функций двух видов для сочетаний и перестановок можно уточнить с помощью следующих определений.

1. **Обычной производящей функцией** для последовательности $a_0, a_1, ...$ называется сумма

$$A(t) = a_0 + a_1 t + a_2 t^2 + ... + a_n t^n + ...$$

2. **Экспоненциальной производящей функцией** для той же последовательности называется сумма

$$E(t) = a_0 + a_1 t + a_2 \frac{t^2}{2!} + ... + a_n \frac{t^n}{n!} + ...$$

В этих определениях элементы $a_0, a_1, ...$ должны быть упорядочены, но не обязательно различны. На переменную t никаких ограничений не накладывается.

Производящие функции были введены в комбинаторный анализ в качестве средства, позволяющего с большой общностью подходить к комбинаторным задачам перечисленного типа. Однако производящие функции, построенные для разных выборок, оказались в большинстве случаев довольно громоздкими. Для более удобной записи производящих функций имеется много средств. Это специальные операторы (сдвига, разности, усреднения), символические исчисления, специальные числа и специальные функции.

3.7 Комбинаторно-логический аппарат

Рассмотрим некоторые логические приемы, составляющие основу многих комбинаторных доказательств

3.7.1 Метод включений и исключений

Метод называют также методом решета, логическим методом, принципом перекрестной классификации, символическим методом.

Пусть дано n-множество S некоторых элементов и N - множество свойств $P_1, P_2, ..., P_N$ которыми элементы множества могут как обладать, так и не обладать. Требуется найти число элементов, не обладающее ни одним из перечисленных свойств.

Выделим какую-либо r-выборку свойств $(P_{i1}, P_{i2}, ..., P_{ir})$. Число элементов множества S, каждый из которых обладает всеми выбранными свойствами, обозначим через $n(P_{i1}, P_{i2}, ..., P_{ir})$. Отсутствие у элементов свойства P_i будем обозначать через $\overline{P_i}$. Таким образом число элементов, обладающих, например, свойствами P_1, P_3, P_5 и не обладающих свойствами P_2, P_4, P_6 запишется как $n(P_1, \overline{P_2}, P_3, \overline{P_4}, P_5, \overline{P_6})$.

Рассмотрим 2 простых случая:

а) имеется только одно свойство P; тогда

$$n(\overline{P}) = n - n(P),$$

б) имеется конечное число свойств $P_1, P_2, ..., P_N$, несовместимых друг с другом, тогда

$$n(\overline{P_1}, \overline{P_2}, ..., \overline{P_N}) = n - \sum_{i=1}^{N} n(P_{i)} .$$

В более общей постановке вопроса элементы множества могут обладать комбинацией совместимых свойств. В этом случае справедлива

Теорема 1. *Если даны n-множество элементов и N-множество свойств $P_i, i = 1,2,...,N$, то*

$$n(\overline{P}_1,\overline{P}_2,...,\overline{P}_N)=n-\sum_{i=1}^{N}n(P_i)+\sum_{1\le i<j\le N}n(P_i,P_j)-$$
$$-\sum_{1\le i<j\le N}n(P_i,P_j,P_k)+...+(-1)^N(P_1,P_2,...,P_N). \qquad (14)$$

Доказательство. Чтобы получить элементы, не обладающие ни одним из указанных свойств, необходимо из n-множества исключить элементы, обладающие свойством P_1, затем - P_2 и т.д. то есть $\sum_i n(P_i)$ элементов.

Однако при этом элементы, обладающие двумя свойствами, например, P_1 и P_2, оказались исключенными дважды (сначала – как элементы, обладающие P_1, затем – как обладающие свойством P_2). Следовательно, нужно возвратить элементы, обладающие двумя свойствами, т.е. добавить $\sum_{i,j}n(P_i,P_j)$. Но при этом оказались включенными элементы обладающие 3-мя свойствами, например, свойствами P_1,P_2,P_3. Рассуждая таким образом и далее, получим алгоритм для вычисления $n(\overline{P}_1,\overline{P}_2,...,\overline{P}_N)$, состоящий в попеременном отбрасывании и возвращении подмножеств. Отсюда и название метода.

Помимо этих простых рассуждений доказательство можно провести индукцией по N. Теорема справедлива для $N=1$:

$n(\overline{P})=n-n(P)$.

В соответствии с формулировкой теоремы мы можем записать также

$$n(\overline{P}_1,\overline{P}_2,...,\overline{P}_N)=n(\overline{P}_1,\overline{P}_2,...,\overline{P}_{N-1})-n(\overline{P}_1,\overline{P}_2,\overline{P}_{N-1},P_N).$$

Пусть теорема верна для $N-1$ свойств, т.е.

$$n(\overline{P}_1,\overline{P}_2,...,\overline{P}_{N-1})=n\sum_i n(P_i)+\sum_{i<j}n(P_i,P_j)-...+$$
$$+(-1)^{N-1}n(P_1,P_2,...,P_{N-1}).$$

Перейдем к ситуации, когда имеется N свойств и применим полученное соотношение для числа $n(\overline{P}_1, \overline{P}_2, ..., \overline{P}_{N-1}, P_N)$:

$$n(\overline{P}_1, \overline{P}_2, ..., \overline{P}_{N-1}, P_N) = n(P_N) - \sum_i n(P_i, P_N) + ... +$$

$$+ (-1)^N n(P_1, P_2, ..., P_N).$$

Вычитая это равенство из предыдущего, получим утверждение теоремы.

Характер доказательства таков, что его можно применить для любой комбинации свойств. В левой части доказанного равенства может стоять, например, $n(P_1, \overline{P}_2, P_3, \overline{P}_4)$. При этом теорема формируется относительно совокупности свойств P_2 и P_4 с обязательным выполнением свойств P_1 и P_3 следующим образом

$$n(P_1, P_3, \overline{P}_2, \overline{P}_4) = n(P_1, P_3) - n(P_1, P_3, P_2) - -n(P_1, P_3, P_4) +$$

$$+ n(P_1, P_2, P_3, P_4).$$

Дальнейшие усложнения метода связаны с введением весов элементов. Ограничений на понятие веса никаких не вводим. Для нас веса – это числовые характеристики элементов множеств, определяемые условиями задачи.

Пусть задано n-множество S и каждому элементу $S_i \in S$, $i = 1,2,...,n,$ приписан вес $V(Si).$ Из N - множества свойств $P_1, P_2, ..., P_N$ возьмем r - выборку $P_{i1}, ..., P_{ir}$ и обозначим сумму весов элементов, обладающих всеми выбранными свойствами, через $V(P_{i1}, P_{i2}, ..., P_{ir})$. А сумму весов, распространенную на все возможные r - выборки свойств, через

$$\sum V(P_{i1}, ..., P_{ir}) = V(r).$$

При $r = 0$ символ $V(0)$ есть сумма весов всех элементов множества S.

Тогда предыдущая теорема переформулируется следующим образом.

Теорема 2. *Если даны* n*-множество* S, *каждый элемент которого имеет вес, и* N *- множество свойств, то сумма* $V_N(0)$ *весов всех элементов, не обладающих ни одним из заданных свойств, определяется по формуле*

$$V_N(0) = V(0) - V(1) + V(2) - ... + (-1)^N V(N).$$

Теорема 2 обобщает теорему 1. Если все элементы $S_i \in S$ имеют единичный вес, то сумма весов равна числу слагаемых в сумме. В этом случае $V(0) = N$, а $V_N(0)$ равно числу элементов множества S, не обладающих ни одним из N свойств; при этом получим формулу (14).

Теорема 2 в свою очередь может быть обобщена.

Теорема 3. *Сумма весов элементов, обладающих в точности* r *-свойствами из свойств* $P_1, P_2, ..., P_N$, *находится по формуле*

$$V_N(r) = V(r) - C_{r+1}^1 V(r+1) + C_{r+2}^2 V(r+2) - ... +$$
$$+ (-1)^{N-r} C_N^{N-r} V(N),$$

или, что то же самое

$$V_N(r) = V(r) - C_{r+1}^r V(r+1) + C_{r+2}^r V(r+2) - ... +$$
$$+ (-1)^{N-r} C_N^r V(N).$$

Теоремы 2 и 3 доказываются аналогично теореме 1.

В качестве примера рассмотрим так называемую **задачу о беспорядках** (задачу о встречах). Пусть имеется конечное упорядоченное множество чисел $1,2,3,...,n.$ Для них могут быть образованы перестановки $a_1, a_2, ..., a_n$. Число всех перестановок равно $n!$. Среди этих перестановок есть такие, где ни один элемент не сохранил своего первоначального места: $a_i \neq i, i = 1,2,...,n$. Такие перестановки называются беспорядками. Сколько существует беспорядков?

Множество n элементов будем рассматривать по отношению к множеству свойств элементов оставаться на своем месте: $P_i \sim \{a_i = i \setminus i = 1,2,...,n\}$. Очевидно, что если s элементов за-

крепляются на своих местах, то число $N(s)$ соответствующих перестановок равно $(n-s)!$ Число беспорядков тогда находится с помощью метода включений и исключений:

$$N(0) = n! - C_n^1(n-1)! + C_n^2(n-2)! - \ldots + (-1)^S C_n^S (n-s)! + \ldots = n!(1 - 1 + \frac{1}{2!} - \frac{1}{3!} + \ldots + \frac{(-1)^n}{n!}),$$

что является целым числом, ближайшим к $n!e^{-1}$.

Если речь идет не о беспорядках, а о числе перестановок, в которых остаются на своих местах s элементов, то

$$N(s) - \frac{n!}{s!}(1 - 1 + \frac{1}{2!} - \frac{1}{3!} + \ldots + (-1)^{n-s}\frac{1}{(n-s)!}).$$

Ход рассуждений очевиден: из n элементов выбирают s неподвижных элементов C_n^S способами, а затем умножают (по правилу произведения) на число беспорядков среди оставшихся $(n-s)$ элементов.

3.7.2 Системы представителей множеств

Рассмотрим один из комбинаторных подходов к характеристике структуры конечных множеств. Основной идеей здесь является замена системы множеств собранием их представителей.

Постановка задач такого типа и методы их решения зависят от того, каким требованиям должны удовлетворяют эти представители.

Системы различных представителей

Пусть имеется n-множество S и множество $P(S)$ всех его подмножеств. Пусть $M = (S_1, S_2, \ldots, S_m)$ - некоторая m выборка из $P(S)$, а $a = (a_1, a_2, \ldots, a_m)$ - некоторая m выборка из S.

Если выборке M можно сопоставить (не обязательно однозначно) выборку a такую, что элементы $a_i, i = 1,2,\ldots,m,$ попарно различны и при этом $a_i \in S_i, i = 12,\ldots,m,$ то говорят, что элемент a_i представляет множество S_i, а вся выборка $(a_1, a_2, \ldots, a_m)$ на-

зывается системой различных представителей (СРП) для M . Заметим, что если $i \neq j$, то $a_i \neq a_j$ даже если $S_i = S_j$. Если множество появляется несколько раз, то всякий раз оно должно иметь представителя отличного от других.

Оказывается, СРП может существовать не для всяких совокупностей множеств. Если в конечной системе множества не пусты и не пересекаются, то СРП, очевидно, существует.

Возьмем более сложный случай. Например, $S = (a,b,c,d,e)$, а M есть совокупность 4-х множеств:

$S_1 = (a,b,c,d); S_2 = (a,b,e); S_3 = S_4 = (b,e).$ Существует две СРП: (c,a,b,e) и (d,a,e,b). Но стоит изменить лишь одно из подмножеств, например, вместо S_2 взять $S_2^{'} = (b,e)$, то мы уже не сможем получить ни одной СРП.

На вопрос о существовании СРП для заданного семейства множеств, отвечает теорема Ф.Холла (1935 г.).

Теорема Ф.Холла. *Подмножества* $S_1, S_2, ..., S_m$ *имеют СРП тогда и только тогда, когда объединение любых* k *из этих множеств содержит не менее* k *элементов. Иначе говоря, СРП для* $S_1, S_2, ..., S_m$ *существует тогда и только тогда, когда* $S_{i1} \cup S_{i2} \cup ... \cup S_{iK}$ *состоит не менее чем из* k *элементов. При этом* $k = 1,2,...,m, a(i_1,...,i_k)$ *- любая* k *-выборка из* $1,2,...,m.$

Доказательство: Необходимость почти очевидна, т.к. существование СРП обеспечивает необходимое количество элементов в качестве различных представителей.

Для доказательства достаточности приведем другую формулировку, дающую также нижнюю грань для числа самих СРП.

Теорема. *Пусть семейство* $M = (S_1, S_2, ..., S_m)$ *удовлетворяет необходимому условию существования СРП и пусть каждое из множеств состоит не менее чем из* t *элементов. Тогда:*

а) если $t \leq m,$ *то* M *имеет не менее чем* $t!$ *СРП;*

б) если $t > m,$ то M имеет не менее чем $\frac{t!}{(t-m)!}$ СРП.

Доказательство проведем по индукции относительно m. Для $m = 1$ (и даже для $m = 2$) теорема очевидна. Докажем, что она верна и для любого конечного m, исходя из ее справедливости для $m' < m.$

Рассмотрим объединение некоторой k -выборки множеств:

$S_{i1} \cup S_{i2} \cup ... \cup S_{iK}$

Возможны два случая:

- число элементов в объединении = k ;
- число элементов > k .

Начнем **со второго случая**. Здесь рассматриваемое объединение содержит не менее $k+1$ элементов, каковы бы ни были k: $k = 1,2,...,m,$ и набор $i_1, i_2, ..., i_k$ из чисел $1,2,...,m.$

Выберем какой-либо из элементов $a_1 \in S_1$ и удалим его из $S_2, S_3, ..., S_m,$ если он там встречается. Придем таким образом к $M^* = (S_2^*, S_3^*, ..., S_m^*).$ Эта $(m-1)-$выборка удовлетворяет необходимому условию существования СРП, т.к. $S_{i1}^* \cup S_{i2}^* \cup ... \cup S_{ik}^*$ содержит не менее k элементов и, кроме того, M^* имеет не менее $(t-1)!$ СРП, если $t \le m$ (и значит, $t-1 \le m-1$), либо не менее $(t-1)!/(t-m)!$ СРП, если $t > m$ (и значит, $t-1 > m-1).$ Искомый результат достигается, если учесть, что в S_1 имеется по меньшей мере t возможностей фиксировать элемент a_1 и что этот элемент вместе с СРП для M^* составляет СРП для M .

Теперь вернемся к **первому случаю**. Здесь проведенное доказательство применить нельзя, т.к. существует некоторая k -выборка $S_{i1}, S_{i2}, ..., S_{ik}$ такая, что $S_{i1} \cup S_{i2} \cup ... \cup S_{ik}$ содержит точно k элементов $(1 \le k \le m-1).$ Перенумеруем множества $S_1, ..., S_m$ следующим образом

$$S_{i1}, S_{i2,} ..., S_{ik}$$
$$\downarrow \quad \downarrow \quad \downarrow$$
$$S_1, S_2, ..., S_k, S_{k+1}, ..., S_m.$$

Поскольку $S_1 \cup S_2 \cup ... \cup S_k$ содержит точно k элементов, то $t \le k.$ Следовательно, по предположению индукции $(S_1, S_2, ..., S_k)$ имеет не менее чем $t!$ СРП.

Рассмотрим одну из этих СРП: $(a_1, a_2, ..., a_k)$, где $a_i \in S_i, i = 1,2,..., k.$ Удалим элементы $a_1, ..., a_n$ из $S_{k+1}, ..., S_k$, если они там встретятся. Получившаяся $(m-k)$-выборка $M* = (S^*_{k+1}, ..., S^*_m)$ удовлетворяет необходимому условию существования СРП.

Действительно, если это не так и если объединение какой-либо k *-выборки $S^*_{k+1} \cup ... S^*_{k+k^*}$ имеет меньше k * элементов, то объединение

$$S_1 \cup S_2 \cup ... \cup S_k \cup ... \cup S_{k+k*}$$

имело бы менее чем $k + k$ * элементов, что противоречит условию теоремы. Таким образом M * имеет хотя бы одну СРП и, следовательно, M имеет не меньше чем $t!$ СРП.

Алгоритм выбора СРП. На практике бывает трудно проверить, выполняются ли в конкретном случае условия теоремы Холла. Кроме того, приведенное доказательство не дает указаний, как найти СРП. (Это характерно, так как теоремы существования обычно и появляются там, где трудно найти алгоритм решения).

Алгоритм, который позволяет для данного конечного набора множеств подобрать СРП или показать что ее не существует, привел М.Холл в качестве доказательства теоремы Ф.Холла о различных представителях.

Пусть задано n множеств: $S_1, S_2, ..., S_n.$ Требуется найти для них СРП или показать, что такой системы не существует. Произвольным образом выберем элемент первого множества $a_1 \in S_1$ в качестве его представителя. Поочередно будем выбирать

$a_2 \in S_2, a_3 \in S_3, \ldots,$ заботясь только о том, чтобы они были различными. Если доведем процесс до $a_n \in S_n$ включительно, то получим искомую СРП.

Возможно, что на r-м шаге мы дойдем до некоторого t-множества S_r, все элементы которого $b_1, b_2, \ldots, b_t$ уже выбраны представителями других множеств. Но это еще не означает, что СРП не существует. Будем брать поочередно все те множества, представителями которых являются $b_1, b_2, \ldots, b_t$, удалять из них все элементы этой последовательности, а оставшиеся приписывать в ее конец. Так будем поступать до тех пор, пока не случится одно из двух:

1) мы достигнем элемента b_i, который не может служить представителем;

2) последовательность исчерпывается элементами $b_1, b_2, \ldots, b_s$ как представителями множеств.

В случае 2) СРП не существует. В самом деле, элементы $b_1, b_2, \ldots, b_s$ являются представителями s множеств и по построению каждый элемент этих s множеств содержится в данной последовательности. Но тогда эти s множеств, а также множество S_r образуют $s+1$ множеств, которые содержат только s различных элементов, что противоречит условию теоремы.

Если же имеет место 1) то на некотором этапе мы находим элемент $b_i = b_{i1} \in S_{j1} (i_1 > t),$ не являющийся до сих пор представителем. Это означает, что представителем S_{j1} уже был выбран другой элемент $b_{i2} (i_2 < i_1).$ Если $i_2 > t$, то значит $b_{i2} \in S_{j2}$, представителем которого уже выбран $b_{i3} (i_3 < i_2)$ и т.д. Таким образом, возникает последовательность $b_{i1}, b_{i2,\ldots,} b_{im}$, индексы которой убывают $(i_m \le t)$, причем в этой последовательности каждый ее член входит в множество, представителем которого является следующий член. Заменяем представителей, выбирая элементы: b_{i1} для

S_{j1}, b_{i2} для $S_{j2},...,b_{im-1}$ для S_{jm-1}. Элемент b_{im} в результате этой замены освобождается для выбора в качестве представителя S_r. Итак, $S_1,...S_r$ имеют различных представителей и мы можем следовать тем же путем, чтобы либо дойти до S_n и получить полную СРП, либо встретить случай 2) и установить несуществование СРП.

Заключение о числе СРП получается из приведенного алгоритма как следствие. Действительно, если СРП существует, то существует и некоторое множество, каждый элемент которого может быть выбран в качестве его представителя в СРП. Следовательно, если множества данного семейства n элементов имеют t или более элементов, то существует по меньшей мере $t!$ СРП, если $t < n$ и $t(t-1)...(t-n+1)$, если $t \geq n$. Выбор первого представителя может быть осуществлен по крайней мере t способами; вычеркнув этот элемент из всех других множеств, получим систему множеств в $S_2^*,...,S_m^*$, наименьшее из которых содержит не менее $t-1$ элементов. Продолжая процесс далее, получим указанный результат.

Замечание. Условие конечности важно, так как для бесконечной системы множеств, например,

$S_0 = \{1,2,3,...,k,...\};$

$S_1 = \{1\};$

……

$S_k = \{k\};$

……

нет СРП, хотя она есть для любой ее части.

Существуют и другие системы представителей множеств. Задачи о разбиении множеств привели к понятию системы общих представителей множеств. Пусть даны два различных разбиения одного и того же множества S на k непустых составляющих:

$$S = A_1 \cup A_2 \cup ... \cup A_k; S = B_1 \cup B_2 \cup ... \cup B_k$$

Если существует подмножество O множества S, состоящее из k элементов и такое, что его пересечение с любым из состав-

ляющих не пусто:

$$O \cap A_i \neq \varnothing;\ O \cap B_i \neq \varnothing,\ i = 1,2,...,k,$$

то оно называется **системой общих представителей** (СОП) данных разбиений. При этом каждое пересечение оказывается состоящим только из одного элемента.

При необходимости, попарно взятые множества первого и второго разбиений можно подобрать так, чтобы они имели только один общий элемент, являющийся их общим представителем. Можно ставить и другие условия: существование заданного числа общих элементов, заданного множества их и т.д.

Понятия о представителях множеств и о системах представителей находит в математике многочисленные и разнообразные приложения, например, это представители классов эквивалентности. Метод систем представителей встречается также в теории сетей при исследовании допустимости потоков и в теории расширения латинских квадратов.

3.8 Общая модель и основные способы описания задач комбинаторного программирования

Задачи комбинаторного программирования, или оптимизационные задачи комбинаторного анализа, весьма разнообразны не только по приложениям, но и по используемым средствам математического описания. При формализации задач используются самые разнородные математические объекты: перестановки, графы, разбиения, целочисленные векторы. Тем не менее, представляется полезным дать общую формулировку задач комбинаторного программирования. Такая модель, во-первых, дает стандартный способ описания задач соответствующего класса, а во-вторых, предоставляет возможность разработки общих методов решения таких задач.

Постановка общей задачи комбинаторного программирования может быть более или менее детализированной. Приведем наименее «подробную» постановку.

Модель 1. На конечном множестве N -мерного пространства X_0 максимизировать функцию $f_0(x)$ при ограничении

$f_i(x) \le 0, i \in I_M$,

где $f_i(x)(i \in I_M = \{0,1,...,M\})$ - заданные действительные функции N переменных $(x_1, x_2,..., x_n)$, составляющих вектор x.

Эта модель пригодна для рассмотрения многих задач комбинаторного программирования, но слабо отражает их специфику. Прежде чем перейти к более детализированной модели, выделим **основные элементы задач** комбинаторного программирования.

Исходные объекты. В любой задаче комбинаторного программирования присутствуют некоторые объекты (элементы), которые нужно упорядочить, разместить, разбить на группы (задания и детали – в задаче о заданиях и задаче о деталях; задания, распределяемые по исполнителям – в задаче распределения и т.д.).

Множество возможных решений. В задачах комбинаторного программирования обычно ищется наилучшая в некотором смысле выборка определенного типа из исходных объектов. Обычно число таких выборок сравнительно велико. Множество Λ_0 всех таких выборок (элемент его – λ) и называется множеством возможных решений. В рассмотренных в п.3.1 задачах в качестве Λ_0 и λ выступали: перестановка π из элементов множества I_N и множество Π_N всех таких перестановок (задача о заданиях, задача о деталях, задача размещения); разбиение $r = (r_1, r_2,...r_M)$ множества I_M на M подмножеств (задача распределения) и т.д.

Исходный массив. Исходные объекты характеризуются некоторыми численными данными, которые для объектов и их комбинаций образуют исходный массив A. Так, в задаче о заданиях исходный массив A составляют: время выполнения заданий $\tau(j)$, директивные сроки $T(j)$, штрафы $a(j)$. В задаче о деталях массив A – это время обработки $\tau(i, j)$ и т.д. Исходный массив может иметь любую структуру и состоять из подмассивов любой мерности.

Целевая функция и ограничения. Значения целевой функции f_0 и функций $f_i(i \in I_M)$, задающих ограничения, однозначно

определяются возможными решениями λ. Поэтому **модель 1** можно переформулировать, заменив множество X_0 на Λ_0, а зависимость $f_i(x)$ – на $f_i(\lambda)(i \in I_M)$. Для большей детализации раскроем характер зависимости $f_i(\lambda)$. Аргументом функции f_i является некоторый массив $X_i \subset A$. Каким образом и какие именно элементы исходного массива A заполняют массив X_i, определяется возможным решением λ.

Будем задавать заполнение массивов X_i элементами исходного массива A с помощью выборки $\beta_i(\lambda)$ (эта выборка содержит коды элементов массива A, совпадающие с кодами соответствующих элементов массива X_i при данном λ). Обозначим через $A_{\beta i(\lambda)}$ массив, получающийся заменой в выборке $\beta_i(\lambda)$ кодов элементов массива A самими элементами, тогда для каждого λ

$$X_i = A_{\beta i(\lambda)},\ f_i(X_i) = f_i[A_{\beta i(\lambda)}], i \in I_M.$$

Рассмотрим в качестве примера задания выборок $\beta_i(\lambda)$ **одну из задач распределения заданий.** Здесь $t(i, j)$ - время, затрачиваемое i-м исполнителем на выполнение j-го задания.

Предположим, что используется ресурс только одного вида $(K = 1)$, причем он также имеет смысл затрачиваемого времени $S_2(i,1)$ (величины $S_1(i,1)$ равны нулю). Эта частная задача при критерии минимального суммарного затраченного времени формулируется следующим образом.

На множестве $R(N, M)$ минимизировать функцию

$$F_1(r) = \sum_{i \in I_M} \sum_{j \in r_i} t(i, j)$$

при ограничениях

$$\sum_{j \in r_i} t(i, j) \le S_2(i,1), i \in I_M.$$

Исходный массив A составляют величины $t(i, j)$ и $S_2(i,1)$. Поэтому A есть матрица размерности $M \times (N+2)$, первые N столбцов которой совпадают с матрицей $[t(i, j)], (N+1)$-й столбец в i-й строке содержит элемент $S_2(i,1), (N+2)$-й столбец введен для удобства и состоит из нулей.

Массив X_0 будем считать одномерным массивом длины N с элементами x_{0j}. Массивы $X_i (i \in I_M)$ примем одномерными массивами длины $N+1$ с элементами x_{ij}. Соответствующий вид имеют и выборки $\beta i(r)$ (возможными решениями в рассматриваемой задаче являются разбиения r). Элементы $\beta_{ik}(r)$ этих выборок равны:

$$\beta_{ok}(r) = (i_k k), k \in I_M,$$

$$\beta_{ik}(r) = \begin{cases} (i_k k) \quad если \quad i = i_k, \quad k \le N, \\ (i, N+2) \, если \quad i \ne i_k, \quad k \le N, \\ (i, N+1) \, если \, k = N+1, \end{cases}$$

где $i \in I_M, k \in I_{N+1}, i_k$ - номер того подмножества r_i в разбиении r, которое заключает элемент k. При заданной структуре и составе массива A, структуре массивов X_i и выборок $\beta i(r)$ функции $fi(X_i)$ имеют вид

$$f_0(X_0) = \sum_{i \in I_M} x_{oi}, \; f_i(X_i) = \sum_{j \in I_N} x_{ij} - x_{i,N+1}.$$

Теперь сформулируем общую задачу комбинаторного программирования, учитывая характер зависимостей $f_i(\lambda)$.

Модель 2. На множестве Λ_0 максимизировать функцию $f_0[A_{\beta 0(\lambda)}]$ при ограничениях $f_i[A_{\beta i(\lambda)}] \le 0, i \in I_M$.

Модели 1 и 2 имеют сходную структуру, но модель 2 лучше отражает специфику комбинаторных задач. Она, кроме того, ис-

пользует исходный массив A. Это важно, так как «внутренние» свойства этого массива существенно влияют на сложность решения задачи.

Модель 2 описана в терминах **выборок ограниченной длины** из конечного множества элементов. Выборки наряду с чисто комбинаторным (перестановки, сочетания, размещения и т.д.) могут иметь и другие представления. Основные из них используют терминологию **теории графов и булевой алгебры**.

Каждое из таких описаний имеет свои преимущества: комбинаторное ближе к практической постановке задачи; графовое обладает большей наглядностью, булевое является универсальным (любая задача комбинаторного программирования может быть описана в булевых переменных).

Рассмотрим возможности различных представлений на примере **задачи о назначениях**. Ранее мы дали чисто **комбинаторное описание** этой задачи. **На множестве** Π_N **минимизировать функцию**

$$F_3(\pi) = \sum_{j \in I_M} t(\pi_j, j).$$

Дадим **графовое описание** задачи. Для этого нам потребуются некоторые дополнительные понятия теории графов.

Граф $G(X)$ называется **двудольным**, если множество X его вершин можно разбить на два таких подмножества X' и X'', что для имеющихся в графе дуг все начальные вершины входят в X', а все конечные – в X''. Двудольный граф называется **полным**, если любая вершина $x' \in X'$ соединена дугой с любой вершиной $x'' \in X''$. $G(N', N'')$ - полный двудольный граф, в котором N' - количество вершин в X' и N'' - количество вершин в X''.

В графе $G(X)$ n**-паросочетанием** называется совокупность из n несмежных дуг. В графе на рис. 3.2 имеется 6 различных 3 – паросочетаний:

$$(l_1, l_5, l_9), (l_1, l_6, l_8), (l_2, l_4, l_9), (l_2, l_6, l_7), (l_3, l_4, l_8),$$
$$(l_3, l_5, l_7).$$

Пусть граф G является взвешенным по дугам. Тогда вес лю-

бого паросочетания равен сумме весов дуг, входящих в него. С учетом этих определений можно дать следующую постановку задачи в терминах теории графов.

Во взвешенном двудольном графе $G(N,N)$, дугам которого $(x_i^{'}, x_j^{''})$ приписаны веса $c(i,j)$, найти N - паросочетание максимального (минимального) веса.

Если в графе $G(3,3)$ (рис. 3.2) задать веса дуг матрицей

$$C = \begin{bmatrix} 1 & 3 & 2 \\ 4 & 2 & 1 \\ 3 & 1 & 3 \end{bmatrix},$$

то максимальным в нем будет паросочетание (l_2, l_4, l_9). Можно показать, что **между N - паросочетаниями графа $G(N,N)$ и перестановками π из Π_N существует взаимооднозначное соответствие**. Оно осуществляется следующим образом:

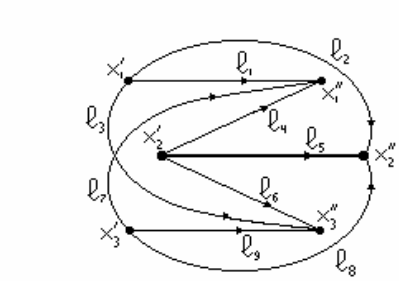

Рис. 3.2 – Пример двудольного графа

если в паросочетание входит дуга $(x_i^{'}, x_j^{''})$, то в соответствующей перестановке $\pi j = i$. При таком соответствии значение функции $F_3(\pi)$ и вес соответствующего паросочетания совпадают.

Рассмотрим **булевое описание** задачи о назначениях. Поставим в соответствие каждой перестановке $\pi \in \Pi_N$ набор из N^2

булевых переменных y_{ij}. Соответствие заключается в следующем:

$$y_{ij} = \begin{cases} 1 & при \quad i = \pi j, \\ 0 & в \quad противном \quad случае. \end{cases}$$

При фиксированном $i = i'$ ровно одна из переменных $y_{i',j}$ отлична от нуля, а при фиксированном $j = j'$ ровно одна из переменных $y_{ij'}$ отлична от нуля, т.е. наборы булевых переменных y_{ij}, соответствующие перестановкам π из Π_N, должны удовлетворять соотношениям:

$$\sum_{i \in I_N} y_{ij} = 1, \sum_{j \in I_N} y_{ij} = 1. \qquad (15)$$

Справедливо и обратное. По любому решению в булевых переменных системы (15) можно построить соответствующую ему перестановку. Например, для $N = 3$ набору переменных y_{ij}.

$$[y_{ij}] = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

соответствует перестановка $\pi' = (2,1,3)$.

Целевая функция задачи назначения для любого набора булевых переменных $[y_{ij}]$, удовлетворяющих соотношениям (15), может быть представлена так:

$$F_1([y_{ij}]) = \sum_{i \in I_N} \sum_{j \in I_N} c(i, j) y_{ij}. \qquad (16)$$

Таким образом, получим следующее булевое описание задачи о назначениях:

На множестве наборов булевых переменных $[y_{ij}]$ **максимизировать (минимизировать) функцию** (16) **при ограничениях** (15)**.**

Мы рассмотрели различные способы описания задач комби-

наторного программирования. Возможность использования описаний различного типа – характерное свойство задач комбинаторного программирования. При этом в общем случае ни один способ описания не имеет явного преимущества перед другими. Выбор способа определяется удобством представления и исследования задачи. В свою очередь, способ описания существенно влияет на выбор метода решения задачи комбинаторного программирования.

Из представленных способов описания задач комбинаторного программирования важными особенностями выделяется булевое описание. **Первой** особенностью является сравнительная простота его получения для большинства задач комбинаторного программирования независимо от их исходной формулировки. **Вторая** особенность состоит в том, что любая задача в булевых переменных может быть приведена к линейному виду, что позволяет использовать для ее решения методы линейного программирования.

Рассмотрим такое преобразование. Пусть имеется задача: на множестве B_N всех булевых векторов $y = (y_1, y_2, ..., y_N)$ максимизировать функцию $f_0(y)$ при ограничениях $f_i(y) \le 0 (i \in I_M)$. Предлагаемое преобразование содержит два перехода:

1) функции $f_i(y) (i \in I_M)$ заменяются полиномами (кстати, такие функции от булевых переменных, которые принимают произвольные действительные значения, называются псевдобулевыми);

2) каждое произведение булевых переменных заменяется новой булевой переменной и двумя ограниченными.

В основе первого перехода лежит возможность представления любой псевдобулевой функции $\varphi(y_1, y_2, ..., y_m)$ в виде

$$\varphi(y_1, y_2, ..., y_m) = \varphi(0, y_2, y_3, ..., y_m) +$$
$$y_1[\varphi(1, y_2, y_3, ..., y_m) - \varphi(0, y_2, y_3, ..., y_m)].$$

Последовательное использование соотношения позволяет любую функцию φ представить в виде полинома.

Второй переход состоит в линеаризации этого полинома. Для примера рассмотрим произведение первых k переменных:

$y_1 \cdot y_2 \cdots y_k$. Введем новую переменную

$$y_0 = y_1 \cdot y_2 \cdots y_k \qquad (17)$$

Очевидно, что y_0 - булевая переменная. Представление (17) для нее может быть заменено двумя линейными неравенствами

$$y_1 + y_2 + ... + y_k - ky_0 \geq 0,$$

$$y_1 + y_2 + ... + y_k - y_0 \leq k - 1.$$

Первое из них обеспечивает равенство y_0 нулю, если хотя бы одна из переменных $y_i (i = 1,2,...,k)$ равна 0. Второе обеспечивает равенство $y_0 = 1$ при всех $y_i = 1 (i = 1,...,k)$.

Таким образом, произведение булевых переменных всегда может быть заменено новой переменной и двумя линейными ограничениями. Такое представление позволяет использовать для решений задачи методы линейного программирования, т.к. задача нахождения экстремума линейной формы при линейных ограничениях и есть задача линейного программирования (ЛП). Многие экстремальные задачи комбинаторного программирования сводятся к задачам линейного программирования, отличаясь лишь дополнительным условием целочисленности аргументов. Среди них важным подклассом являются задачи булевого линейного программирования (только что рассмотренные).

Можно сказать, что задача ЛП представляет собой типичный пример комбинаторной задачи (даже без добавления условия целочисленности переменных, т.к. оптимальным решением задачи ЛП является одна из вершин многогранника, определяемого системой ограничений (неравенств), а любая вершина однозначно определяется как сочетание из $N + M$ элементов по N (N – число переменных, M – число ограничений).

Тем не менее, вряд ли целесообразно включать ЛП в комбинаторное, поскольку первое представляет собой специфическую и хорошо разработанную ветвь математического программирования.

3.9 Методы решения экстремальных задач комбинаторного программирования

К настоящему времени разработано большое число алгоритмов решения экстремальных задач комбинаторного программирования. Наиболее перспективные из них выделились в отдельную область комбинаторного программирования. Общая идея этих методов состоит в замене полного перебора всех вариантов частичными переборами меньших объемов. При реализации этой идеи отыскивают подмножества, заведомо не содержащие искомого экстремума и сужают область перспективных вариантов.

Зачем это делается? Дело в том, что несмотря на конечное число возможных решений, полный их перебор и др. более тонкие точные методы дают для задач реальной размерности (например, число исходных объектов) такие объемы вычислений, что требуемые затраты делают эти методы практически нереальными. Это явление в комбинаторном программировании называется «проклятием размерности» (современный американский математик Р.Беллман).

Первая группа – точные методы. Среди них можно выделить две разновидности: переборные и неперебoрные методы. К первой разновидности относятся методы, которые не гарантируют, что не придется «просматривать» все возможные решения.

Переборные методы:

1) метод разбиения и анализа (обобщение широко известного метода ветвей границ);

2) метод отсечений;

3) метод полного перебора.

Ко второй разновидности относятся методы, для которых существует гарантия того, что будет просмотрена только часть возможных решений. Выделение «просматриваемого» подмножества возможных решений требует определенных, иногда достаточно больших вычислительных затрат. Поэтому неперебoрные методы не всегда более экономичны, чем переборные.

Неперебoрные методы:

1) метод динамического программирования;

2) перестановочный метод.

Охарактеризуем кратко упомянутые методы.

Метод разбиения и анализа. Состоит из двух процедур: про-

цедуры разбиения множества возможных решений X_0 на подмножества и процедуры анализа подмножетв возможных решений с целью исключения некоторых из них. Метод является итерационным. На каждой итерации выполняются обе процедуры. При этом **разбиению** подвергается одно из подмножеств возможных решений, полученных и не исключенных на предыдуих итерациях. Процедура **анализа** нацелена на то, чтобы проверить отсутствие в расссматриваемомм подмножестве допустимого решения, лучшего по значению целевой функции. Если это удается сделать, то такое подмножество отбрасывается (исключается из дальнейшего рассмотрения).

Пользуясь терминологией теории графов (в данном случае это наиболее удобно), можно представить метод как итерационную процедуру построения дерева, вершины которого соответствуют отдельным подмножествам озможных решений. Перед началом первой итерации имеем одну вершину – X_0 (корневая). Процесс заканчивается, когда из дерева ветвлений исключается корневая вершина.

Каждый этап построения дерева имеет свои критерии (правила):

1) при выборе вершины для очередного ветвления;
2) при выборе способа разбиения очередной вершины;
3) при анализе соответствующей вершины.

Метод разбиения и анализа является наиболее общим методом, применимым для решения любой задачи комбинаторного программирования. При этом эффективность метода тем выше, чем более учтена при его реализации специфика решаемой задачи.

Метод отсечений. Метод состоит в замене исходной задачи последовательностью вспомогательных задач с той же целевой функцией, но с более широкими множествами допустимых решений. Последовательность задач строится до тех пор, пока на некотором шаге построения не окажется, что решение построений на этом шаге задачи является допустимым решением исходной. Это решение, очевидно и будет искомым оптимальным решением. В методе предполагается, что вспомогательные задачи являются существенно более простыми, чем исходная.

На k-м шаге метода решается k-я задача: определить на множестве X_1^k подмножество X_2^k векторов x^k, доставляющих функ-

ции $f_0(x)$ (целевой функции исходной задачи) экстремальное (максимальное) значение.

Таким образом метод отсечений определяется характером последовательности множеств $\{X_1^k\}$. Обязательно должны выполняться условия (для всех $k = 1,2,\ldots$), определяющие идею метода:

$$X_1^k \supset X_1, \quad X_1^k \supset \quad X_1^{k+1}, \quad X_2^{k+1} \not\subset X_1^k$$

В "отсекаемую" часть множества X_1^k всегда попадает множество X_2^k оптимальных решений k-го шага

Основной способ построения $\{X_1^k\}$.

Формируется некоторое ограничение (свое на каждом k-м шаге) в виде неравенства $\varphi_k(x) \le 0,$ которому удовлетворяют все $x \in X_1^k$, но не удовлетворяют $x \in X_2^k$. Такое ограничение называется правильным отсечением. С помощью него и формируют множество X_1^{k+1} для $(k+1)$-го шага:

$$X_1^{k+1} = \{x / x \in X_1^k, \quad \varphi_k(x) \le 0\}.$$

Метод полного перебора. По сравнению с предыдущими существенно прост и применим по крайней мере в двух областях комбинаторного программирования:

1) задачи сравнительно малой размерности;
2) "изощренные задачи" со сложным заданием целевой функции и (или) множества допустимых решений.

При реализации метода последовательно просматриваются все возможные решения задачи. Просмотр состоит в проверке допустимости данного решения и (при положительном ответе) – в подсчете для этого решения значения целевой функции. Эффективность метода зависит от объема промежуточной информации, подлежащей запоминанию, и от близости "соседних" возможных решений.

Метод динамического программирования

Идея метода. Решение одной задачи с заданным числом исходных объектов заменяется при решением целого семейства вспомогательных задач, в которых число исходных объектов меняется от

минимально возможного до заданного в основной задаче. Вспомогательные задачи имеют ту же целевую функцию и ту же (либо близкую) структуру множеств возможных и допустимых решений, что у основной задачи. Метод эффективен тогда, когда при решении одних вспомогательных задач удается существенно использовать оптимальные решения других. Таким образом метод состоит в последовательном решении некоторой совокупности задач, близких по структуре к основной, с использованием получаемых в ходе процедуры оптимальных решений. Заканчивается процедура получением оптимальных решений вспомогательных задач, по которым достаточно эффективно строится оптимальное решение исходной задачи. Такая замена процесса решения одной задачи процессом последовательного решения некоторой совокупности вспомогательных задач и определила название метода. Основоположник – Р.Беллман.

Перестановочный метод. Выбирается некоторый оператор, действующий на множества допустимых решений, который может быть применим к одним допустимым решениям и неприменим к другим. (Например, в перестановке меняет местами соседние элементы, для которых соответствующие исходные данные связаны определенным соотношением. Если в некоторой перестановке окажется, что для всех соседних элементов указанное соотношение не выполняется, то оператор к этой перестановке не применим). Оператор должен удовлетворять условию: допустимое решение к которому оператор применим, должно переводиться им в новое допустимое решение, не уступающее по значению целевой функции старому. Очевидно, когда такой оператор выбран, оптимальное решение следует искать среди неподвижных для него допустимых решений (тех к которым он неприменим). Метод эффективен, когда число таких неподвижных решений невелико. В частности, если все неподвижные допустимые решения доставляют целевой функции одинаковые значения, то любое из них является оптимальным решением. Во всех случаях, используя выбранный оператор, можно из любого допустимого решения получить неподвижное допустимое (многократным использованием оператора).

Рассмотрим применение метода к задаче о деталях (частный случай обработки N деталей на $M = 2$ станках). Смысл задачи состоит в следующем. Допустимые решения могут быть представлены

в виде перестановок π из Π_N, задающих порядок обработки деталей. Можно показать, что перестановка π', полученная из любой перестановки π с помощью оператора

$$g(j)\pi = g(j)(\pi_1,...,\pi_{j-1},\pi_j,\pi_{j+1},...\pi_N) = \\ = (\pi_1,...,\pi_{j-1},\pi_{j+1},\pi_j,...,\pi_N) \quad (18)$$

(перестановка j-го и $j+1$-го элементов) не уступает по значению целевой функции перестановке π при следующем условии:

$$\min\{\tau(1,\pi_j),\ \tau(2,\pi_{j+1})\} > \min\{\tau(1,\pi_{j+1}),\ \tau(2,\pi_j)\}. \quad (19)$$

Выберем в качестве оператора перестановочного метода оператор g_1, который находит и меняет местами в перестановке π любую пару "соседних" элементов, для которых выполняется условие (19). Легко проверить, что при отсутствии в наборе $\{\tau(i,j)\}$ равных величин оператор g_1 имеет единственную неподвижную перестановку (т.е. такую, к которой оператор неприменим) $\pi^{(1)}$, определяемую следующим образом.

Пусть $\tau(i_1,j_1)$ – минимальный по величине элемент набора $\{\tau(i,j)\}$. Если $i_1 = 1$, то $\pi_1^{(1)} = j_1$, в противном случае $(i_2 = 2)\pi_N^{(1)} = j_1$.

Затем элемент $\tau(i_1,j_1)$ удаляется из набора $\{\tau(i,j)\}$, а в оставшейся его части снова находится минимальный по величине элемент $\tau(i_2,j_2)$. Если оказывается, что $i_2 = 1$, то элемент j_2 помещается в незанятую на предыдущем шаге позицию перестановки $\pi^{(1)}$ с минимальным номером. В противном случае $(i_2 = 2)$ этот элемент помещается в незанятой позиции $\pi^{(1)}$ с максимальным номером. Дальнейшеее заполнение перестановки $\pi^{(1)}$ элементами I_N осуществляется аналогично. Можно проверить, что к полученной перестановке $\pi^{(1)}$ оператор g_1 неприменим. Построим такую

перестановку, используя данные к этой задаче (см.таблицу 3.2). В примере на некоторых шагах в числе минимальных оказывается несколько элементов из совокупности π^1: в таких случаях выбирается любой из них. Заметим, что если бы в данной задаче среди элементов набора $\{\tau(i, j)\}$ были равные по величине (как в табл.3.2), то оператор g_1 имел бы не одну, а несколько неподвижных перестановок. Легко проверить, что все эти перестановки эквивалентны с точки зрения критерия задачи.

И последнее замечание. Представленное решение задачи о деталях (для 2-х станков) было плучено современным американским математиком Р.Джонсоном. Решение оказалось настолько изящным, что за данной задачей прочно укрепилось название – **задача Джонсона**.

Вторая группа – приближенные методы. Рассмотренные точные методы решения в применении ко многим практическим задачам могут быть сложны в реализации из-за ограниченности вычислительных ресурсов.

Кроме того, большие затраты на получение точного решения не всегда оправданы, так как исходные данные для задачи всегда получены с некоторой погрешностью. В связи с этим важное значение для практики имеют приближенные методы решения. Многие из них являются модификациями точных.

1. Приближенные модификации переборных методов.

Приближенный вариант метода **разбиения и анализа** отличается от точного только тем, что наряду со способами анализа, гарантирующим исключение допустимых решений, не лучших оптимального, применяются способы анализа, не дающие такой гарантии.

2. Приближенные варианты неперебoрных методов.

Наиболее широко применяется приближенный вариант метода **динамического программирования** (называется методом пошагового построения) и **перестановочного метода** (называется методом локальной оптимизации).

Метод пошагового построения решения. При реализации этого метода процесс построения решения разбивается на этапы на каждом из которого фиксируется одна или несколько компонент вектора x (один или несколько элементов в определенных позициях выборки λ). При этом используются эвристические соображения,

иногда подкрепленные оценками.

Метод локальной оптимизации позволяет путем локальных изменений улучшить некоторое допустимое решение, полученное, например, уже перечисленными переборными методами. В основе метода лежит выбор оператора $g(\gamma)$, действующего на множестве X_1. Метод сводится к построению с помощью оператора $g(\gamma)$ монотонно улучшающейся (по значению целевой функции) последовательности $\{x^{(k)}\}$,

$$x^{(k+1)} = g(\gamma_k)x^{(k)}.$$

Выбор γ_k - параметра осуществляется так, чтобы выполнялось условие $f_0[x^{k+1}] - f_0[x^k] > 0$.

В методе локальной оптимизации могут быть использованы одновременнно несколько операторов типа $g(\gamma)$. Они могут чередоваться при формированиии последовательности $\{x^{(k)}\}$. Построение последовательности продолжается до получения допустимого решения, являющегося локально оптимальным по отношению ко всем использованным операторам.

3.10 Задачи и упражнения

1. С помощью комбинаторных рассуждений (используя только определение числа сочетаний) доказать тождества:
   а) $C(n,k) = C(n-1,k) + C(n-1,k-1)$,
   б) $C(n,k) = C(n-1,k) + C(n-2,k-1) + \ldots + C(n-k-1,0)$.
2. Доказать тождество
$$C(n+m,m) = \sum_{k=0}^{m} C(n+k-1,k).$$
3. На диск кодового замка нанесены 12 букв. «Секретное слово» состоит из пяти букв. Сколько неудачных попыток может сделать человек, не знающий «секретного слова»?
4. В скольких различных состояний может находиться ЭВМ, если ее оперативная память состоит из 2048 ячеек, в каждой из которых 43 двоичных разряда?

5. Из суеверных соображений члены клуба велосипедистов провели перерегистрацию, при которой исключили все номера билетов, содержащие цифру 8. Сколько членов было в клубе, если известно, что использованы все трехзначные номера, не содержащие ни одной «восьмерки» (000 использован)?
6. В n-ичной системе счисления используется n цифр. Сколько в ней натуральных чисел, записываемых ровно k знаками?
7. Сколькими способами можно посадить за круглый стол n мужчин и n женщин так, чтобы никакие два лица одного пола не сидели рядом?
8. Из колоды, содержащей 52 карты, вынули 10 карт. В скольких случаях окажется, что среди вынутых карт
   а) хотя бы один туз; б) ровно один туз;
   в) не менее двух тузов; г) ровно два туза.
9. Сколько существует чисел от 0 до 10^n, в которые не входят две идущие друг за другом одинаковые цифры?
10. Сколькими способами можно разложить n различных шаров по m различным урнам (вместимость урн неограничена)?
11. Сколькими способами можно выбрать 6 карт из колоды, содержащей 52 карты так, чтобы среди них были карты каждой масти?
12. Сколькими способами можно разместить n одинаковых шаров по m различным урнам?
13. Решить задачу 12 при условиях:
    а) пустых урн нет;
    б) во второй урне ровно k шаров.
14. Какова вероятность, купив 1 билет, угадать в спортлото (из 49):
    а) k номеров $k = 1,2,...,6)$;
    б) хотя бы k номеров.
15. Сколькими способами можно разместить n_1 белых, n_2 черных и n_3 синих шаров по m различным урнам?
16. Генуэзская лотерея. Участники этой лотереи покупают билеты, на которых стоят числа от 1 до 90. На некоторых билетах стоят сразу 2, 3, 4 или 5 чисел. В день розыгрыша случайным образом выбирают 5 жетонов с номерами от 1 до 90. Выигрывают те участники, у которых все номера на билетах окажутся среди выбранных (сумма выигрыша зависит от количества чисел на биле-

те). Какова вероятность выигрыша в случае покупки билета:

а) с одним числом;

б) с k числами $(1 \le k \le 5)$?

17. Доказать, что

$$\sum_{i=0}^{m} C(r,i) \cdot C(s,m-i) = C(r+s,m).$$

Указание. $(1+x)^r \cdot (1+x)^s = (1+x)^{r+s}$. Дать другое доказательство этого тождества, рассматривая число способов выбора комиссии в составе m человек из группы, состоящей из r мужчин и s женщин.

18. Сколько существует положительных чисел, меньших чем 10^n (в десятичной системе счисления), цифры которых расположены в неубывающем порядке?

19. Сколькими способами можно n одинаковых подарков раздать r детям:

а) без ограничений;

б) если каждый ребенок должен получить хотя бы один подарок?

20. Из колоды в $4n$ карт, которая содержит четыре различных масти, в каждой масти по n карт $(n \ge 5)$, занумерованных числами $1,\ldots,n,$ выбраны пять карт. Расположить в порядке возрастания частоты появления, зависящей от n, следующие наборы:

а) пять последовательных карт одной масти;

б) четыре карты из пяти с одинаковым номером;

в) три карты с одним номером и две карты с другим;

г) пять карт одной масти;

д) пять последовательно занумерованных карт;

е) три карты из пяти с одним и тем же номером;

ж) две карты с одним номером и две с другим;

з) две карты из пяти с одним номером.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО КУРСУ «ДИСКРЕТНАЯ МАТЕМАТИКА»

для специальностей 220400 и 071900

Часть 2

Контрольные работы №№ 3,4, выполняемые в третьем семестре, имеют целью закрепить теоретические знания, полученные в разделах: «Основы математической логики», «Основы теории алгоритмов», «Элементы комбинаторного анализа».

Контрольная работа № 3

Тема работы – математическая логика. В работе требуется выполнить следующие задания.

1. Определить, является ли справедливой приведенная формула алгебры высказываний, не прибегая к составлению таблицы истинности, а используя только свойства соответствующих операций.
2. Для указанной функции трех переменных:
   - составить таблицу истинности;
   - определить, к каким классам булевых функций она относится;
   - записать совершенные ДНФ и КНФ;
   - найти минимальную ДНФ;
   - для полученной минимальной ДНФ построить логическую схему в базисах: а) { $\vee$, $\bar{}$ } (дизъюнкция, отрицание); б) { $\wedge$, $\bar{}$ } (конъюнкция, отрицание).
3. Доказать полноту (или неполноту) приведенной системы булевых функций.

При выполнении контрольной работы затруднения может вызвать проверка линейности булевой функции.

Для исследования функции на линейность ее следует представить в общем виде с помощью полинома по модулю два, а затем, используя значения функции и значения переменных на конкретных наборах, попытаться определить коэффициенты полинома. Оставшиеся неиспользованными наборы используют для проверки правильности выражения. Полученное хотя бы на одном из наборов

противоречие означает некорректность выражения, а значит и нелинейность функции.

В качестве примера рассмотрим функции двух переменных $f_{10} = x_1 \sim x_2$ – эквивалентность и $f_{14} = x_1 \to x_2$ – импликация. Таблица истинности для этих функций имеет вид:

x_1	x_2	f_{10}	f_{14}
0	0	1	1
0	1	0	1
1	0	0	0
1	1	1	1

Представим f_{10} в виде полинома по модулю два:

$f_{10}(x_1, x_2) = a_0 \oplus a_1 x_1 \oplus a_2 x_2$.

Подставляя значения функции и переменных из набора № 0, получим

$1 = a_0 \oplus a_1 0 \oplus a_2 0$, откуда $a_0 = 1$.

Используя аналогичным образом набор № 1, получим (с учетом того, что $a_0 = 1$)

$0 = 1 \oplus a_1 0 \oplus a_2 1$, откуда $a_2 = 1$.

Из набора № 2:

$0 = 1 \oplus a_1 1 \oplus a_2 0$, откуда $a_1 = 1$.

Подставляя полученные значения коэффициентов в выражение, где значения переменных и функции соответствуют набору № 3, получим

$1 = 1 \oplus 1 \oplus 1$, что является истиной. Отсюда следует, что функция $x_1 \sim x_2$ линейна и может быть представлена как

$x_1 \sim x_2 = 1 \oplus x_1 \oplus x_2$.

Исследуя аналогичным образом функцию $f_{14} = x_1 \to x_2$, получим:

$a_0 = 1$ (из набора № 0);

$a_2 = 0$ (из набора № 1);

$a_1 = 1$ (из набора № 2).

Используя для проверки набор № 3, получим

$1 = 1 \oplus 1 \oplus 0$, что является ложью. Следовательно, функция $f_{14} = x_1 \to x_2$ нелинейна.

Вариант 1

1. Докажите, что $(A \vee B) \wedge (B \vee C) \wedge (C \wedge \bar{A}) = (A \vee B) \wedge (C \wedge \bar{A})$, где A, B, C – простые высказывания.
2. Функция $f(x_1, x_2, x_3)$ принимает единичные значения на наборах №№ 1, 2, 5, 6, 7.
3. Является ли полной система булевых функций, состоящая из дизъюнкции, константы 0 и эквивалентности?

Вариант 2

1. Верно ли, что $(A \wedge (\bar{B} \vee C) \wedge (\bar{C} \vee \bar{D}) = \overline{\bar{A} \vee B \wedge \bar{C} \vee C \wedge D}$, где A, B, C, D – простые высказывания?
2. Функция $f(x_1, x_2, x_3)$ принимает единичные значения на наборах №№ 0, 1, 4, 6, 7.
3. Является ли полной система булевых функций, состоящая из дизъюнкции и конъюнкции?

Вариант 3

1. Верно ли, что

$$\overline{A \wedge B \vee \bar{B} \wedge C \vee \bar{D} \wedge E} = (\bar{A} \vee \bar{B}) \wedge (B \vee \bar{C}) \wedge (D \vee E),$$

где A, B, C, D, E – простые высказывания?
2. Функция $f(x_1, x_2, x_3)$ принимает единичные значения на наборах №№ 0, 1, 3, 6, 7.
3. Является ли полной система булевых функций, состоящая из импликации и отрицания?

Вариант 4

1. Верно ли, что

$$\overline{A \wedge B \vee C \wedge D \vee E} = (\bar{A} \vee B) \wedge (\bar{C} \vee \bar{D}) \wedge E,$$

где A, B, C, D, E – простые высказывания?
2. Функция $f(x_1, x_2, x_3)$ принимает единичные значения на наборах №№ 0, 2, 3, 6, 7.
3. Является ли полной система булевых функций, состоящая из конъюнкции, константы 1 и сложения по модулю два?

Вариант 5

1. Верно ли, что

$$\overline{(A \wedge B \vee C \wedge \bar{D} \vee E \vee F)} = (\bar{A} \vee \bar{B}) \wedge (\bar{C} \vee D) \vee \bar{E} \vee \bar{F},$$

где A, B, C, D, E, F – простые высказывания?
2. Функция $f(x_1, x_2, x_3)$ принимает единичные значения на наборах №№ 1, 2, 3, 5, 6.
3. Является ли полной система булевых функций, состоящая из конъюнкции, константы 0 и эквивалентности?

Вариант 6

1. Верно ли, что

$$\overline{A \wedge B \wedge C \wedge (\bar{D} \vee \bar{E})} = \bar{A} \vee \bar{B} \vee \bar{C} \vee D \wedge E,$$

где A, B, C, D, E – простые высказывания?
2. Функция $f(x_1, x_2, x_3)$ принимает единичные значения на наборах №№ 1, 2, 3, 5, 7.
3. Является ли полной система булевых функций, состоящая из импликации и эквивалентности?

Вариант 7

1. Верно ли, что

$$\overline{(A \vee \bar{B} \vee \bar{C}) \wedge (D \vee \bar{E}) \wedge F} = \bar{A} \wedge B \wedge C \vee \bar{D} \wedge E \vee \bar{F},$$

где A, B, C, D, E, F – простые высказывания?
2. Функция $f(x_1, x_2, x_3)$ принимает единичные значения на наборах №№ 0, 2, 5, 6, 7.
3. Является ли полной система булевых функций, состоящая из конъюнкции и импликации?

Вариант 8

1. Верно ли, что

$$\overline{A \wedge B \wedge C \wedge (D \vee E) \wedge F} = \bar{A} \vee \bar{B} \vee \bar{C} \vee \overline{(D \wedge E)} \vee \bar{F},$$

где A, B, C, D, E, F – простые высказывания?
2. Функция $f(x_1, x_2, x_3)$ принимает единичные значения на наборах №№ 2, 4, 5, 6, 7.
3. Является ли полной система булевых функций, состоящая из

конъюнкции, эквивалентности и сложения по модулю два?

Вариант 9

1. Верно ли, что

$$\overline{A \wedge (\bar{B} \vee \bar{C}) \wedge (D \vee \bar{E}) \wedge F} = \bar{A} \vee B \wedge C \vee \bar{D} \wedge E \vee \bar{F},$$

где A, B, C, D, E, F – простые высказывания?
2. Функция $f(x_1, x_2, x_3)$ принимает единичные значения на наборах №№ 2, 3, 5, 6, 7.
3. Является ли полной система булевых функций, состоящая из дизъюнкции и импликации?

Вариант 10

1. Верно ли, что

$$\overline{(A \vee B) \wedge (C \vee \bar{D}) \wedge E \wedge F} = \bar{A} \wedge \bar{B} \vee \bar{C} \wedge D \vee \bar{E} \vee \bar{F},$$

где A, B, C, D, E, F – простые высказывания?
2. Функция $f(x_1, x_2, x_3)$ принимает единичные значения на наборах №№ 0, 3, 4, 6, 7.
3. Является ли полной система булевых функций, состоящая из сложения по модулю два, константы 1 и эквивалентности?

Контрольная работа № 4

Эта контрольная работа включает в себя задания по теории алгоритмов и перечислительной комбинаторике. В работе требуется выполнить четыре задания.

1. Приведите три самостоятельных примера применения оператора подстановки к простейшим числовым функциям. Например,
$$s(C_2^3 (I_1^3 (3, 2, 4), I_2^3 (5, 8, 1), I_3^3(5, 6, 7))) = 3.$$
2. Приведите два самостоятельных примера применения оператора примитивной рекурсии (аналогично примерам из конспекта лекций).
3. Напишите программу для машины Тьюринга в соответствии с Вашим вариантом.
4. Решите комбинаторную задачу в соответствии с Вашим вариантом.

Вариант 1

1. Составьте программу машины Тьюринга, печатающей число 0. В результате работы программы происходит следующее преобразование машинных слов:
$$01^x q_1 1^y 000 \rightarrow 0\,1^{x+y}\,0\,q_0 10.$$
2. В n-ичной системе счисления используется n цифр. Сколько в ней натуральных чисел, записываемых k знаками?

Вариант 2.

1. Составьте программу машины Тьюринга, стирающей данный массив единиц. В результате работы программы происходит следующее преобразование машинных слов:
$$01^x 0\,1^y 1^{z-1} q_1 1\,0 \rightarrow 0\,1^{x-1} q_0 10\,1^{y+z+1}.$$
2. Сколькими способами можно посадить за круглый стол n мужчин и n женщин так, чтобы никакие два лица одного пола не сидели рядом?

Вариант 3

1. Составьте программу машины Тьюринга, уменьшающей данное число на единицу. В результате работы программы происходит следующее преобразование машинных слов:
$$01^x q_1 1^y 0 \rightarrow 0\,1^{x+y-2} q_0 100.$$
2. Из колоды, содержащей 52 карты, вынули 10 карт. В скольких случаях окажется, что среди вынутых карт
а) хотя бы один туз; б) ровно один туз.

Вариант 4

1. Составьте программу машины Тьюринга, заполняющей единицами промежуток до следующего справа массива единиц (за исключением одного нуля). В результате работы программы происходит следующее преобразование машинных слов:
$$01^x q_1 1\,0\,1^{y+1}\,1^z \rightarrow 0\,1^{x+y} q_0 10\,1^z\,0$$
($x, y \geq 0, z > 0$).
2. Из колоды, содержащей 52 карты, вынули 10 карт. В скольких случаях окажется, что среди вынутых карт
а) не менее двух тузов; б) ровно два туза.

Вариант 5

1. Составьте программу машины Тьюринга, сдвигающей головку влево на следующий массив единиц. В результате работы программы происходит следующее преобразование машинных слов:

$$01^{x+1}\,0^{y}\,1^{z}\,q_1 1^{w}\,0 \;\rightarrow\; 0\,1^{x}\,q_0 10^{y}\,1^{z+w}\,0.$$

2. Сколько существует чисел от 0 до 10^n, в которые не входят две идущие друг за другом одинаковые цифры?

Вариант 6

1. Составьте программу машины Тьюринга, печатающей число 1. В результате работы программы происходит следующее преобразование машинных слов:

$$01^{x}\,q_1 1^{y}\,0000 \;\rightarrow\; 0\,1^{x+y}\;0\,1q_0 10.$$

2. Сколькими способами можно выбрать 6 карт из колоды, содержащей 52 карты так, чтобы среди них были карты каждой масти?

Вариант 7

1. Составьте программу машины Тьюринга, стирающей предыдущий массив единиц. В результате работы программы происходит следующее преобразование машинных слов:

$$01^{x}\,0^{y}\,1^{z-1}\,q_1 1\,0 \;\rightarrow\; 0^{x+y}\,q_0 0\,1^{z}\,0.$$

2. Сколькими способами можно разместить n одинаковых шаров по m различным урнам при условиях:

   а) пустых урн нет;

   б) во второй урне ровно k шаров.

Вариант 8

1. Составьте программу машины Тьюринга, уменьшающей данное число на два. В результате работы программы происходит следующее преобразование машинных слов:

$$01^{x}\,q_1 1^{y}\,0 \;\rightarrow\; 0\,1^{x+y-3}\,q_0 1000.$$

2. Сколькими способами можно разместить n_1 белых, n_2 черных и n_3 синих шаров по m различным урнам?

Вариант 9

1. Составьте программу машины Тьюринга, заполняющей единицами промежуток до следующего слева массива единиц (за исключением одного нуля). В результате работы программы происходит следующее преобразование машинных слов:

$$01^x 0\, 1^{y+1} 1^{z-1} q_1 1\, 0 \;\rightarrow\; 0\, 1^{x-1} q_0 10\, 1^{y+z} 0.$$

2. Одновременно подбрасывают три кубика, имеющих 6, 8 и 10 граней соответственно. Сколькими различными способами они могут упасть, если известно, что, по крайней мере, два кубика упали на цифру 1?

Вариант 10

1. Составьте программу машины Тьюринга, сдвигающей головку вправо на следующий массив единиц. В результате работы программы происходит следующее преобразование машинных слов:

$$01^x q_1\, 1^y 0\, 1^{z+1} 0 \;\rightarrow\; 0\, 1^{x+y} 0\, 1^z q_0 10.$$

2. Сколькими способами из 28 костей домино можно выбрать две кости так, чтобы их можно было приложить друг к другу (т.е. какое-то число очков встречалось на обеих костях)?

СПИСОК ЛИТЕРАТУРЫ

1. Виленкин Н.Я. Популярная коибинаторика. – М. Наука, 1975. – 328 с.
2. Горбатов В.А. Основы дискретной математики. – М.: Высш. школа, 1986. – 312 с.
3. Кориков А.М., Сафьянова Е.Н. Основы системного анализа и теории систем: Учебное пособие. – Томск: изд-во Том. ун-та, 1989. – 207 с.
4. Мальцев А.И. Алгоритмы и рекурсивные функции. 2-е изд. – М.: Наука, 1986. – 368 с.
5. Основы кибернетики. Математические основы кибернетики / Под. ред. К.А. Пупкова. – М.: Высш. школа, 1974. – 416 с.
6. Риордан Дж. Введение в комбинаторный анализ. – М. ИЛ, 1963. – 288 с.
7. Рыбников К.А. Введение в комбинаторный анализ. – М.: Изд-во МГУ, 1985. – 312 с.
8. Шевелев Ю.П. Высшая математика 5. Дискретная математика. Ч.1: Теория множеств. Булева алгебра (для автоматизированной технологии обучения): Учебное пособие. – Томск: Том. гос. ун-т систем управления и радиоэлектроники, 1998. – 114 с.
9. Шевелев Ю.П. Высшая математика 6. Дискретная математика. Ч.2: Теория конечных автоматов. Комбинаторика. Теория графов (для автоматизированной технологии обучения): Учебное пособие. – Томск: Том. гос. ун-т систем управления и радиоэлектроники, 1999. – 120 с.