Министерство образования Республики Беларусь
Учреждение образования
Белорусский государственный технологический университет

В.А. Иванюкович

ОСНОВЫ ДИСКРЕТНОЙ МАТЕМАТИКИ

Минск
2018

Автор:
В.А. Иванюкович, кандидат физико-математических наук, доцент

Рецензенты:
заведующий кафедрой управления информационными ресурсами

Иванюкович, В.А.

И18 Основы дискретной математики: учебно-методическое пособие /

ISBN

Приводятся избранные сведения из алгебры логики и теории множеств, теории отношений и функций, булевой алгебры, комбинаторики, понятия графов и некоторые операции над ними, которые лежат в основе различных программных и технических приложений. Материал сопровождается большим количеством примеров решения задач, которые позволяют усвоить учебный материал.

Предназначено для студентов факультета информационных технологий Белорусского государственного технологического университета

УДК 519.1
ББК 22.174

ISBN

Оглавление

1. Введение в дисциплину

Дискретная математика и логика является основой любого современного изучения информатики. Предметом дискретной математики являются конечные совокупности различимых объектов, называемых множествами, и определенные на них структуры.

Темы, которые представлены в учебном пособии, широко используются как в самой математике, так и в дисциплинах, использующих математический аппарат. Например, применяемые в информатике формальные методы опираются на такие понятия дискретной математики, как логика, множества, отношения и функции, графы. Так, современные системы баз данных построены на реляционной модели данных, в основе которых лежит теория отношений.

В основе учебного пособия лежит книга Рода Хаггарти, которая используется им для преподавания начального курса информатики в Оксфорде, и некоторые другие учебные материалы, перечисленные в списке рекомендуемой литературы.

Для успешного изучения дисциплины необходимы знания основ высшей математики и программирования, предусмотренных учебными программами соответствующих дисциплин для специальностей, ориентированных на подготовку специалистов в области информационных технологий.

1.1. Моделирование

Математическое моделирование – это процесс, использующий математический аппарат для решения реальных задач с требуемой точностью. Его можно представить в виде диаграммы, изображенной на рис. 1.1.

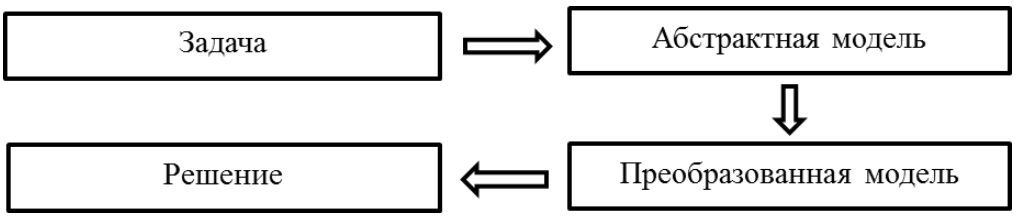


Рисунок 1.1. Основные этапы математического моделирования

В качестве примера моделирования рассмотрим следующую задачу: расстояние между шестью белорусскими городами – Брест, Витебск, Гомель, Гродно, Минск и Могилев – дано в табл. 1.1. Требуется найти дорожную сеть минимальной длины, связывающую все шесть городов. Сама таблица является абстрактной моделью реальной задачи. На ее основе трудно получить решение. Поэтому преобразуем ее в графическую модель, представляющую

Таблица 1.1. Расстояние между городами (аьстрактная модель задачи)

	Брест	Витебск	Гомель	Гродно	Минск	Могилев
Брест	–	618	531	274	348	526
Витебск		–	334	538	270	169
Гомель			–	573	308	178
Гродно				–	268	475
Минск					–	327
Могилев						–

собой граф, чьи вершины обозначают города, а ребра – связывающие их

Рисунок 1.2. Графическая (преобразованная) модель задачи

дороги. Граф, изображенный на рис. 1.2, называется нагруженным. Ребра имеют вес, который равен расстоянию между соответствующими городами и

соответствует данным из таблицы 1.1. Более подробно графы будут изучены в отдельных разделах курса. Для решения поставленной задачи с помощью подходящего *алгоритма* (алгоритм – это последовательность однозначных команд, выполнение которых дает решение поставленной задачи за конечное время) мы построим новый граф, в котором все шесть городов будут соединены дорогами, а минимальный общий вес ребер будет минимальным.

Алгоритм Прима

Шаг 1. Выбрать произвольную вершину и ребро, соединяющее ее с ближайшим (по весу) соседом.
Шаг 2. Найти не присоединенную (еще) вершину, ближе всего лежащую к одной из присоединенных, и соединить эти вершины.
Шаг 3. Повторять шаг 2 до тех пор, пока все вершины не будут соединены.

На рис. 1.3 изображен граф, который получается в результате применения алгоритма Прима, если начинать с вершины «Минск». Этот граф изображает

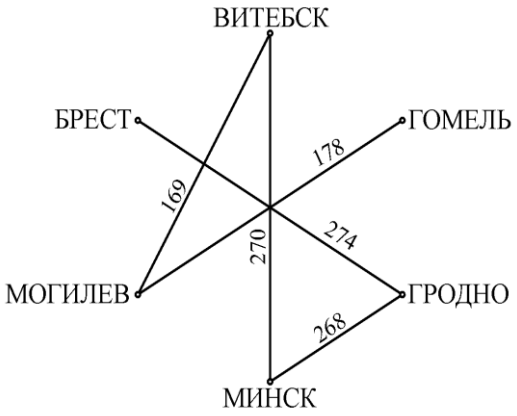


Рис. 1.3. Граф, построенный при помощи алгоритма Прима

минимальную сеть дорог длиной 1159 км, соединяющую все шесть городов. Если в качестве первой вершины выбрать другой город, то получим такой же результат. Это дает надежду на то, что примененный для решения нашей задачи алгоритм верный, но не доказывает, что он верен всегда.

Алгоритм, который мы применяли, написан на языке человеческого общения. Разговорный язык обычно оказывается слишком неоднозначным и, вследствие этого, не соответствующим сложной проблеме решения задачи. С

другой стороны, языки программирования, которые применяются для решения математических задач, зачастую скрывают истинный смысл алгоритма, что затрудняет его анализ и развитие. Подходящий компромисс в этой ситуации – использовать так называемый псевдокод, состоящий из небольшого числа структурных элементов языка программирования вместе с «разговороподобным» описанием действий реализуемого алгоритма.

1.2. Псевдокод

Мы будем использовать псевдокод, основанный на языке программирования Pascal. Алгоритм в нем выглядит следующим образом:

```
begin
    операторы исполняемых действий
    операторы, управляющие порядком выполнения
end
```

Элементами алгоритмического языка являются операторы, которые можно разбить на две категории: операторы присваивания и операторы управления.

Оператор присваивания приписывает переменным определенные величины и имеет такую общую форму: имя переменной := выражение

Пример 1.1. Алгоритм сложения двух чисел *A* и *B* и присвоение результата переменной *Sum*.

```
begin
    Input A and B;
    Sum := A + B
end
```

Оператор управления определяет порядок, в котором должны выполняться шаги алгоритма. Операторы управления бывают трех типов:

- составные операторы;
- условные операторы;
- операторы цикла.

Составные операторы представляют собой список операторов, которые должны выполняться как отдельная команда в том порядке, в котором они записаны. Составные операторы имеют следующий вид:

```
begin
    оператор 1;
    оператор 2;
    ...
    оператор n
end
```

Пример 1.2. Алгоритм обмена значений двух переменных *A* и *B*.

```
begin
    input A and B;
    Temp := A;
    One := B;
    B := Temp
end
```

Условные операторы позволяют делать выбор между двумя альтернативными ситуациями. Они записываются в виде **if-then** или **if-then-else.** В псевдокоде условные операторы изображают так:

```
begin
    if условие then оператор
end
```

или так:

```
begin
    if условие then оператор 1
    else оператор 2
end
```

Пример 1.3. Алгоритм вычисления модуля числа *n* и присвоение результата переменной *abc*.

```
begin
    input n;
    if n < 0 then abc:= –n
    else abc:=n;
    output abc;
end
```

В этом алгоритме оператор, стоящий во второй строке, выполняется при

отрицательных значениях переменной *n*, а в третьей – при положительных (и нулевом). Можно написать и другой алгоритм, решающий ту же задачу, но не использующий **else:**

```
begin
    input n;
    if n < 0 then n:= – n;
    abc:=n;
    output abc;
end
```

Здесь оператор во второй строчке выполняется только при отрицательных значениях *n* и игнорируется при любом другом значении. При неотрицательных значениях *n* выполняется оператор, записанный в третьей строке.

Оператор цикла или просто *цикл* может иметь одну из трех форм записи:

1. **for** *X := A* **to** *Z* **do** оператор;
2. **while** выражение **do** оператор;
3. **repeat**
   оператор 1;
   оператор 2;
   …
   оператор n;
   until условие.

Здесь *X* – переменная, а *A* и *Z* – ее начальное и конечное значения.

В случае (1) цикл повторяется определенное число раз. Его разновидность выглядит следующим образом:

for всех элементов множества **do** оператор

В случае (2) цикл выполняется не определенное число раз, а до тех пор, пока выражение, о котором в нем идет речь, остается верным. Как только выражение становится ложным, цикл заканчивается.

В случае (3) цикл выполняется до тех пор, пока конечное условие остается ложным. Единственное различие между (2) и (3) заключается в том, что в последнем цикл выполнится по крайней мере один раз, поскольку истинность условия в нем проверяется *после* каждого прохода цикла.

Пример 1.4. Алгоритм вычисления суммы квадратов первых *n* натуральных чисел.

```
begin
    Sum:=0;
    for i:= 1 to n do
    begin
       j := i * i;
       Sum := Sum + j;
    end
    output Sum
End
```

Проследим алгоритм в случае *n* = 4, записав результаты в табл. 1.2.

Таблица 1.2 Действие алгоритма из примера 1.4

	i	***j***	***Sum***
Перед выполнением цикла	–	–	0
Первый проход цикла	1	1	1
Второй проход цикла	2	4	5
Третий проход цикла	3	9	14
Четвертый проход цикла	4	16	30

Выводимый результат: *Sum* = 30. В том случае, когда есть несколько различных алгоритмов, решающих одну и ту же задачу, возникает вопрос: какой из них является более эффективным? Эффективность алгоритма определяется скоростью вычислений и размером используемой памяти. Кроме того, желательно, чтобы алгоритм был понятен для анализа.

Рассмотрим еще один алгоритм.

Пример 1.5.

```
begin
    Input n;
    k:= 1;
    l:= 0;
    Sum:= 0;
    while k< 2n do
    begin
       l:=l + k;
       Sum:=Sum+ l;
       k:=k + 2;
    end
    Output Sum;
```

End

Что считает этот алгоритм? Результат его действия не очевиден.

Проследим изменение значений переменных *l*, *Sum* и *k* при *n* = 6 (см. табл. 1.3).

Таблица 1.3. Действие алгоритма из примера 1.5

l	***Sum***	***k***	***k*<12?**
0	0	1	Да
1	1	3	Да
4	5	5	Да
9	14	7	Да
16	30	9	Да
25	55	11	Да
36	91	13	Нет

Видим, что значение параметра *Sum* также, как в примере 1.4 равно сумме квадратов первых *n* натуральных чисел. Очевидно, что из-за дополнительного составного оператора, рассчитывающего значение *k*, время вычислений и размер используемой памяти увеличиваются, снижая эффективность алгоритма. Кроме того, как отмечалось выше, действие этого алгоритма не так очевидно, как алгоритма, примененного в примере 1.4.

Сейчас запишем псевдокод для рассмотренного ранее алгоритма Прима, который позволяет выделить граф с минимальным общим весом в исходном графе, имеющем ребра между любыми двумя вершинами. В нем мы не будем детализировать отдельные блоки программы, а запишем только этапы вычислений.

Пример 1.6. Псевдокод алгоритма Прима.

```
begin
    v:= произвольная вершина;
    u:= ближайшая соседняя вершина;
    связать v и u;
    while остаются неприсоединенные вершины do
      begin
       u:= неприсоединенная вершина, ближайшая к одной из
       присоединенных вершин;
       соединить вершину u с ближайшей из присоединенных вершин;
      end
end
```

Всегда необходимо проверять корректность написанного на псевдокоде алгоритма. Например, откуда мы можем знать, что алгоритм из примера 1.6 действительно дает минимальную сеть дорог?

Проблемы корректности и эффективности алгоритмов будут обсуждаться в дальнейшем после того, как мы освоим необходимый для этого аппарат дискретной математики.

2. Логика и доказательство

В любой формальной дисциплине логика состоит из правил получения обоснованного вывода. Логику можно выделить из контекста тех дисциплин, в которых она используется, и изучать как отдельный раздел науки. Математическая логика лежит в основе неоспоримых рассуждений и доказательств. Мы изучим логику высказываний, имеющую дело с истинностью или ложностью простых описательных утверждений, а также научимся работать с составными высказываниями. Рассмотрим понятие предикатов, которые обобщают логику высказываний и широко используются в алгоритмах обработки массивов данных.

2.1. Высказывания и логика

Стандартными блоками формальной логики являются высказывания. *Высказыванием* называется утверждение, которое имеет значение истинности, т. е. может быть *истинным* (обозначается буквой *И***)** или *ложным* (обозначается *Л*). Каждое из высказываний можно обозначить своей буквой. Пусть, например, *P* обозначает высказывание «Земля плоская», *Q* – «Катя – врач» и *R* – «29 – простое число».

Используя такие логические операторы, как *не, или, и*, можно построить новые *составные высказывания* как комбинацию простых. Например:

- **(не** *P)* – это высказывание «земля не плоская»;
- *(P* **или** *Q)* – «земля плоская» или «Катя – врач»;
- *(P* **и** *Q)* – «земля плоская» и «Катя – врач».

Чтобы уметь определять значение истинности составных высказываний, нам необходимо разобраться со смыслом логических операций, т.е., какой эффект они оказывают на истинностное значение простых высказываний. Результат действия логических операторов принято представлять при помощи

таблиц истинности.

Отрицанием произвольного высказывания *P* называется высказывание вида (**не** *P*), чье истинностное значение строго противоположно значению *P*. В табл. 2.1 приведена таблица истинности отрицания высказывания.

Таблица 2.1 Таблица истинности отрицания высказывания

P	**(не *P*)**
И	*Л*
Л	*И*

Конъюнкцией, или *логическим умножением* двух высказываний *P* и *Q*, называют составное высказывание вида *(P* **и** *Q)*. Оно принимает истинное значение только в том случае, когда истинны обе его составные части. Такое определение хорошо согласуется с обычным пониманием союза «и» в разговорном языке. В табл. 2.2 показана соответствующая таблица истинности.

Таблица 2.2 Таблица истинности конъюнкции двух высказываний

P	***Q***	**(*P* и *Q*)**
И	*И*	*И*
И	*Л*	*Л*
Л	*И*	*Л*
Л	*Л*	*Л*

Дизъюнкцией, или *логическим сложением* двух высказываний *P* и *Q*, называется составное высказывание *(P* **или** *Q)*. Оно истинно, если хотя бы одна из ее составных частей имеет истинное значение, что в некотором смысле также согласуется с обычным пониманием союза «или». Другими словами, *(P* **или** *Q)* означает, что «или *P,* или *Q,* или и то, и другое». Таблица истинности дизъюнкции:

Таблица 2.3 Таблица истинности дизъюнкции двух высказываний

P	***Q***	**(*P* или *Q*)**
И	*И*	*И*
И	*Л*	*И*
Л	*И*	*И*
Л	*Л*	*Л*

Пример 2.1. Что можно сказать об истинности составного высказывания: «либо луна делается из зеленого сыра и Генрих VIII имел шесть жен, или не верно, что дронт вымер»? (Дронт – вымершая птица отряда голубеобразных, обитавшая на островах Индийского океана и истребленная в XVII–XVIII вв. завезенными туда свиньями.)

Решение. Обозначим через *P* высказывание «луна делается из зеленого сыра», через *Q* – «Генрих VIII имел шесть жен» и через *R* – «дронт вымер». Символьная запись данного высказывания имеет вид: *(P* **и** *Q)* **или (не** *R).* Известно, что высказывание *P* ложно, а *Q* и *R* истинны. Поэтому высказывание *(P* **и** *Q)* **или (не** *R)* имеет такое истинностное значение: (Л **и** И) **или** Л, что эквивалентно Л. Следовательно, составное высказывание ложно.

Дадим определение логически эквивалентных высказываний. Два составных высказывания называются *логически эквивалентными*, если они построены разными путями из одних и тех же простых высказываний и могут принимать одинаковые значения истинности на любом возможном наборе значений истинности своих составных частей.

Пример 2.2. Показать, что высказывание **(не** (P **и (не** *Q)))* логически эквивалентно утверждению **((не** *P)* **или** *Q).*

Решение. Заполним совместную таблицу истинности (табл. 2.4) для составных высказываний:

R = **(не** (P **и (не** *Q)))* и *S* = ((не *P)* **или** *Q).*

Вспомогательные колонки 3, 4, 5 используются для построения обоих выражений *R* и *S* из утверждений *P* и *Q*.

Таблица 2.4 Таблица истинности для примера 2.2

P	**Q**	**не P**	**не Q**	**P и (не Q)**	**R**	**S**
1	2	3	4	5	6	7
И	*И*	*Л*	*Л*	*Л*	*И*	*И*
И	*Л*	*Л*	*И*	*И*	*Д*	*Д*
Л	*И*	*И*	*Л*	*Л*	*И*	*И*
Л	*Л*	*И*	*И*	*Л*	*И*	*И*

Две последние колонки таблицы идентичны. Это означает, что высказывание *R* логически эквивалентно высказыванию *S*.

Важно понимать еще один тип логического оператора, результатом которого является *условное высказывание.* Пример такого высказывания: «если завтра будет суббота, то сегодня – пятница».

В логике условное высказывание «**если** *P*, **то** *Q*» принято считать ложным только в том случае, когда *предпосылка P* истинна, а *заключение Q* ложно. В любом другом случае условное высказывание считается истинным.

Используя символ импликации «⇒», мы пишем $P \Rightarrow Q$ для обозначения условного высказывания «**если** *P*, **то** *Q*». Такая запись читается как «из *P* следует *Q*». Значения истинности условного высказывания или импликации приведены в табл. 2.5.

Таблица 2.5 Таблица истинности импликации двух высказываний

P	***Q***	***(P⇒Q)***
И	*И*	*И*
И	*Л*	*Л*
Л	*И*	*И*
Л	*Л*	*И*

При определении истинностного значения условного высказывания необходимо различать фактическую и логическую истину. Рассмотрим это на примере, который показывает отличие между фактической и логической истиной.

Пример 2.3. Пусть *P* – ложное высказывание 1 = 5, *Q* – ложное высказывание 3 = 7 и *R* – истинное утверждение 4 = 4. Показать, что условные высказывания: «если *P,* то *Q*» и «если *P*, то *R*» являются истинными.

Решение. Если 1 = 5, то, прибавляя 2 к обеим частям равенства, мы получим, что 3 = 7. Следовательно, высказывание «если *P*, то *Q*» справедливо. Вычтем теперь из обеих частей равенства 1 = 5 число 3 и придем к равенству –2 = 2. Но $(-2)^2 = 2^2$, т. е. 4= 4. Таким образом, «если *P*, то *R*» тоже верно.

Этот пример показывает, что если предпосылка *P* ложна, то мы можем получить логически корректное высказывание и когда *Q* ложно, и когда оно

истинно. В том случае, когда предпосылка P истинна, мы не можем получить логически корректного заключения, если Q ложно. Т.е., из истины ложь не следует.

Пример 2.4. Высказывание ((**не** Q)$\Rightarrow$(**не** P)) называется *противоположным*, или *контрапозитивным*, к высказыванию ($P \Rightarrow Q$). Показать, что ((**не** Q) $\Rightarrow$ **(не** P)) логически эквивалентно высказыванию ($P \Rightarrow Q$).

Решение. Рассмотрим совместную таблицу истинности (табл. 2.6).

Таблица 2.6 Таблица истинности для примера 2.4

P	***Q***	**не *P***	**не *Q***	**(*P*⇒*Q*)**	**((не *Q*)⇒(не *P*)**
И	*И*	*Л*	*Л*	*И*	*И*
И	*Л*	*Л*	*И*	*Л*	*Л*
Л	*И*	*И*	*Л*	*И*	*И*
Л	*Л*	*И*	*И*	*И*	*И*

Поскольку два последних столбца этой таблицы совпадают, то высказывания **(не** $Q \Rightarrow$ **(не** P**)** и $P \Rightarrow Q$ логически эквивалентны.

Рассмотрим пример того, как можно использовать математическую логику для решения задач.

Пример 2.5. Студенты Комаров, Соколов и Мальцев сдавали экзамен. Известно, что истинными являются два утверждения:

а) если не сдал Соколов или сдал Комаров, то сдал и Мальцев;

б) если не сдал Соколов, то не сдал и Мальцев.

Определить, кто сдал экзамен.

Решение. Обозначим простые высказывания «сдал экзамен Комаров» как К, «сдал экзамен Соколов» как С и «сдал экзамен Мальцев» как М. Тогда известные факты можно записать в виде составных высказываний:

а) ((**не** С) **или** К) $\Rightarrow$ М;

б) (**не** С) $\Rightarrow$ (**не** М).

Поскольку оба высказывания истинны, то их конъюнкция также истинна, т.е.,

с) ((((**не** С) **или** К) ⇒ М) **и** ((**не** С) ⇒ (**не** М))) = *И*.

Составим таблицу истинности для этого высказывания (табл. 2.7.).

Таблица 2.7. Таблица истинности для составного высказывания
А = (((**не** С) **или** К) ⇒ М) **и** ((**не** С) ⇒ (**не** М))

К	***С***	***М***	((**не** С) **или** К) ⇒ М	(**не** С) ⇒ (**не** М)	***А***
И	*И*	*И*	*И*	*И*	*И*
И	*И*	*Л*	*Л*	*И*	*Л*
И	*Л*	*И*	*И*	*Л*	*Л*
И	*Л*	*Л*	*Л*	*И*	*Л*
Л	*И*	*И*	*И*	*И*	*И*
Л	*И*	*Л*	*И*	*И*	*И*
Л	*Л*	*И*	*И*	*Л*	*Л*
Л	*Л*	*Л*	*Л*	*И*	*Л*

Итак, значение истина получено в трех строках таблицы истинности. В каждой из этих строк простое высказывание С имеет значение истины, т.е., можно утверждать, что Соколов сдал экзамен. Высказывания К и М принимают значения истина и ложь. Следовательно, нет достаточно данных для выводов о сдаче экзамена Комаровым и Мальцевым.

2.2. Предикаты и кванторы

Логика высказываний применяется к простым декларативным высказываниям, где базисные высказывания – либо истинны, либо ложны. Утверждения, содержащие переменные, могут быть верными при некоторых значениях переменных и ложными при других.

Предикатом называется утверждение, содержащее переменные и принимающее значение истина или ложь в зависимости от значений переменных. Например, выражение *«x* – целое число, удовлетворяющее соотношению $x = x^2$*»* является предикатом, поскольку оно истинно при x=0 или x=1 и ложно в любом другом случае.

Логические операции можно применять и к предикатам. В общем случае истинность составного предиката зависит от значений входящих в него переменных. Однако существуют логические операторы, называемые

кванторами, применение которых к предикатам превращает последние в ложные или истинные высказывания.

Часто мы имеем дело с утверждениями о том, что некоторое свойство имеет место *для всех* рассматриваемых объектов или что *существует* по крайней мере один объект, обладающий данным свойством (например: Сумма внутренних углов любого треугольника равна 180°. Или: Существует простое четное число).

Выражения «для всех» и «существует» называются *кванторами* и обозначаются $\forall$ и $\exists$ соответственно. Включая в предикат кванторы, мы превращаем его в высказывание. Поэтому предикат с кванторами принимает значение истина или ложь.

Пример 2.6. Обозначим через *P(x)* предикат *«x* – целое число и $x^2=16$». Выразите словами высказывание $\exists x:P(x)$ и определите его истинностное значение.

Решение. Высказывание $\exists x:P(x)$ означает, что найдется целое число *x,* удовлетворяющее уравнению $x^2=16$. Высказывание, конечно, истинно, поскольку уравнение $x^2=16$ превращается в верное тождество при $x=4$. Кроме того, $x= -4$ – также решение данного уравнения. Однако нам не требуется рассуждать о знаке переменной *x,* чтобы проверить истинность высказывания $\exists x:P(x)$.

Использование квантора $\forall$ сделает этот же предикат ложью.

Для общего предиката *P(x)* есть следующие логические эквивалентности: (в символьной форме логически эквивалентные высказывания обозначаются значком «$\Leftrightarrow$»):

не $\exists x\ P(x) \Leftrightarrow \forall$x не *P(x);*

не $\forall x:P(x) \Leftrightarrow \exists x:P(x)$.

Предикат может быть преобразован в высказывание при помощи нескольких кванторов.

Пример 2.7. Предположим, что x и y – вещественные числа, а $P(x, y)$ обозначает предикат $x + y = 0$. Выразите каждое из высказываний словами и определите их истинность.

(а) $\forall x \exists y: P(x,y)$;

(б) $\exists y: \forall x\ P(x,y)$.

Решение.

(а) Высказывание $\forall x\ \exists y:\ P(x,y)$ говорит о том, что для любого вещественного числа x найдется такое вещественное число y, что $x+y=0$. Оно, очевидно, верно, поскольку какое бы число x мы ни взяли, число $y=-x$ обращает равенство $x+y=0$ в верное тождество.

(б) Высказывание $\exists y: \forall x\ P(x,y)$ читается следующим образом: существует такое вещественное число y, что для любого вещественного числа x выполнено равенство $x+y=0$. Это, конечно, не так: не существует вещественного числа y, обладающего указанным свойством. Следовательно, высказывание ложно.

Ранее говорилось, что предикаты широко используются при работе с массивами данных. Например, условие отбора данных из массивов задается в виде предиката. Будут отобраны те данные, для которых предикат принимает значение истина.

2.3. Методы доказательств

Итак, при доказательстве математических теорем применяется логическая аргументация. Доказательства в информатике – неотъемлемая часть проверки корректности алгоритмов. Необходимость доказательства возникает, когда нам нужно установить истинность высказывания вида $(P \Rightarrow Q)$. Существует несколько стандартных способов доказательств. Мы рассмотрим некоторые из них.

Прямое рассуждение

Предполагаем, что высказывание P истинно и показываем

справедливость высказывания Q. Такой способ доказательства исключает ситуацию, когда P истинно, а Q – ложно, поскольку именно в этом и только в этом случае импликация или условное высказывание $(P \Rightarrow Q)$ принимает ложное значение.

Пример 2.8. Покажите прямым способом рассуждений, что произведение xy двух нечетных целых чисел x и y всегда нечетно.

Решение. Любое нечетное число, в том числе и x, можно записать в виде $x=2m+1$, где m – целое число. Аналогично, $y=2n+1$ с некоторым целым n. Значит, их произведение $xy = (2m+1)(2n+1) = 4mn+2m+2n+1 = 2(2mn+m+n)+1$, то есть, тоже является нечетным числом.

Обратное рассуждение

В примере 2.4 было доказано, что высказывание $(P \Rightarrow Q)$ и противоположное высказывание ((**не** Q)$\Rightarrow$(**не** P)) являются логически эквивалентными. Поэтому для доказательства справедливости высказывания $(P \Rightarrow Q)$ предполагаем, что высказывание Q ложно и показываем ошибочность высказывания P. То есть, фактически прямым способом проверяем истинность импликации ((**не** Q)$\Rightarrow$(**не** P)).

Пример 2.9. Пусть n – натуральное число. Покажите, используя обратный способ доказательства, что если n^2 нечетно, то и n нечетно.

Решение. Отрицанием высказывания о нечетности числа n^2 служит утверждение «n^2 – четно», а высказывание о четности n является отрицанием утверждения «число n нечетно». Таким образом, нам нужно показать прямым способом рассуждений, что четность числа n влечет четность его квадрата n^2.

Так как n четно, то $n=2m$ для какого-то целого числа m. Следовательно, $n^2 = 4m^2 = 2(2m^2)$, то есть четное число.

Аналогичными рассуждениями доказывается, что если n^2 четно, то и n четно.

Метод «от противного»

Предположив, что высказывание P истинно, а Q ложно, используя аргументированное рассуждение, получаем противоречие. Этот способ также основан на том, что импликация $(P \Rightarrow Q)$ принимает ложное значение только тогда, когда P истинно, а Q ложно.

Пример 2.10. Методом «от противного» покажите, что корень уравнения $x^2=2$ является иррациональным числом, те есть, не может быть записан в виде дроби с целыми числителем и знаменателем.

Решение. Допустим, что корень x уравнения $x^2=2$ рационален, т.е. записывается в виде дроби $x=m/n$ с целыми m и n, причем $n\neq 0$. Для доказательства исходного утверждения нам необходимо получить противоречие либо с предположением, либо с каким-то ранее доказанным фактом.

Как известно, рациональное число неоднозначно записывается в виде дроби. Например, $x = m/n = 2m/2n = 3m/3n = \ldots$ Для однозначности записи будем считать, что m и n не имеют общих делителей.

Итак, предполагаем дополнительно, что дробь $x = m/n$ несократима (m и n не имеют общих делителей). По условию число x удовлетворяет уравнению $x^2 = 2$. Значит, $(m/n)^2 = 2$, откуда $m^2 = 2n^2$.

Из последнего равенства следует, что число m^2 четно. Следовательно, m тоже четно (доказано в примере 2.9) и может быть представлено в виде $m = 2p$ для какого-то целого числа p. Подставив такое значение в равенство $m^2=2n^2$, мы получим, что $4p^2 = 2n^2$, т. е. $n^2 = 2p^2$. Но тогда n тоже является четным числом. Таким образом, мы показали, что как m, так и n – четные числа. Поэтому они обладают общим делителем 2. Но это противоречит нашему предположению об отсутствии общего делителя у числителя и знаменателя дроби m/n. Это противоречие приводит нас к однозначному выводу: решение уравнения $x^2=2$ не может быть рациональным числом, т. е. оно иррационально.

Математическая индукция

Математическая индукция – еще один мощный и широко используемый метод доказательств.

Компьютерную программу называют *корректной,* если она делает то, что указано в ее спецификации. Несмотря на то, что тестирование программы может давать ожидаемый результат в случае каких-то отдельных начальных данных, необходимо доказать, используя приемы формальной логики, что правильные выходные данные будут получаться при любых вводимых начальных значениях.

Одним из случаев, требующих строгого доказательства корректности работы, являются алгоритмы, содержащие циклы. Для такой проверки используется, как правило, метод *математической индукции.* Продемонстрируем возможности этого метода, доказав корректность следующего рекуррентного алгоритма, определяющего максимальный элемент из конечного набора натуральных чисел $a_1, a_2, a_3, \ldots, a_n$:

```
begin
  j:= 0;
  M:= 0;
       while j < n  do
          begin
                  j:= j + 1;
                  M:= max(M, aj);
          end
end
```

Действие алгоритма на наборе данных: $a_1 = 4$, $a_2 = 7$, $a_3 = 3$ и $a_4 = 8$ прослежено в табл. 2.7. В качестве выходных данных получено значение $M = 8$, что соответствует действительности. Заметим, что после каждого прохода цикла переменная M равна наибольшему из чисел набора, просмотренных к этому моменту.

Таблица 2.8 Пример работы алгоритма

j	***M***	***j* < 4?**
0	0	Да

1	4	Да
2	7	Да
3	7	Да
4	8	Нет

Но будет ли алгоритм работать правильно при любом вводимом наборе чисел длины n?

Рассмотрим вводимый набор $a_1, a_2, a_3, \ldots, a_n$ длины n и обозначим через M_k значение переменной M после k-го прохода цикла.

1. Если мы вводим набор a_1 длины 1, то цикл сделает только один проход и M присвоится наибольшее значение из 0 и a_1, которым, очевидно, будет a_1 (натуральные числа больше 0). В этом простом случае вывод будет правильным.
2. Если после k-го прохода цикла M_k – наибольший элемент из набора $a_1, a_2, a_3, \ldots, a_k$, то после следующего прохода M_{k+1} будет равно $\max(M_k, a_{k+1})$, т.е. максимальному элементу набора $a_1, a_2, a_3, \ldots, a_k, a_{k+1}$.

В пункте 1 мы показали, что алгоритм работает правильно при любом вводимом наборе длины 1. Поэтому, согласно пункту 2, он будет правильно работать и на любом наборе длины 2. Вновь применяя пункт 2 рассуждений, мы убеждаемся, что алгоритм работает правильно и на любых наборах длины 3, и т.д. Таким образом, алгоритм правильно работает на любых наборах произвольной длины n, т.е. он корректен.

На формальном языке использованный метод доказательства выглядит следующим образом.

Пусть $P(n)$ – предикат, определенный для всех натуральных чисел n.

Если

1. $P(1)$ истинно и
2. $\forall\, k \geq 1$ импликация $(P(k) \Rightarrow P(k+1))$ верна.

Тогда $P(n)$ истинно при любом натуральном значении n.

Пример 2.11. Найти сумму первых n натуральных чисел.

Решение. Анализируя результат сложения первых двух, трех, четырех и т. д. натуральных чисел, мы можем предположить, что в общем случае

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

для всех натуральных n. Истинность этого предположения можно доказать методом математической индукции.

Пусть *P(n)* – предикат $1 + 2 + 3 + \cdots + n = \big(n(n+1)\big)/2.$

В случае n=1 левая и правая части равенства равны 1.

Следовательно, *P(1)* истинно.

Предположим теперь, что равенство $1 + 2 + 3 + \cdots + k = \big(k(k+1)\big)/2$ справедливо для какого-то натурального числа k. Тогда сумма первых $k+1$ членов натурального ряда равна:

$$1 + 2 + 3 + \cdots + k + (k+1) = \frac{k(k+1)}{2} + (k+1) =$$

$$= \frac{1}{2}\big(k(k+1) + 2(k+1)\big) = \frac{1}{2}(k+1)(k+2).$$

Таким образом, при любом натуральном k импликация $(P(k) \Rightarrow P(k+1))$ справедлива. Значит, по принципу математической индукции, предикат *P(n)* имеет истинное значение при всех натуральных n.

Пример 2.12. Методом математической индукции докажите, что $7^n - 1$ делится на 6 при любом натуральном показателе n.

Решение. Целое число a делится на целое число b тогда и только тогда, когда выполняется равенство $a = mb$ при каком-то целом числе m. Например, 51 делится на 17, поскольку 51= 3×17. Кроме того, известно простое свойство делимости чисел, которое утверждает, что сумма делящихся на b чисел также делится на b. Пусть *P(n)* обозначает предикат «$7^n - 1$ делится на 6». При n=1 имеем $7^n - 1 = 7 - 1 = 6$, т.е., предикат $P(1)$ имеет истинное значение. Первое требование метода математической индукции выполнено.

Предположим, что $7^k - 1$ делится на 6 при каком-то натуральном k. Тогда $7^{k+1} - 1 = 7(7^k - 1) + 7 - 1 = 7(7^k - 1) + 6$.

Так как 7^k–1 делится на 6 по нашему предположению, то по упомянутому выше свойству делимости сумма $7(7^k–1)+6$ тоже делится на 6.

Мы доказали, что если 7^k–1 делится на 6, то и 7^{k+1}– 1 также будет делиться на 6 при любом натуральном k, т.е., импликация $(P(k) \Rightarrow P(k + 1))$ истинна.

Таким образом, методом математической индукции мы доказали истинность предиката $P(n)$ для всех натуральных n.

Пример 2.13. Последовательность целых чисел $x_1, x_2, \ldots, x_n$ определена рекуррентной формулой $x_1 = 1$ и $x_{k+1} = x_k + 8k$ при $k \geq 1$.

Доказать, что для этой последовательности справедлива формула: $x_n = (2n – 1)^2$ для всех $n \geq 1$.

Решение. Предикат $x_n = (2n – 1)^2$ обозначим через $P(n)$. Если $n = 1$, то $(2n–1)^2 = (2–1)^2 = 1$, значит высказывание $P(1)$ истинно.

Допустим теперь, что $x_k = (2k – 1)^2$ для некоторого $k \geq 1$. Тогда

$x_{k+1} = x_k + 8k = (2k – 1)^2 + 8k = 4k^2 + 4k + 1 = (2k + 1)^2 = (2(k + 1) – 1)^2$.

Значит истинность импликации $(P(k) \Rightarrow P(k + 1))$ доказана при всех $k > 1$. Следовательно, согласно принципу математической индукции, предикат $P(n)$ превращается в истинное высказывание при любом натуральном значении переменной n, т.е., n-ый член последовательности может быть найден по формуле $x_n = (2n – 1)^2$.

2.4. Корректность алгоритмов

Чтобы доказать корректность алгоритма (иными словами, убедиться, что он делает именно то, что и предусмотрено), нам нужно проверить все изменения используемых в нем переменных *до, в течение* и *после* работы алгоритма. Эти изменения и условия можно рассматривать как небольшие утверждения или предикаты.

Пусть P – предикат, истинный для входных данных алгоритма A, и Q – предикат, описывающий условия, которым должны удовлетворять выходные

данные. Высказывание $\{P\}A\{Q\}$ означает, что «если работа алгоритма A начинается с истинного значения предиката P, то она закончится при истинном значении Q». Предикат P называется *входным условием*, или *предусловием*, а Q – *выходным условием,* или *постусловием.* Высказывание $\{P\}A\{Q\}$ само является предикатом. Поэтому доказательство корректности алгоритма A равносильно доказательству истинности $\{P\}A\{Q\}$. Для простых алгоритмов это делается достаточно прямолинейно.

Пример 2.14. Докажите корректность алгоритма ***Разность.***

```
Разность
begin
  z:=x–y;
end
```

Решение. В данном случае предусловием P являются равенства: $x = x_1$ и $y = y_1$. Постусловие Q – это $z = x_1–y_1$. Предикат *{P} Разность {Q}* читается как «если $x = x_1$ и $y = y_1$, то $z = x_1–y_1$». Истинность последнего предиката легко проверяется подстановкой $x = x_1$ и $y = y_1$ в тело алгоритма, содержащего переменные z, x и y. С формальной точки зрения соотношения: $z = x–y$, $x = x_1$ и $y = y_1$ приводят к тождеству $z = x_1–y_1$.

Когда в алгоритме A происходит много различных действий с переменными, мы разбиваем его на подходящие части A_1, A_2, ..., A_n и доказываем цепочку утверждений вида

$$\{P\}A_1\{Q_1\},\ \{Q_1\}A_2\{Q_2\},\ \ldots,\ \{Q_{n-1}\}A_n\{Q\},$$

где постусловие любого отрезка служит предусловием следующего.

Пример 2.15. Докажите правильность алгоритма ***Квадратный многочлен.***

```
Квадратный многочлен
{x – вещественное число}
begin
  y:=ax;
  y:= (y+b)x;
  y:= y+c;
end
  {y = ax²+bx+c}
```

Решение. Разобьем алгоритм на части, зафиксировав при этом обозначения пред- и постусловий.

```
      P→{x = x1}
      begin
        y:= ax;
Q1→           {y=ax1 и x=x1}
        y:= (y+b)x;
Q2→           {y=ax1²+bx1}
        y:= y+c;
      end
Q→    {y = ax1²+bx1+c}
```

Подстановки, сделанные выше, показывают, что все высказывания $\{P\}\ y{:=}ax\ \{Q_1\}$, $\{Q_1\}\ y{:=}(y+b)x\ \{Q_2\}$ и $\{Q_2\}\ y{:=}y+c\ \{Q\}$ верны. Следовательно, предикат $\{P\}$ ***Квадратный многочлен*** $\{Q\}$ истинен и алгоритм ***Квадратный многочлен*** корректен.

Алгоритм с условными высказываниями тоже должен быть доказан. Когда в алгоритме появляется условный оператор **if**... **then,** во входных и выходных условиях должны быть отражены альтернативные пути через весь алгоритм. Более точно: предположим, что условное составное высказывание

```
if условие then
  высказывание 1;
  else
  высказывание 2;
```

вводит предусловие P, а на выходе дает условие Q. Тогда следует доказать истинность обоих предикатов:

{P и условие} высказывание 1 {Q}

и

{P и не условие} высказывание 2 {Q}.

Пример 2.16. Докажите, что алгоритм ***Модуль*** корректен.

```
Модуль
{x – вещественное число}
begin
  if x≥0  then
        abc:= x;
        else
```

```
        abs := –x;
end
{abs – модуль числа x}
```

Решение. Предусловием P в нашем алгоритме служит $\{x = x_1\}$, а соответствующим постусловием Q является {*abs* – модуль числа x}.

Предикат $\{P$ **и** $x \geq 0\}$ ***abs***: = $x\{Q\}$ имеет истинное значение, поскольку модуль неотрицательного числа x_1 совпадает с ним самим.

Предикат $\{P$ **и не** $(x \geq 0)\}$ ***abs***:= $-x\{Q\}$ тоже истинен, т.к. модуль отрицательного числа x_1 отличается от него знаком.

Использование пред- и постусловий при проверке алгоритмов, в которых участвуют циклы типа **while ... do,** довольно громоздко. Предпочтительнее доказывать корректность таких алгоритмов методом математической индукции.

Пример 2.17. Докажите методом математической индукции корректность алгоритма ***Квадрат.***

```
Квадрат
{n – натуральное число}
begin
  sq:=0;
  for i:=1 to n do
        sq:=sq+2i–1;
end
{sq = n2}
```

Решение. Пусть $P(n)$ обозначает предикат «$sq=n^2$ после n-го прохода цикла», а sq_k – значение переменной sq после k-го прохода цикла.

Очевидно, что после первого прохода цикла $sq_1 = 1$ и первое требование метода математической индукции выполнено. Предположим, что после k-ой петли цикла $sq_k = k^2$. Тогда после следующего прохода

$sq_{k+1} = sq_k + 2(k+1)–1 = k^2 + 2k+1 = (k+1)^2$.

Таким образом, второе требование также выполняется.

Итак, мы установили, что $P(1)$ истинно. Кроме того, импликация $(P(k) \Rightarrow P(k+1))$ справедлива при любом $k \geq 1$. Следовательно, согласно

принципу математической индукции, *P(n)* истинно для всех натуральных *n*.

В рассмотренной задаче цикл **for** ограничен определенным числом итераций. В том случае, когда число петель цикла заранее не определено, как в цикле **while** ... **do,** при доказательстве методом математической индукции следует предположить, что число проходов ограничено, и показать правильность выходных данных. Затем необходимо доказать, что число петель такого цикла действительно конечно.

3. Теория множеств

В математике понятие множества является первичным и не имеет строгого определения. Множества используются для описания многих концепций не только в математике, но и в других науках, в том числе и в информатике. Элементы теории множеств изучались и применялись в различных разделах высшей математики. Мы рассмотрим только те свойства конечных множеств, которые будут необходимы при изучении нашего курса.

3.1. Множества и операции над ними

Множество – это совокупность различимых объектов, называемых *элементами* множества. Например,

- {Брест, Витебск, Гомель, Гродно, Минск, Могилев};
- {2, 3, 5, 7, 11};
- {сыр, яйцо, молоко, сметана}.

В этом примере множества заданы перечислением составляющих их элементов, которые принято заключать в фигурные скобки. Чтобы обеспечить возможность ссылок, мы будем обозначать множества прописными латинскими буквами. Например, $S = \{3, 2, 11, 5, 7\}$ – множество, содержащее данные элементы. Заметим, что множество S совпадает с одним из множеств, выписанных выше, поскольку порядок, в котором записываются элементы множества, значения не имеет.

Выражение $a \in S$ означает, что объект a является элементом множества S. Часто говорят, что a принадлежит множеству S. Если объект a не принадлежит S, то пишут: $a \notin S$.

Невозможно выписать все элементы очень больших, в особенности бесконечных множеств. В этом случае множества определяются с помощью подходящих предикатов. Запись $S = \{x: P(x)\}$ означает, что множество S состоит из таких элементов x, для которых предикат $P(x)$ имеет истинное

значение. Например, выражение $S = \{x: x$ – нечетное натуральное число$\}$ описывает множество $S = \{1, 3, 5, 7, ...\}$.

Поскольку любое натуральное нечетное число может быть представлено в виде $2n–1$, где n – любое натуральное число, альтернативное допустимое определение того же множества задается формулой

$S = \{2n–1: n$ – натуральное число$\}$.

Пример 3.1. Найдите более простое описание множеств, перечисляющее их элементы.

(а) $A = \{x: x$ – целое и $x^2 + 4x = 12\}$;

(б) $B = \{x: x$ – название дня недели, не содержащее буквы «е»$\}$;

(в) $C = \{n^2: n$ – натуральное$\}$.

Решение.

(а) Если $x^2+4x = 12$, то решение заключается в нахождении корней квадратного уравнения $x^2+4x–12 = 0$: $x = –6$ или $x = 2$. Следовательно, $A = \{–6, 2\}$;

(б) $B = \{$вторник, пятница, суббота$\}$;

(в) $C = \{0, 1, 4, 9, 16, ...\}$.

Некоторые часто используемые множества чисел имеют стандартные названия и обозначения:

$\varnothing$ – пустое множество;

$N = \{1, 2, 3, ...\}$ – множество натуральных чисел;

$Z = \{0, \pm1, \pm2, \pm3, ...\}$ – множество целых чисел;

$Q = \{p/q : p,q \in Z, q \neq 0\}$ – множество рациональных чисел;

$R = \{$все десятичные дроби$\}$ – множество вещественных чисел.

В современных языках программирования требуется, чтобы переменные объявлялись как принадлежащие к определенному типу данных. Тип данных представляет собой множество объектов со списком стандартных операций над ними. Определение типа переменных равносильно указанию множества,

из которого переменным присваиваются значения, что соответствует понятию домена в теории баз данныхю. Это мощный инструмент, обеспечивающий корректность обработки данных.

Существует несколько способов конструирования нового множества из двух данных множеств. Опишем основные интересующие нас операции на множествах. Прежде всего отметим, что в приведенных выше примерах все элементы некоторых множеств принадлежали другим, бо́льшим множествам. Например, все элементы множества $N = \{0, 1, 2, 3, ...\}$ содержатся в множестве $Z = \{0, \pm1, \pm2, \pm3, ...\}$, которое, в свою очередь, содержится в множестве $Q = \{p/q : p,q \in Z, q \neq 0\}$ и т.д.

Говорят, что множество A является *подмножеством* множества S, если каждый его элемент является элементом множества S. Часто при этом говорят, что множество A содержится в множестве S. Этот факт обозначают так: $A \subset S$. На рис. 3.1 дана диаграмма Венна, которая иллюстрирует это определение.

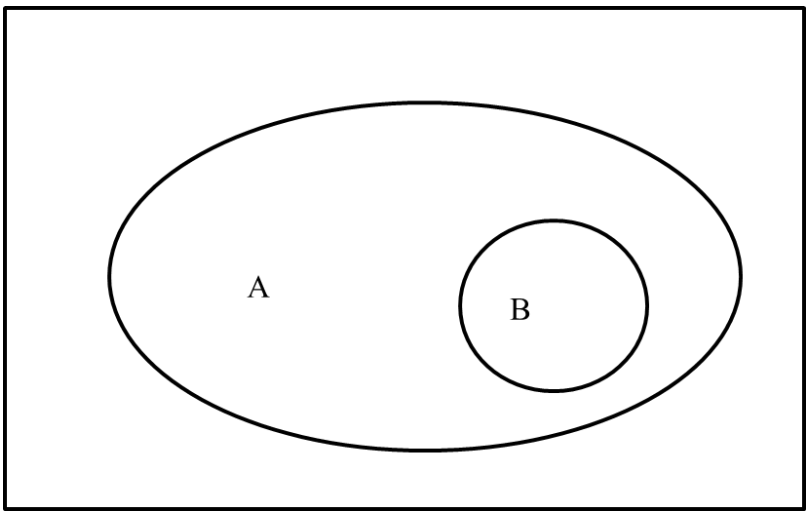


Рисунок 3.1. Диаграмма Венна подмножества A⊂S

Два множества считаются *равными*, если каждое из них содержится в другом. Поэтому для доказательства равенства множеств нам нужно показать, что они состоят из одних и тех же элементов. На формальном языке для равенства множеств $A=B$ необходимо проверить истинность двух импликаций: $\{x \in A \Rightarrow x \in B\}$ и $\{x \in B \Rightarrow x \in A\}$.

Пример 3.2. Пусть $A = \{n:\ n^2$ – нечетное целое число$\}$, $B = \{n:\ n$ – нечетное целое число$\}$. Показать, что $A = B$.

Решение. Если $x \in A$, то x^2 – нечетное целое число, что было доказано в примере 2.9. Отсюда вытекает, что само число x – целое и нечетное. Следовательно, $x \in B$, т.е. $A \subset B$.

С другой стороны, пусть $x \in B$. Тогда x – нечетное целое число. В этом случае x^2 тоже будет нечетным целым числом, а значит, $x \in A$. Ввиду произвольности взятого элемента $x \in B$ мы можем утверждать, что все элементы из B принадлежат A, т.е. $B \subset A$. Следовательно, множество A и множество B равны.

Объединением двух множеств A и B называется множество

$$A \cup B = \{x:\ x \in A \quad \textbf{или} \quad x \in B\}.$$

Оно состоит из тех элементов, которые принадлежат либо множеству A, либо множеству B, а возможно и обоим сразу. Диаграмма Венна объединения двух множеств показана на рис. 3.2.

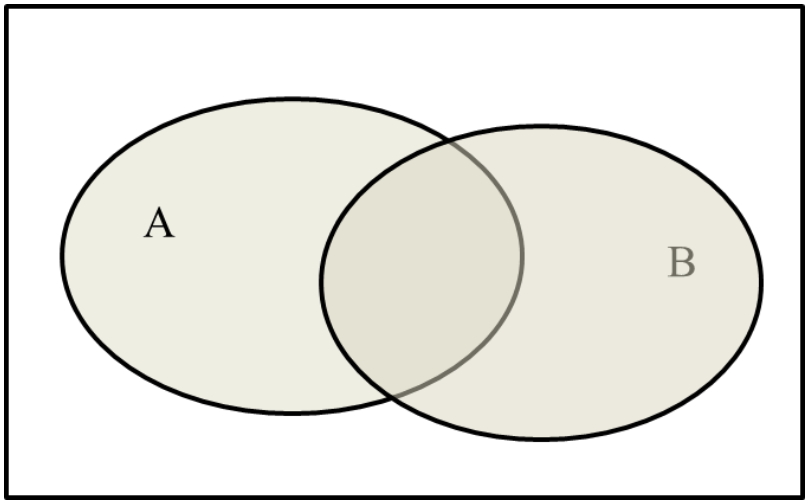


Рисунок 3.2 Диаграмма Венна объединения множеств $A \cup B$

Пересечением двух множеств A и B называется множество

$$A \cap B = \{x:\ x \in A \quad \textbf{и} \quad x \in B\}.$$

Оно состоит из элементов, которые принадлежат как множеству A, так и

множеству B. Диаграмма Венна пересечения двух множеств приведена на рис. 3.3.

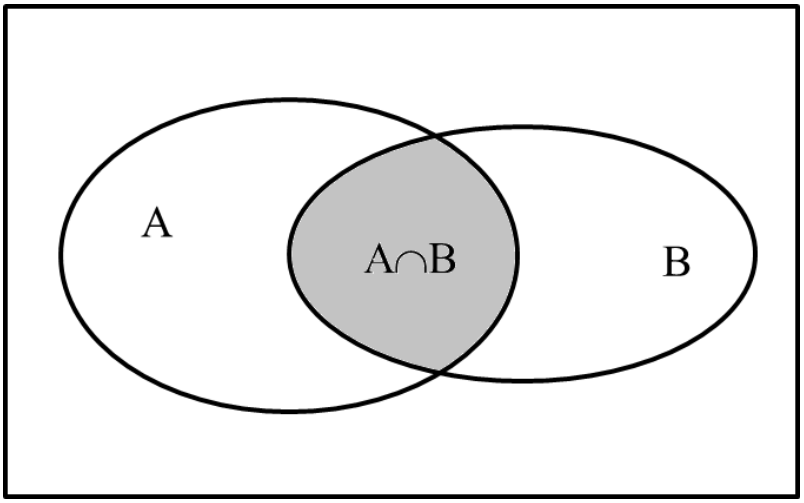


Рисунок 3.3 Диаграмма Венна пересечения множеств A∩B

Разностью множеств *A* и *B* или *дополнением* множества *B* до множества *A* называется множество

$$A \backslash B = \{ x : \ x \in A \ \ \textbf{и} \ \ x \notin B \} .$$

Разность множеств *A\B* состоит из всех элементов множества *A,* которые не принадлежат *B* (рис. 3.4).

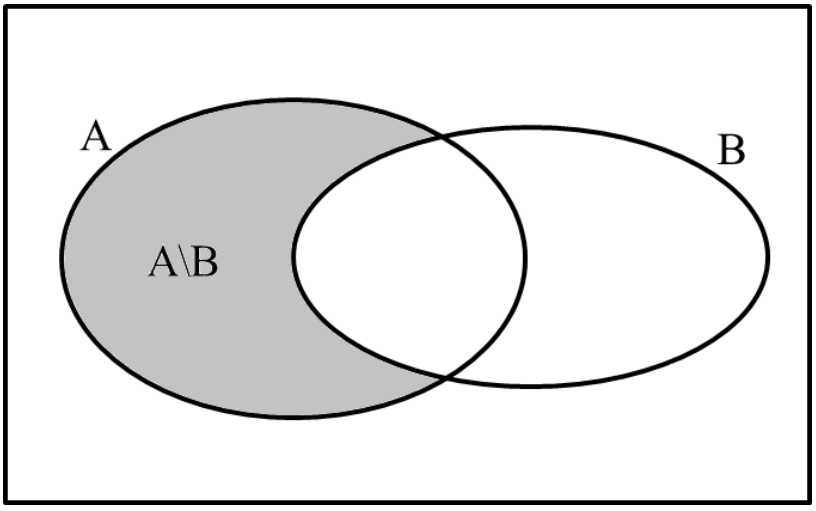


Рисунок 3.4 Диаграмма Венна разности множеств A\B

Если при решении задачи мы оперируем подмножествами некоего большого множества *U,* мы называем *U универсальным множеством* для данной задачи. На наших диаграммах Венна прямоугольник как раз и

символизирует это универсальное множество. Для подмножества A универсального множества U можно рассматривать дополнение A до U, т.е. $U\backslash A$. Поскольку в каждой конкретной задаче универсальное множество фиксировано, множество $U\backslash A$ обычно обозначают $\overline{A}$ и называют просто дополнением множества A. Таким образом, понимая, что мы работаем с подмножествами универсального множества U, можно записать

$$\overline{A} = \{x: \ \textbf{не} \ (x \in A)\} \quad \Leftrightarrow \quad \overline{A} = \{x: x \notin A\}.$$

Диаграмма Венна дополнения изображена на рис. 3.5.

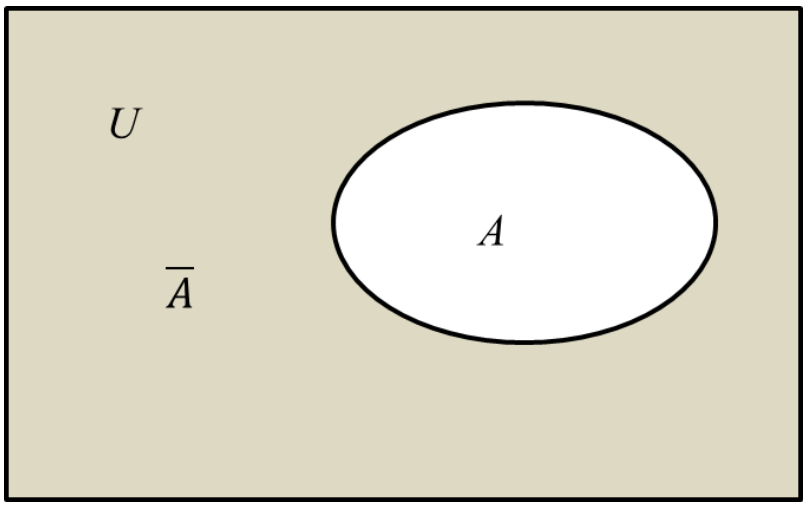


Рисунок 3.5 Диаграмма Венна дополнения $\overline{A}$

Симметрической разностью двух множеств A и B называют множество

$$A \Delta B = \{x: (x \in A \textbf{ и } x \notin B) \textbf{ или } (x \in B \textbf{ и } x \notin A).$$

Оно состоит из всех тех и только тех элементов универсального множества, которые либо принадлежат A и не принадлежат B, либо наоборот, принадлежат B, но не принадлежат A. Другими словами, симметрическая разность состоит из элементов, лежащих либо в A, либо в B, но не одновременно в обоих множествах.

Диаграмма Венна, иллюстрирующая симметрическую разность, показана на рис. 3.6.

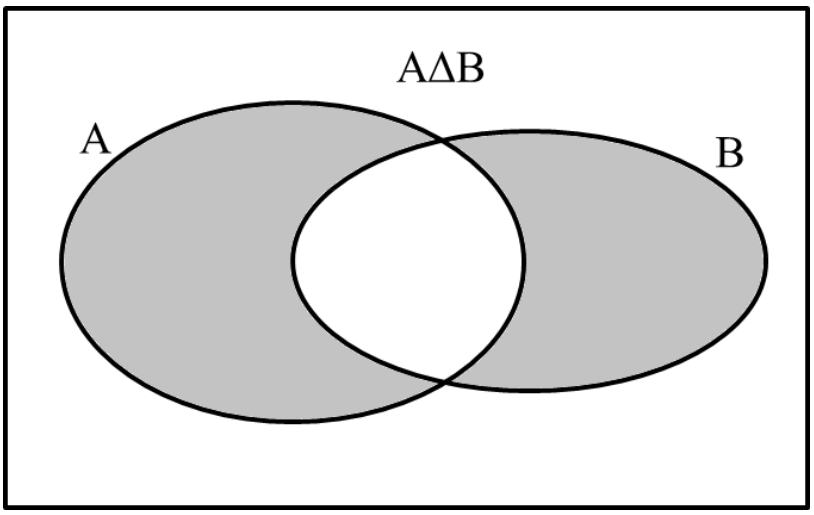


Рисунок 3.6 Диаграмма Венна симметрической разности AΔB множеств A и B

В определении симметрической разности мы привели формулу мы использовали предикат для описания множества. При решении задач часто удобнее использовать определение симметрической разности на языке множеств:

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

Пример 3.3. Пусть

$A = \{1, 3, 5, 7\}$; $B = \{2, 4, 6, 8\}$; $C = \{1, 2, 3, 4, 5\}$.

Найдите $A \cup C$, $B \cap C$, $A \setminus C$ и $B \Delta C$.

Решение.

$A \cup C = \{1, 3, 5, 7, 2, 4\}$;

$B \cap C = \{2, 4\}$;

$A \setminus C = \{7\}$;

$B \Delta C = (B \setminus C) \cup (C \setminus B) = \{6, 8\} \cup \{1, 3, 5\} = \{6, 8, 1, 3, 5\}$.

Пример 3.4. Пусть $A = \{x : 1 \leq x \leq 12$ и x четное целое число$\}$,

$B = \{x : 1 \leq x \leq 12$ и x целое число, кратное $3\}$.

Убедитесь, что $\left(\overline{A \cap B}\right) = \bar{A} \cup \bar{B}$.

Решение. Прежде всего заметим, что универсальным множеством в этой

задаче может быть множество чисел

U = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}.

Кроме того,

A = {2, 4, 6, 8, 10, 12} и B = {3, 6, 9, 12}.

Поэтому

$\left(\overline{A \cap B}\right) = \{\overline{6,12}\}$= {1, 2, 3, 4, 5, 7, 8, 9, 10, 11}

$\bar{A} \cup \bar{B}$= {1, 3, 5, 7, 9, 11} ∪ {1, 2, 4, 5, 7, 8, 10, 11} =

= {1, 2, 3, 4, 5, 7, 8, 9, 10, 11}.

Следовательно, $\left(\overline{A \cap B}\right) = \bar{A} \cup \bar{B}$.

Поскольку мы рассмотрели два произвольных множества A и B, то можно предположить, что такое равенство справедливо для любых множеств. Доказательство этого предположения рассмотрим в следующем примере.

3.2. Алгебра множеств

Из рассмотренных в предыдущем параграфе операций можно вывести полезные свойства множеств. Некоторые из них кажутся очевидными, другие – меньше, но все требуют доказательства. Доказательства будут основываться на соответствии между операциями на множествах и логическими операциями над предикатами, которое приведено в табл. 3.1 (строго говоря, это соответствие тоже необходимо обосновать).

Таблица 3.1. Соответствия между операциями на множествах и логическими операциями

Операции над множествами	Логические операции
$\overline{}$	не
∪	или
∩	и
⊂	⇒

Пример 3.5. Докажем, что сделанное в предыдущем примере предположение $\left(\overline{A \cap B}\right) = \bar{A} \cup \bar{B}$ справедливо для любых множеств A и B.

Решение.

$\left(\overline{A \cap B}\right)$= {x: x∉(A∩B)} = {x: **не** (x∈(A∩B))} = {x: **не** ((x∈A) **и** (x∈B))};

$\bar{A} \cup \bar{B}$ = {*x*: (x∉A) **или** (x∉B)} = {x: (**не** (x∈A)) **или** (**не** (x∈B))}.

Если обозначить простые высказывания (x∈A) как *P* и (x∈B) как *Q*, и учесть соответствие между логическими операциями и операциями над множествами (см. табл. 3.1), то можно легко увидеть, что предикат **не** (*P* **и** *Q*) соответствует множеству $\left(\overline{A \cap B}\right)$, а предикат (**не** *P*) **или** (**не** *Q*) – множеству $\bar{A} \cup \bar{B}$. Сравнивая таблицы истинности, легко установить логическую эквивалентность составных предикатов **не** (*P* **и** *Q*) и **(не** *P*) **или (не** *Q*) (табл. 3.2).

Таблица 3.2. Таблица истинности для примера 3.5

P	*Q*	*P* и *Q*	не (*P* и *Q*)	не *P*	не *Q*	(не *P*) или (не *Q*)
И	*И*	*И*	*Л*	*Л*	*Л*	*Л*
И	*Л*	*Л*	*И*	*Л*	*И*	*И*
Л	*И*	*Л*	*И*	*И*	*Л*	*И*
Л	*Л*	*Л*	*И*	*И*	*И*	*И*

Следовательно, $\left(\overline{A \cap B}\right) = \bar{A} \cup \bar{B}$.

Свойство, доказанное в примере 3.5, известно как один из законов де Моргана. Фундаментальные свойства, аналогичные законам де Моргана, составляют законы *алгебры множеств.* Эти свойства перечислены в табл. 3.3. Каждое из них может быть доказано с помощью логических аргументов, аналогичных тем, что использованы в примере 3.5.

Внимательное изучение свода законов алгебры множеств (табл. 3.3) позволяет заметить, что каждое из тождеств правой колонки может быть получено из соответствующего тождества левой путем замены ∪ на ∩, ∅ на *U* и наоборот. Такое соответствие тождеств называется *законом двойственности,* а соответствующие тождества – *двойственными* друг другу.

Закон двойственности является сложной теоремой алгебры множеств. Его доказательство выходит за рамки нашего курса. Однако, приняв его на веру, можно упростить себе жизнь, поскольку, доказав какое-то тождество

множеств и обратив операции, можно обосновать и двойственное тождество.

Таблица 3.3. Законы алгебры множеств

Законы ассоциативности	
$A\cup(B\cup C) = (A\cup B)\cup C$	$A\cap(B\cap C) = (A\cap B)\cap C$
Законы коммутативности	
$A\cup B = B\cup A$	$A\cap B = B\cap A$
Законы тождества	
$A\cup\varnothing = A$ $A\cup U = U$	$A\cap U = A$ $A\cap\varnothing = \varnothing$
Законы идемпотентности	
$A\cup A = A$	$A\cap A = A$
Законы дистрибутивности	
$A\cap(B\cup C) = (A\cap B)\cup(A\cap C)$	$A\cup(B\cap C) = (A\cup B)\cap(A\cup C)$
Законы дополнения	
$A\cup\bar{A} = U$ $\bar{U} = \emptyset$ $\bar{\bar{A}} = A$	$A\cap\bar{A} = \varnothing$ $\bar{\varnothing} = U$ $\bar{\bar{A}} = A$
Законы де Моргана	
$\overline{(A\cup B)} = \bar{A}\cap\bar{B}$	$\overline{(A\cap B)} = \bar{A}\cup\bar{B}$

Пример 3.6. Используя законы алгебры множеств, докажите, что произвольные множества A и B удовлетворяют свойству:

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

Решение. Определение симметрической разности множеств можно записать следующим образом: $A\Delta B = (A\cap\bar{B})\cup(B\cap\bar{A})$.

Согласно законам алгебры множеств, преобразуем правую часть доказываемого равенства:

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

$= (A\cup B)\cap(\bar{A}\cup\bar{B})$= (закон дистрибутивности) =

$= ((A\cup B)\cap\bar{A})\cup((A\cup B)\cap\bar{B})$ = (закон коммутативности) =

$= (\bar{A}\cap(A\cup B))\cup(\bar{B}\cap(A\cup B))$ = (закон дистрибутивности) =

$= ((\bar{A}\cap A)\cup(\bar{A}\cap B))\cup((\bar{B}\cap A)\cup(\bar{B}\cap B))$ = (закон коммутативности) =

$= ((A\cap\bar{A})\cup(B\cap\bar{A}))\cup((A\cap\bar{B})\cup(B\cap)\bar{B})$ = (закон дополнения) =

$= \varnothing \cup (B \cap \bar{A})) \cup ((A \cap \bar{B}) \cup \varnothing)$ = (законы коммутативности и тождества) =

$= (A \cap \bar{B}) \cup (B \cap \bar{A}) = A \Delta B$.

Следовательно, $A \Delta B = (A \cup B) \cap \overline{(A \cap B)}$.

В этом примере мы использовали законы алгебры множеств для того, чтобы доказать справедливость тождества для любых исходных множеств. Эти же законы можно использовать для решения уравнений на множествах.

Пусть задано равенство двух множеств, определенных формулами, которое устанавливает отношения между входящими в формулы множествами. Решение уравнения сводится к выяснению этих отношений в терминах взаимного включения множеств (их взаимного расположения). Решение основано на соотношениях типа:

- если $A \cup B \cup C = \varnothing$, то $A = \varnothing$, $B = \varnothing$ и $C = \varnothing$;
- если $A \cap B = \varnothing$, то $A \subset$ или $B \subset \bar{A}$.

Если уравнение записано в виде $A=B$, где оба множества не пустые, то его нужно привести к виду, когда справа будет $\varnothing$. Учитывая, что симметрическая разность равных множеств есть пустое множество, то уравнение вида $A=B$ можно заменить на уравнение $(A \cap \bar{B}) \cup (\bar{A} \cap B) = \varnothing$.

Пример 3.7. Уравнение $(\bar{A} \cap B) \cup (A \cap \bar{B} \cap C) = \varnothing$. Определить взаимные включения множеств A, B и C.

Решение. Очевидно, что $\overline{A} \cap B = \varnothing$ и $A \cap \bar{B} \cap C = \varnothing$. Следовательно, $B \subset A$. $A \cap \bar{B} \cap C = (A \cap C) \cap \bar{B} = \varnothing \Rightarrow (A \cap C) \subset B$. Так как $B \subset A$, то $C \subset B$.

Взаимное соотношение множеств A, B и C показано на рис. 3.7.

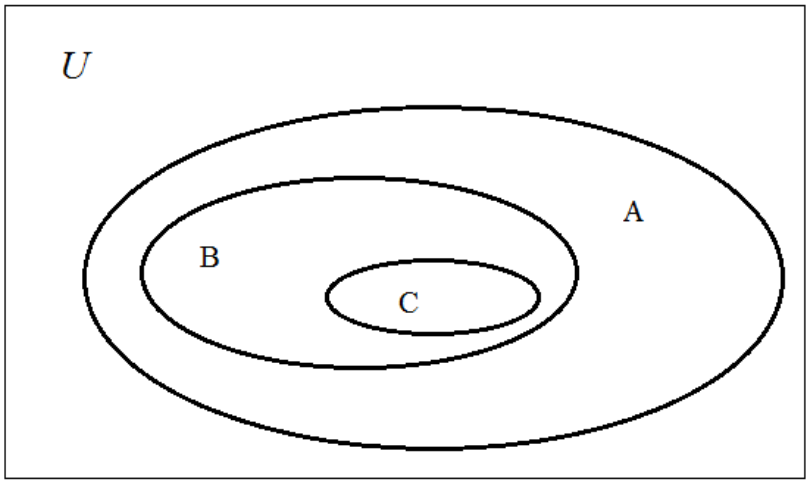


Рисунок 3.7. Диаграмма Венна взаимного включения множеств A, B и C для примера 3.7.

3.3. Дополнительные свойства множеств

В информатике часто возникает потребность подсчета количества данных. Вопросы пересчета становятся особенно важными, когда ограничены ресурсы. Например, сколько пользователей может поддерживать данная компьютерная сеть? Или сколько операций будет сделано при работе данного алгоритма?

Мощностью конечного множества S называется число его элементов. Она обозначается символом $|S|$.

Следующая теорема дает простое правило вычисления мощности объединения двух множеств.

Формула включений и исключений

Мощность объединения двух множеств равна

$$|A \cup B| = |A| + |B| - |A \cap B|.$$

Доказательство. Как показано на рис. 3.8, множество $A \cup B$ состоит из подмножеств $A \setminus B$, $A \cap B$ и $B \setminus A$, которые не имеют общих элементов. Более того, $A = (A \setminus B) \cup (A \cap B)$ и $B = (B \setminus A) \cup (A \cap B)$.

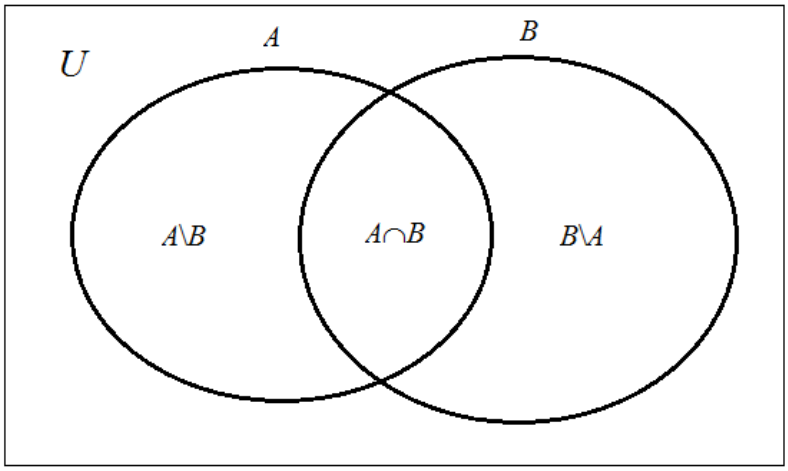


Рисунок 3.8. Диаграмма Венна для доказательства формулы включений
и исключений для случая двух множеств

Введем обозначения: $|A \setminus B| = m$, $|A \cap B| = n$, $|B \setminus A| = p$. Тогда

$|A| = m+n$, $|B| = n+p$ и

$|A \cup B| = m+n+p = (m+n) + (n+p) - n = |A|+|B|-|A \cap B|$.

Пример 3.8. Каждый из 48 студентов второго курса специальности ИСТ может посещать и дополнительные лекции. Если 16 из них слушают курс менеждмента, 25 – курс радиоэкологии и 5 изучают обе эти дисциплины, то сколько студентов вообще не посещают упомянутых дополнительных занятий?

Решение. Введем обозначения множеств.

A = {студенты, слушающие курс менеджмента};

B = {студенты, слушающие курс радиоэкологии}.

Тогда $|A| = 16$, $|B| = 25$, $|A \cap B| = 5$.

Поэтому $|A \cup B| = 16 + 25 - 5 = 36$.

Следовательно, 48–36 = 12 студентов не посещают дополнительных курсов.

Формула для подсчета мощности объединения двух множеств может быть обобщена на произвольное число множеств и доказана методом

математической индукции.

При обсуждении конечных множеств порядок, в котором перечисляются их элементы, значения не имеет. Однако бывает необходимо работать и с упорядоченными наборами.

Упорядоченной парой называется запись вида (a, b), где a – элемент некоторого множества A, b – элемент множества B. Множество всех таких упорядоченных пар называется декартовым или прямым произведением множеств A и B, и обозначается A×B:

$$A\times B=\{(a,b):a\in A \text{ и } b\in B\}.$$

Мы рассмотрели еще одну операцию на множествах. Декартово произведение множеств имеет практическое значение, поскольку вплотную подводит нас к понятиям «отношение» и «функция», играющим важную роль в информатике и составляющим предмет изучения последующих лекций.

Пример 3.9. Пусть $A = \{x, y\}$ и $B = \{1, 2, 3\}$. Найдите декартовы произведения: $A\times B$, $B\times A$ и $B\times B$.

Решение. Декартовым (или прямым) произведением $A\times B$ является множество

$\{(x, 1), (x, 2), (x, 3), (y, 1), (y, 2), (y, 3)\}$.

Декартово произведение $B\times A$ – это множество

$\{(1, x), (2, x), (3, x), (1, y), (2, y), (3, y)\}$.

Заметим, что множества $A\times B$ и $B\times A$ различны!

Декартово произведение $B\times B$ – это множество

$\{(1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (2, 3), (3, 1), (3, 2), (3, 3)\}$.

Основываясь на примере 3.9, можно предположить, что мощность прямого произведения конечных множеств A и B равна $|A\times B|=mn$, если $|A|=m$ и $|B|=n$, поскольку каждый элемент множества A (а их ровно m штук) участвует ровно в n различных упорядоченных парах.

Если же одно из множеств или сразу оба бесконечны, то и произведение будет иметь бесконечное число упорядоченных пар.

В качестве примера рассмотрим прямое произведение множества R вещественных чисел на само себя. Множество $R \times R$, или R^2, как его часто обозначают, состоит из всех упорядоченных пар вещественных чисел (x, y). Их можно представлять как координаты точек на плоскости. Множество R^2 называется *декартовой плоскостью*, которая изображена на рис. 3.9.

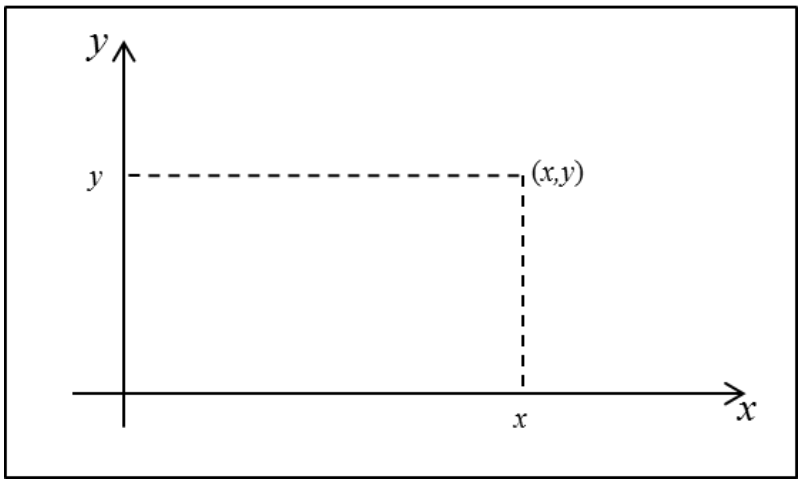


Рисунок 3.9. Декартова плоскость, состоящая из элементов множества R^2

Декартовым произведением произвольного числа множеств $A_1, A_2, \ldots, A_n$ называется множество

$$A_1 \times A_2 \times \ldots \times A_n = \{(a_1, a_2, \ldots, a_n) : a_i \in A_i,\ i = 1, 2, \ldots, n\}.$$

Элементы этого множества, которые называются кортежами – это конечные упорядоченные наборы размером n, число которых равно произведению мощностей A_i. Это объекты, с которыми работают все языки программирования. Подмножества декартовых произведений также представляют собой объект обработки в базах данных и других приложениях информатики.

В том случае, когда каждое из множеств $A_1, A_2, \ldots, A_n$ совпадает с множеством A, то прямое произведение n экземпляров множества A обозначается как A^n. Если A содержит m элементов, то мощность множества A^n равна m^n.

Пример 3.10. Пусть $B = \{0, 1\}$. Опишите множество B^n.

Решение. Множество B^n состоит из набора последовательностей нулей и

единиц длины n.

$B^2 = \{(0,0), (0,1), (1,0), (1,1)\}$;

$B^3 = \{(0,0,0), (0,0,1), (0,1,0), (0,1,1), (1,0,0), (1,0,1), (1,1,0), (1,1,1)\}$;

и т.д.

Эти последовательности называются *строкой бит*, или *битовой строкой* длины n.

Покажем, как строка бит применяется для моделирования операций на конечных множествах. Пусть $S = \{s_1, s_2, ..., s_n\}$. Если $A \subset S$, мы поставим ему в соответствие n-битную строку $(b_1, b_2, ..., b_n)$, где $b_i = 1$, если $s_i \in A$ и $b_i = 0$ в противном случае. Такая строка бит называется *характеристическим вектором* подмножества A. Если S универсальное множество для некоторой задачи, то можно записать характеристические векторы для всех рассматриваемых в задаче подмножеств и использовать их для вычислений. Мы можем имитировать операции на множествах логическими операциями, применяемыми к соответствующим характеристическим векторам, условившись считать 1 за истину (И), а 0 за ложь (Л).

Пример 3.11. Пусть $S = \{1, 2, 3, 4, 5\}$; $A = \{1, 3, 5\}$ и $B = \{3, 4\}$. Выписать характеристические векторы множеств A и B, а затем определить характеристические векторы множеств $A \cup B$, $A \cap B$ и $\bar{B}$ и перечислить их элементы.

Решение. Характеристические векторы, построенные на множестве S, представляют собой битовые строки длиной 5. Характеристический вектор для множества A – битовая строка a=(1, 0, 1, 0, 1). Характеристический вектор множества B – b=(0, 0, 1, 1, 0). Для расчета множеств $A \cup B$, $A \cap B$ и $\bar{B}$ используем соответствующие логические операторы (см. табл. 3.1) и находим:

a **или** b = (1, 0, 1, 0, 1) **или** (0, 0, 1, 1, 0) = (1, 0, 1, 1, 1);

a **и** b = (1, 0, 1, 0, 1) **и** (0, 0, 1, 1, 0) = (0, 0, 1, 0, 0);

не b = **не** (0,0, 1, 1, 0) = (1, 1, 0, 0, 1).

Полученные векторы позволяют нам легко прочитать элементы

требуемых подмножеств: $A \cup B = \{1, 3, 4, 5\}$, $A \cap B = \{3\}$ и $\bar{B} = \{1, 2, 5\}$.

3.4. Базы знаний

Экспертная система создается для того, чтобы подменить собой специалистов в какой-то предметной области. Это достигается путем создания *баз знаний*, которые сочетают базы данных с накопленными известными фактами и наборы *правил вывода*. Правила вывода позволяют получать логическим путем дополнительную информацию из имеющихся фактов.

Мы построим простую экспертную систему «Королевская династия» для ответа на вопросы об английских королях и королевах и их семьях, начиная с Георга I. Прежде всего, мы подготовим список фактов, используя предикаты *родитель* и *жена* (табл. 3.4). Условимся, что предикат *родитель(x,y)* означает, что *x* является родителем *y*, а *жена(x,y)* означает, что *x* – жена *y*. Это стандартное чтение предикатов, используемых языками программирования.

Таблица 3.4. Пример базы знаний «Королевская династия»

Предикат *родитель(x,y)*	***Предикат жена(x,y)***
родитель(Георг I, Георг II)	жена(Вильгельмина, Георг II)
родитель(Георг III, Вильгельм IV)	жена(Шарлотта, Георг III)
родитель(Георг III, Эдвард)	жена(Каролина, Георг IV)
родитель(Эдвард, Виктория)	жена(Аделаида, Вильгельм IV)
родитель(Виктория, Эдвард VII)	жена(Виктория, Альберт)
родитель(Эдвард VII, Георг V)	жена(Александра, Эдвард VII)
родитель(Георг V, Эдвард VIII)	жена(Виктория Мари, Георг V)
родитель(Георг V, Георг VI)	жена(Елизавета, Георг VI)
родитель(Георг VI, Елизавета II)	жена(Елизавета II, Филлип)
родитель(Виктория, Элис)	
родитель(Элис, Виктория Альберта)	
родитель(Виктория Альберта, Элис-Моунтбаттен)	
родитель(Элис-Моунтбаттен, Филипп)	

Чтобы извлечь информацию, мы будем ставить вопросы перед базой данных. Например, если мы спрашиваем: «Является ли Георг I отцом Георга III?», то ответ будет отрицательным, поскольку предикат *родитель*(Георг I, Георг III) отсутствует в нашем списке фактов.

Договоримся запросы записывать в виде: «? – предикат». Кроме того,

будем предполагать, что наличие переменной в предикате равносильно вопросу о существовании. Например, запрос «? – *жена(x*, Георг IV) понимается как «была ли жена у Георга IV?». В этом случае ответ положителен, так как, заменяя *x* на «Каролина», мы получим высказывание, присутствующее в списке фактов.

Задача 1. Найдите ответы на следующие запросы:

(а) ? – *жена*(Елизавета II, Филипп);

(б) ? – *родитель*(Шарлотта, Эдвард);

(в) ? – *женщина*(Каролина);

(г) ? – *жена*(Филипп, Елизавета II).

Решение. Положительный ответ будет выдан только на первый запрос, так как только для него в списке фактов есть соответствующий предикат. В то же время, в запросе (б) ответ тоже должен быть положительным, так из таблицы видно, что Шарлотта – жена Георга III, который является отцом Эдварда, но предикат *родитель*(Шарлотта, Эдвард) в базе данных отсутствует. Напомним, что отрицательный ответ на запрос дается в том случае, если список фактов не содержит предиката из запроса.

Чтобы система с базой знаний могла решать более сложные задачи, мы введем так называемые *правила вывода*. Правило вывода определяет новый предикат в терминах, которые присутствуют в исходном списке фактов. Ответы на запросы о новых предикатах могут быть логически выведены из списка фактов, генерируя, таким образом, новую информацию.

В системе «Королевская династия» кажется очевидным, что переменная *x,* попавшая в предикат *жена*(*x, y*), соответствует женщине. Определим новый предикат, который будет означать, что «если *x* – жена *y, то x* – женщина»:

(1) женщина(x) **from** жена(x, y).

Аналогично введем правило, определяющее предикат *муж*. Он означает, что «если *x* – жена *y, то y* – муж *x*»:

(2) муж(y, x) from жена(x, y).

Теперь на запрос (в) будет дан положительный ответ согласно правилу (1), примененному к исходным данным *жена*(Каролина, Георг IV).

Задача 2. Найдите ответы на дополнительные запросы:

(д) ? – *женщина*(Эллис-Моунтбаттен);

(е) ? – *муж*(Альберт, Виктория);

(ж) ? – *мужчина*(Альберт).

Решение. На запрос (д) ответ будет отрицателен, так как Элис-Моунтбаттен в основном списке не упомянута в качестве чьей-либо жены.

Ответ в случае (е) – положителен, ввиду наличия в списке предиката *жена*(Виктория, Альберт) и правила (2). Отрицательный ответ будет дан на запрос (ж), т.к. предикат *мужчина* пока еще не определен.

Подходящее правило вывода, дающее информацию о принадлежности к мужской половине человечества, аналогично правилу (1):

(3) мужчина(y) **from** жена(x, y).

Можно сформулировать правило, представляющее информацию о сыновьях:

(4) сын(x, y) **from** {мужчина(x) **и** родитель(y, x)).

Задача 3. Ответьте на следующие запросы:

(з) ? – мужчина(Вильгельм IV);

(и) ? – *сын*(Вильгельм IV, Георг III);

(к) ? – *сын*(Вильгельм IV, Шарлотта);

(л) ? – *сын*(Эдвард VIII, Георг V).

Решение.

(з) Положительный ответ следует из предиката *жена*(Аделаида, Вильгельм IV) по правилу (3).

(и) Положительный ответ следует из предиката *родитель*(Георг III, Вильгельм IV) ввиду положительного ответа на запрос (з) и правила вывода (4).

Ответы на запросы (к) и (л) – отрицательны.

Обратите внимание, что при ответах на запросы (к) и (л) необходимо твердо придерживаться фактов и правил вывода ввиду ограничений,

наложенных на систему. Только монархи считаются родителями, в то время как их супруги появляются только в предикате *жена.* Так, хотя Шарлотта была замужем за Георгом III, и Вильгельм IV – один из их сыновей, база данных считает его родителем только Георга III. Следовательно, правило вывода (4) не может дать положительный ответ на запрос (к). Причина отрицательного ответа в случае (л) заключается в том, что в списке отсутствует жена у Эдварда VIII. Поэтому правило (3) дает ответ «Нет» на запрос «? – *мужчина*(Эдвард VIII)».

Как мы увидели, в случае неполной информации, содержащейся в системе данных, как это часто бывает в реальных экспертных системах, отрицательный ответ на запрос может означать, что нам просто ничего не известно. Должное внимание к формулировке правил вывода и выбору исходных предикатов базы данных может частично решить эту проблему. К сожалению, при неопределенности отрицательных ответов мы не можем полностью доверять и положительным, если в предикатах участвует операция отрицания **не**.

Задача 4. Сформулируйте правило вывода для извлечения информации о матерях из экспертной системы «Королевская династия». Определите правило *мать*(*x*) так, чтобы положительный ответ на соответствующий запрос выдавался в том случае, если *x* – жена чьего-то родителя или *x* – женщина и чей-то родитель. Примените новое правило совместно с правилом (1) к исходной базе данных для определения максимально возможного числа матерей. Удовлетворительным ли получилось новое правило вывода?

Решение. Требуемое правило вывода может быть определено так: мать(x) from([жена(x, y) и родитель(y, z)] или [женщина(x) и родитель(x, y)]).

Часть [*жена*(*x, y*) **и** *родитель*(*y, z*)] нашего правила определит как «мать» следующих королев: Александра, Шарлотта, Елизавета, София и Виктория Мари. Вторая часть [*женщина*(*x*) **и** *родитель*(*x, y*)] выявит Викторию, но Элис, Виктория Альберта и Элис-Моунтбаттен, которые также являются матерями,

найдены не будут, т.к. хранящаяся в базе информация недостаточна при имеющихся правилах вывода.

Первая часть правила будет считать матерями и мачех. Кроме того, проблема будет возникать и тогда, когда у монарха было несколько жен. Это показывает трудности, возникающие при попытке ограничить реальный мир рамками простой математической модели.

4. Отношения

В математике отношения – это множество, построенное на элементах одного или нескольких множеств. Элементами отношения являются упорядоченные наборы элементов исходных множеств, удовлетворяющих заданному условию. Такие упорядоченные наборы называются кортежами отношения (см. параграф 3.3).

Когда говорят о родстве двух людей, например Яся и Янины, то подразумевают, что есть некая семья, к членам которой они относятся. Упорядоченная пара (Ясь, Янина) отличается от других упорядоченных пар людей тем, что между Ясем и Яниной есть некое родство (муж, кузина, отец, и т. д.). Более того, есть отличие даже от упорядоченной пары (Янина, Ясь). Например, в первом случае степень родства – муж, во втором – жена. Если из рассматриваемого множества людей M выбрать все упорядоченные пары (x,y) такие, что x является мужем y, то говорят, что построено отношение {x – муж y на множестве людей M}.

В математике если строится отношение между элементами множеств A и B, то среди всех упорядоченных пар прямого произведения $A \times B$ двух множеств тоже выделяются некоторые пары такие, что между их компонентами есть некоторые «родственные» отношения, которых нет у других.

В качестве примера рассмотрим множество S студентов университета и множество K имеющихся специальностей. В прямом произведении $S \times K$ можно выделить большое подмножество упорядоченных пар (s,k), обладающих свойством: студент s изучает специальность k. Построенное подмножество отражает отношение «s изучает специальность k», естественно возникающее между множествами студентов и специальностей.

Для математического описания любых связей между элементами двух множеств вводится понятие бинарного отношения. Мы ограничимся

изучением именно бинарных отношений. Будут рассмотрены различные способы определения отношений и изучены их свойства, которые интересны с точки зрения обработки массивов данных.

4.1. Бинарные отношения

Бинарным отношением между множествами A и B называется подмножество R прямого произведения $A \times B$. В том случае, когда $A=B$, мы говорим просто о бинарном отношении R на множестве A, т.е. R – подмножество упорядоченных пар, состоящих из элементов множества A.

Напомним, что декартовым (прямым) произведением множества A на множество B называется множество $A \times B$ всевозможных пар (a,b), где $a \in A$ и $b \in B$.

Пример 4.1. Рассмотрим генеалогическое древо, изображенное на рис. 4.1. Выпишем упорядоченные пары, находящиеся в следующих отношениях на множестве P членов этой семьи:

(а) $R = \{(x,y)$: x – *дедушка* $y\}$;

(б) $S = \{(x,y)$: x – *сестра* $y\}$.

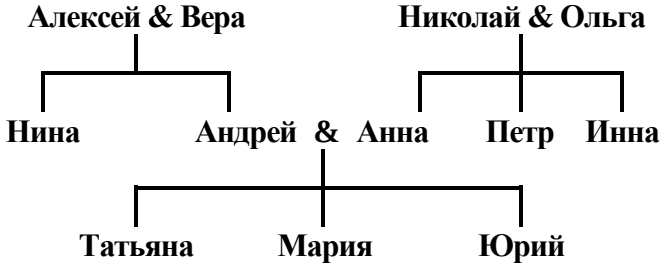


Рис. 4.1. Генеалогическое дерево

Решение. (а). R содержит упорядоченные пары: (Алексей, Татьяна), (Алексей, Мария), (Алексей, Юрий), (Николай, Татьяна), (Николай, Мария) и (Николай, Юрий).

(б). S состоит из 9 пар: (Анна, Петр), (Анна, Инна), (Инна, Анна), (Инна, Петр), (Татьяна, Мария), (Татьяна, Юрий), (Мария, Татьяна), (Мария, Юрий) и (Нина, Андрей).

1

Пример 4.2. Выпишите упорядоченные пары, принадлежащие следующим бинарным отношениям на множествах:

$A = \{1, 3, 5, 7\}$ и $B = \{2, 4, 6\}$:

(а) $U = \{(x,y): x + y = 9\}$;

(б) $V = \{(x,y): x < y\}$.

Решение.

(а) U состоит из пар: (3, 6), (5, 4) и (7, 2);

(б) $V = \{(1, 2), (1, 4), (1, 6), (3, 4), (3, 6), (5, 6)\}$.

Пример 4.3. Множество $R = \{(x, y) : x$ – делитель $y\}$ определяет отношение на множестве $A = \{1, 2, 3, 4, 5, 6\}$. Найдите все упорядоченные пары, ему принадлежащие.

Решение. R состоит из пар: (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (2, 2), (2, 4), (2, 6), (3, 3), (3, 6), (4, 4), (5, 5) и (6, 6).

В рассмотренных примерах отношения задавались путем перечисления принадлежащих им упорядоченных пар. Теперь познакомимся с двумя более удобными способами представления упорядоченных пар, принадлежащих данному отношению. Первый из них основан на понятии «ориентированный граф», а второй использует матрицы.

Пусть A и B – два конечных множества и R – бинарное отношение между ними. Мы изобразим элементы этих множеств точками на плоскости.

Для каждой упорядоченной пары отношения R нарисуем стрелку, соединяющую точки, представляющие компоненты пары. Такой графический объект называется *ориентированным графом*, или *орграфом*. В нем, в отличие от неориентированных графов, которые мы использовали в §1.1, ребра имеют направления и называются дугами.

В качестве иллюстрации рассмотрим отношение V между множествами $A = \{1, 3, 5, 7\}$ и $B = \{2, 4, 6\}$ из примера 4.2 (б). Соответствующий ориентированный граф показан на рис. 4.2.

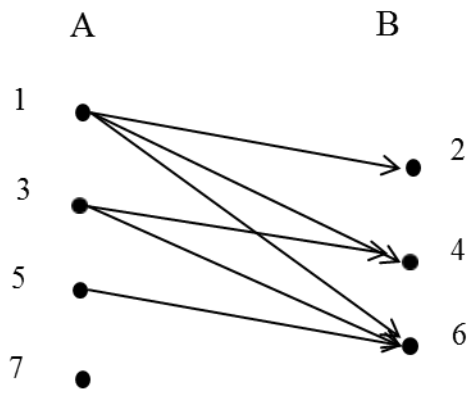


Рисунок 4.2. Отношение *V* между элементами множеств *A* и *B* из примера 4.2.

Еще один способ задания бинарного отношения на конечных множествах основан на использовании таблиц. Предположим, что мы хотим определить бинарное отношение *R* между множествами *A* и *B*:

$$A = \{a_1, a_2, \ldots, a_n\}, \qquad B = \{b_1, b_2, \ldots, b_m\}.$$

Для определения отношения *R* заполним таблицу *M* с *n* строками и *m* столбцами. Строки «перенумеруем» элементами множества *A*, а столбцы – элементами множества *B* в соответствии с порядком, в котором записаны элементы. Ячейку таблицы, стоящую на пересечении *i*-той строки и *j*-того столбца будем обозначать через *M(i, j)*, а заполнять ее будем по правилу:

$$M(i, j) = И, \text{ если } (a_i, b_j) \in R, \text{ и } M(i, j) = Л, \text{ если } (a_i, b_j) \notin R.$$

Такого сорта таблицы называются $n \times m$ *матрицами.*

В этих терминах отношение *V* из примера 4.2(б) с помощью матрицы задается следующим образом:

	2	4	6
1	*И*	*И*	*И*
3	*Л*	*И*	*И*
5	*Л*	*Л*	*И*
7	*Л*	*Л*	*Л*

Чтобы лучше понять такой способ задания отношений, мы явно пометили столбцы и строки матрицы элементами множеств *A* и *B*. В общем случае это делать не обязательно.

Пример 4.4. Отношение R на множестве $A = \{a,b,c,d\}$ задается матрицей, порядок строк и столбцов в которой соответствует порядку выписанных элементов множества A. Назовите упорядоченные пары, принадлежащие R.

$$\begin{bmatrix} Л & И & И & Л \\ Л & Л & И & И \\ Л & И & Л & Л \\ И & И & Л & И \end{bmatrix}$$

Решение. Отношение R содержит упорядоченные пары: (a, b), (a, c), (b, c), (b, d), (c, b), (d, a), (d, b) и (d, d).

Если R – бинарное отношение, то вместо записи $(x, y) \in R$ можно употреблять обозначение xRy. Например, предикат «x – сестра y» определяет отношение на множестве всех людей. В примере 4.3 предикат «x – делитель y» дает ясное словесное описание еще одного отношения.

Итак, мы рассмотрели четыре способа записи бинарного отношения между конечными множествами:

- словами (с помощью подходящих предикатов);
- как множество упорядоченных пар;
- как орграф;
- как матрица.

4.2. Свойства бинарных отношений

Ограничимся рассмотрением бинарных отношений, заданных на одном множестве, и рассмотрим некоторые свойства. Говорят, что отношение R на множестве A:

рефлексивно, если для всех $x \in A$ xRx;

симметрично, если $xRy \Rightarrow yRx$ для каждой пары x и y из A;

кососимметрично, если (xRy и $yRx \Rightarrow x=y$) для всех x и y из A;

транзитивно, если (xRy и $yRz \Rightarrow xRz$) для любой тройки элементов

$(x, y, z) \in A$.

В терминах упорядоченных пар эти свойства определяются следующим образом. Данное отношение R рефлексивно, если $(x, x) \in R$ для любого возможного значения переменной x; симметрично, если из включения $(x, y) \in R$ следует, что $(y, x) \in R$; кососимметрично, если из предположений $(x, y) \in R$ и $x \neq y$ вытекает, что $(y, x) \notin R$; транзитивно, если включения $(x, y) \in R$ и $(y, z) \in R$ влекут $(x, z) \in R$.

У ориентированного графа, изображающего рефлексивное отношение, каждая вершина снабжена петлей, т.е. стрелкой, начинающейся и заканчивающейся в одной и той же вершине. Орграф симметричного отношения вместе с каждой стрелкой из вершины x в вершину y имеет стрелку, направленную в обратную сторону: из y в x. Если отношение кососимметрично, то при наличии стрелки из вершины x в несовпадающую с ней вершину y стрелка из y в x будет обязательно отсутствовать. И, наконец, орграф транзитивного отношения устроен так, что вместе со стрелками из вершины x в y и из y в z у него будет стрелка и из x в z.

Перечислим свойства матриц, задающих отношения. Прежде всего заметим, что матрица отношения на отдельном множестве A будет квадратной, т.е. количество ее строк будет равно количеству столбцов.

Матрица M, задающая рефлексивное отношение, отличается от других тем, что каждый ее элемент, стоящий на главной диагонали ($M(i,i)$), равен $И$, или, другими словами, единичная матрица $Id_M \in M$.

Матрица M симметричного отношения будет симметричной относительно главной диагонали, т.е., $M(i,j) = M(j,i)$, значит матрица смежности равна транспонированной матрице: $M = M^{\mathrm{T}}$.

В матрице кососимметричного отношения выполняется условное высказывание:

$$(M(i,j) = И \text{ и } i \neq j) \Rightarrow M(j,i) = Л,$$

или, другими словами, пересечение матрицы смежности отношения с транспонированной матрицей принадлежит единичной матрице:

$$M \cap M^{T} \in Id_{M}.,$$

так как вне главной диагонали не должно быть ненулевых элементов.

Отличительное свойство матрицы транзитивного отношения довольно трудно сформулировать четко и наглядно. Тем не менее, вычисления на матрицах широко используется для определения свойства транзитивности отношения. Позже мы узнаем, как применяется композиция отношений для определения транзитивности.

Пример 4.5. Что можно сказать о свойствах рефлексивности, симметричности, кососимметричности и транзитивности следующих отношений:

(а) «*x* делит *y*» на множестве натуральных чисел;

(б) «$x \neq y$» на множестве целых чисел;

Решение.

(а) Поскольку *x* всегда делит сам себя, то это отношение рефлексивно. Оно не симметрично, поскольку, например, 2 является делителем 6, но не наоборот: 6 не делит 2. Проверим, что отношение делимости транзитивно. Предположим, что *x* делит *y,* а *y* в свою очередь делит *z*. Тогда из первого предположения вытекает, что $y = mx$, а из второго – $z = ny$ для некоторых натуральных чисел *m* и *n*. Следовательно, $z = ny = (nm)x$, т.е. *x* делит *z*. Значит, данное отношение транзитивно. Наконец, наше отношение кососимметрично, поскольку из предположений: *x* делит *y* и *y* делит *x* следует, что $y = x$.

(б) Так как высказывание «$x \neq x$» ложно, то это отношение не рефлексивно. Оно симметрично, поскольку $x \neq y$ тогда и только тогда, когда $y \neq x$. Наше отношение не обладает свойством транзитивности, так как, например, $2 \neq 3$ и $3 \neq 2$, но, тем не менее, $2 = 2$. Наше отношение не кососимметрично, поскольку из условий $x \neq$ у и у$\neq x$ не следует, что $x = y$.

Если отношение R на множестве A не обладает тем или иным свойством, то его можно продолжить до отношения R^*, которое будет иметь нужное свойство. Под «продолжением» мы понимаем присоединение некоторых упорядоченных пар к подмножеству $R \subset A \times A$ так, что новое полученное множество R^* уже будет обладать требуемым свойством. Ясно, что исходное множество R будет подмножеством в R^*. В том случае, если вновь построенное множество R^* будет минимальным среди всех расширений R с выделенным свойством, то говорят, что R^* является замыканием R относительно данного свойства.

Более строго, R_P^* называется *замыканием отношения* R относительно свойства P, если:

- R_P^* обладает свойством P;
- $R \subset R_P^*$;
- R_P^* является подмножеством любого другого отношения, содержащего R и обладающего свойством P.

Пример 4.6. Пусть $A = \{1,2,3\}$, а отношение R на A задано упорядоченными парами: $R = \{(1, 1), (1, 2), (1, 3), (3, 1), (2, 3)\}$. Оно не рефлексивно, не симметрично и не транзитивно. Найдите соответствующие замыкания.

Решение. Замыкание по рефлексивности должно содержать все пары вида (x, x). Поэтому искомое замыкание имеет вид:

$$R_R^* = \{(1, 1), (1, 2), (1, 3), (3, 1), (2, 3), \boldsymbol{(2, 2), (3, 3)}\},$$

где добавленные пары выделены курсивом.

Такой же результат можно получить, объединив матрицу смежности с единичной матрицей:

$$R \cup Id_R = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}.$$

Замыкание по симметричности должно содержать все пары,

симметричные исходным. Значит,

$$R_S^* = \{(1, 1), (1, 2), (1, 3), (3, 1), (2, 3), \textbf{\textit{(2, 1), (3, 2)}}\}.$$

Матрица замыкания по симметричности рассчитывается так:

$$R \cup R^T = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}.$$

Чтобы найти замыкание относительно транзитивности, необходимо выполнить несколько шагов. Так как *R* содержит пары (3, 1) и (1, 2), замыкание обязано включать в себя и пару (3, 2). Аналогично, пары (2, 3) и (3, 1) добавляют пару (2, 1), а пары (3, 1) и (1, 3) – пару (3, 3). Добавим сначала эти пары:

$$R^* \supset \{(1, 1), (1, 2), (1, 3), (3, 1), (2, 3), \textbf{\textit{(3, 2), (2, 1), (3, 3)}}\}.$$

Теперь у нас возникло сочетание (2, 1) и (1, 2). Значит, замыкание R_T^* должно содержать пару (2, 2). Можно увидеть, что все необходимые пары мы добавили (хотя бы потому, что перебрали все пары из A^2). Следовательно,

$$R_T^* = \{(1, 1), (1, 2), (1, 3), (3, 1), (2, 3), \textbf{(3, 2), (2, 1), (3, 3), (2, 2)}\}.$$

Способ, которым мы нашли замыкание по транзитивности в примере 4.6, довольно сложен даже для небольших матриц. Позже, после изучения композиции отношений, мы рассмотрим метод расчета матрицы замыкания по транзитивности, а сейчас приведем метод ее расчета без объяснений.

Матрица смежности умножается на саму себя до тех пор, пока $R^n \neq R^{n+1}$. Достигнув равенства $R^n = R^{n+1}$, строят матрицу замыкания, объединяя *n* рассчитанных матриц:

$$R_T^* = R \cup R^2 \cup R^3 \cup \ldots \cup R^n.$$

Применим этот алгоритм к нашему примеру.

$$R^2 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix},$$

$$R^3 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

Очевидно, что все последующие степени матрицы R будут иметь такой же вид, т.е., $R^3=R^4$. Значит,

$$[R_T^*] = [R \cup R^2 \cup R^3] = \begin{bmatrix}1 & 1 & 1\\ 0 & 0 & 1\\ 1 & 0 & 0\end{bmatrix} + \begin{bmatrix}1 & 1 & 1\\ 1 & 0 & 0\\ 1 & 1 & 1\end{bmatrix} + \begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} = \begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}.$$

Итак, замыкание по транзитивности отношения R состоит из всевозможных упорядоченных пар, построенных на множестве $A=\{1,2,3\}$. Такой же результат получен и в аналитическом решении.

Замыкание по транзитивности имеет множество приложений. Допустим, нам дан ориентированный граф, отражающий коммуникационную сеть. В этом случае матрица замыкания по транзитивности позволит нам определить, существует ли возможность переправить сообщение из одного места в другое.

4.3. Отношения эквивалентности и частичного порядка

Мы остановимся на двух специальных типах бинарных отношений.

Эквивалентность

Рефлексивное, симметричное и транзитивное бинарное отношение на множестве A называется *отношением эквивалентности.* Отношение эквивалентности в некотором смысле обобщает понятие равенства. *Эквивалентные элементы* (т.е. находящиеся в отношении эквивалентности), как правило, обладают какими-то общими признаками.

Приведем примеры отношения эквивалентности.

- Отношение «... имеет те же углы, что и ...» на множестве всех треугольников. Очевидно, треугольники эквивалентны относительно этого отношения тогда и только тогда, когда они подобны.
- Отношение R, заданное условием: xRy, если $xy > 0$ на множестве ненулевых целых чисел. При этом эквивалентные числа имеют одинаковый знак.
- Отношение «... имеет тот же возраст, что и ...» на множестве всех

людей. «Эквивалентные» люди принадлежат к одной и той же возрастной группе.

Эти примеры наводят на мысль, что если на множестве задано отношение эквивалентности, то все его элементы можно естественным способом разбить на непересекающиеся подмножества. Все элементы в любом из таких подмножеств эквивалентны друг другу в самом прямом смысле. Наличие такого разбиения – основа любой классификационной системы.

Разбиением множества A называется совокупность непустых подмножеств A_1, A_2, ..., A_n множества A, удовлетворяющих следующим требованиям:

1) $A = A_1 \cup A_2 \cup \ldots \cup A_n$;

2) $A_i \cap A_j = \varnothing$ при $i \neq j$.

Подмножества A_i называются *блоками* разбиения.

Диаграмма Венна разбиения множества A на пять блоков показана на рис. 4.3. Блоки изображены как участки, не заходящие один на другой. Это связано с тем, что блоки разбиения не могут иметь общих элементов.

Введем понятие *класс эквивалентности* E_x произвольного элемента $x \in A$ как подмножество $E_x = \{z \in A: zRx\}$.

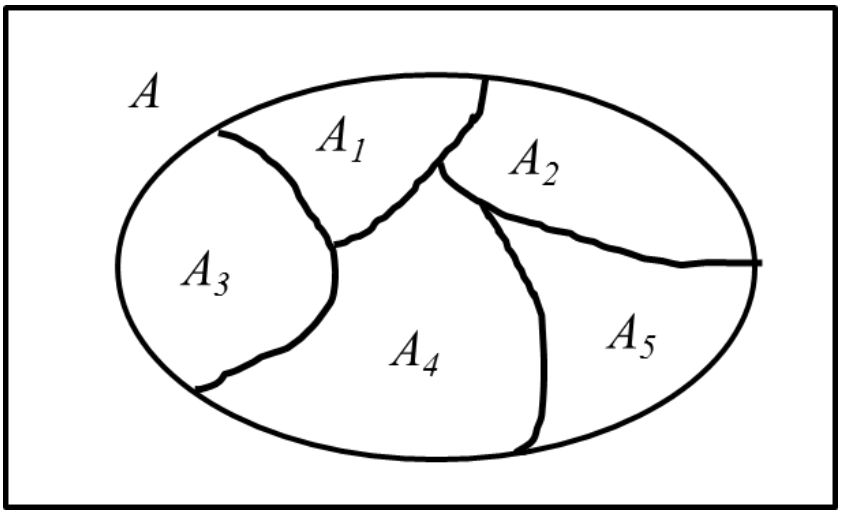


Рисунок 4.3. Диаграмма Венна разбиения множества A

Пример 4.7. Отношение S на множестве вещественных чисел R задано

условием: «xSy, если $(x - y)$ – целое число». Докажите, что R – отношение эквивалентности и опишите классы эквивалентности E_0, $E_{1/2}$ и $E_{\sqrt{2}}$.

Решение. Так как $x–x=0\in Z$ для любого вещественного числа x, отношение S рефлексивно. Если $x–y$ число целое, то и противоположное к нему $y–x=–(x–y)$ является целым. Следовательно, S – симметричное отношение. Пусть $x–y$ и $y–z$ – целые числа. Тогда $x–z=(x – y) + (y – z)$ – сумма целых чисел, т.е. целое число. Это означает, что S транзитивно.

Итак, мы показали, что S рефлексивно, симметрично и транзитивно. Следовательно, S – отношение эквивалентности.

Класс эквивалентности E_x произвольного вещественного числа x определяется по формуле $E_x = \{z\in R: z–x$ – целое число$\}$. Поэтому $E_0 = Z$;

$E_{1/2} = \{z\in R: z–1/2$ – целое число$\} = \{\ldots, -1\frac{1}{2}, -\frac{1}{2}, \frac{1}{2}, 1\frac{1}{2}, \ldots\}$;

$E_{\sqrt{2}}\{z\in R: z–\sqrt{2}$–целоечисло$\}=\{\ldots,–1+\sqrt{2}, \sqrt{2}, 1+\sqrt{2}, 2+\sqrt{2},\ldots\}$.

Итак, отношение эквивалентности S на множестве A задает на нем разбиение. Блоки разбиения при этом состоят из эквивалентных друг другу элементов. Мы сейчас докажем это утверждение.

Теорема. Пусть R – отношение эквивалентности на непустом множестве A. Тогда различные классы эквивалентности определяют разбиение A.

Доказательство. Доказательство состоит из четырех частей.

Сначала покажем, что классы эквивалентности являются непустыми подмножествами в A. По определению, E_x – подмножество в A. Кроме того, R– рефлексивное отношение, т.е. xRx. Следовательно, $x\in E_x$ и E_x не пусто.

Далее покажем, что из xRy вытекает равенство $E_x=E_y$. Предположим, что xRy, и возьмем произвольный $z\in E_x$. Тогда zRx и xRy. Поскольку R – транзитивное отношение, мы получаем, что zRy. Иными словами, $z\in E_y$. Следовательно, $E_x\subset E_y$. Аналогично можно показать, что $E_y\subset E_x$, откуда $E_x=E_y$, что и требовалось доказать.

Теперь мы покажем, что классы эквивалентности удовлетворяют первому свойству разбиения, а именно что A является объединением всех классов эквивалентности. Как уже отмечалось в первой части нашего доказательства, E_x – подмножество в A и поэтому объединение всех классов эквивалентности тоже будет подмножеством в A. С другой стороны, если $x \in A$, то $x \in E_x$. В частности, x принадлежит объединению классов эквивалентности. Значит, и A является подмножеством нашего объединения. Следовательно, A совпадает с объединением классов эквивалентности.

И, наконец, покажем, что два разных класса эквивалентности не пересекаются, т.е. что классы удовлетворяют второму свойству разбиения. Воспользуемся методом «от противного». Допустим, что $E_x \cap E_y \neq \varnothing$. Тогда найдется элемент z в A, принадлежащий пересечению $E_x \cap E_y$. Следовательно, zRx и zRy. Так как R – симметричное отношение, можно утверждать, что xRz и zRy. Ввиду транзитивности A это влечет xRy. Значит, по второй части доказательства $E_x = E_y$. Итак, мы предположили, что *разные* классы эквивалентности E_x и E_y пересекаются, и доказали, что они на самом деле совпадают. Полученное противоречие доказывает последнюю часть наших рассуждений. Теорема доказана. Заметим, чтобы показать, что классы эквивалентности служат блоками разбиения множества A, мы использовали все определяющие свойства отношения эквивалентности: рефлексивность, симметричность и транзитивность.

Частичный порядок

Рефлексивное, транзитивное, но кососимметричное отношение R на множестве A называется *частичным порядком.* Частичный порядок важен в тех ситуациях, когда мы хотим как-то охарактеризовать старшинство. Иными словами, решить, при каких условиях считать, что один элемент множества превосходит другой.

Примеры частичных порядков:

– « ≤ » на множестве вещественных чисел;

– « ⊂ » на подмножествах универсального множества;

– «... делит ...» на множестве натуральных чисел.

Множества с частичным порядком принято называть *частично упорядоченными множествами.*

Если R – отношение частичного порядка на множестве A, то при $x \neq y$ и xRy x называется *предшествующим элементом*, или *предшественником,* а y – *последующим.* У произвольно взятого элемента y может быть много предшествующих элементов. Однако если x предшествует y и не существует таких элементов z, для которых xRz и zRy, x называется *непосредственным предшественником* y и записывается как $x \prec y$.

Непосредственных предшественников можно условно изобразить с помощью графа, известного как *диаграмма Хассе.* Вершины графа изображают элементы частично упорядоченного множества A, и если $x \prec y$, то вершина x помещается ниже вершины y и соединяется с ней ребром. Диаграмма Хассе выдаст полную информацию об исходном частичном порядке, если подняться по всем цепочкам ребер. Элементы множества, входящие в такие цепочки, составляют *линейно упорядоченные подмножества* (порядок устанавливается по условию отношения).

Пример 4.8. Дано, что отношение «...делитель...» определяет частичный порядок на множестве $A=\{1, 2, 3, 6, 12, 18\}$. Составьте таблицу предшественников и непосредственных предшественников, после чего постройте соответствующую диаграмму Хассе.

Решение. Предшественники и непосредственные предшественники каждого элемента множества показаны в таблице 4.1, а диаграмма Хоссе – на рис. 4.4.

Таблица 4.1. Данные для диаграммы Хассе из примера 4.8

Элемент	Предшественник	Непосредственный предшественник

1	нет	нет
2	1	1
3	1	1
6	1, 2, 3	2, 3
12	1, 2, 3, 6	6
18	1, 2, 3, 6	6

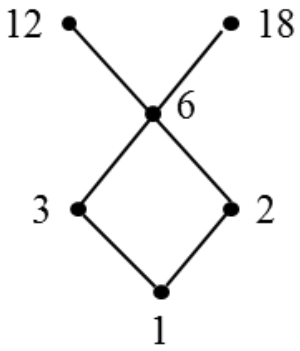


Рисунок 4.4. Диаграмма Хассе

Линейным порядком на множестве *A* называется отношение частичного порядка, при котором из любой пары элементов можно выделить предшествующий и последующий (т.е, любые два элемента сравнимы).

Примеры линейного порядка:

« ≤ » на множестве вещественных чисел;

лексикографическое упорядочение слов в словаре.

Различные сортирующие процедуры в информатике требуют, чтобы элементы сортируемых множеств были линейно упорядочены. В этом случае они могут выдавать упорядоченный список. Другие приложения используют частичный порядок, предполагая, что в любом конечном частично упорядоченном множестве найдется *минимальный элемент* (не имеющий предшественников) и *максимальный* (не имеющий последующих элементов). В случае бесконечных множеств это не так. Например, в множестве Z относительно порядка «≤» нет ни минимального, ни максимального элемента.

Частично упорядоченное множество из примера 4.8 обладает одним минимальным элементом – числом 1. В нем есть два максимальных элемента

– 12 и 18. В этом множестве содержится несколько линейно упорядоченных относительно отношения «...делитель...» подмножеств. Каждое из них соответствует цепочке ребер на диаграмме Хассе. Это подмножества {1,2,6,12}, {1,2,6,18}, {1,3,6,12} и {1,3,6,18}. Все подмножества, включающие элементы {2, 3} или {12, 18} линейно не упорядочены, т.к. для этих пар элементов невозможно установить старшинство в соответствии с условием отношения.

4.4. Обратные отношения и композиция отношений

Рассмотрим два способа построения новых бинарных отношений, которые основаны на вычислении обратного отношения и определении композиции отношений.

Пусть R – бинарное отношение между множествами A и B. Определим *обратное отношение* R^{-1} между B и A по формуле $R^{-1} = \{(b,a): (a,b) \in R\}$.

Например, обратным к отношению «... родитель ...» на множестве всех людей будет отношение «...ребенок...».

На графическом языке обратное отношение получается обращением всех стрелок в орграфе, изображающем исходное отношение.

Дадим определение композиции отношений. Пусть R – бинарное отношение между множествами A и B, а S – бинарное отношение между B и третьим множеством C. *Композицией отношений* R и S называется бинарное отношение между A и C, которое обозначается $S \circ R$ и определяется формулой: $S \circ R = \{(a,c)$: $a \in A$, $c \in C$ и aRb, bSc для некоторого $b \in B\}$.

Новое отношение устанавливает связь между элементами множеств A и C, используя элементы из B в качестве посредников.

Пример 4.11. Пусть R – отношение «a – сестра b», а S обозначает отношение «b – мать c» на множестве всех людей. Опишите на словах композиции $S \circ R$ и $S \circ S$ и $R \circ S$.

Решение. Если a – сестра b, а b – мать c, то a, очевидно, будет сестрой матери c, т. е. a приходится тетей c. Значит, отношение $S \circ R$ – это отношение «a – тетя c».

Аналогичными рассуждениями легко установить, что $S \circ S$ – это «a – бабушка c», а $R \circ S$ – это «a – мать с».

Пример 4.12. Предположим, что отношения R и S заданы орграфами, представленными на рис. 4.5. Найдите орграф, соответствующий композиции отношений $S \circ R$.

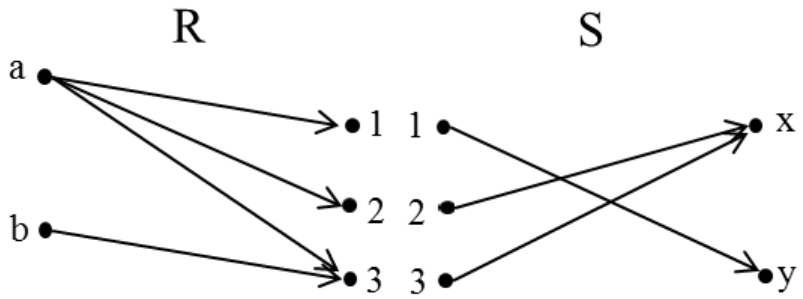


Рисунок 4.5. Орграфы отношений R и S

Решение. Используя орграфы, выпишем упорядоченные пары, принадлежащие отношениям R и S:

$R = \{(a, 1), (a, 2), (a, 3), (b, 2)\}$; $S = \{(1, y), (2, x), (3, x)\}$.

Применим определение композиции отношений:

$$aR1 \text{ и } 1Sy \Rightarrow (a,y) \in S \circ R;$$

$$aR2 \text{ и } 2Sx \Rightarrow (a,x) \in S \circ R;$$

$$aR3 \text{ и } 3Sx \Rightarrow (a,x) \in S \circ R;$$

$$bR2 \text{ и } 2Sx \Rightarrow (b,x) \in S \circ R.$$

Теперь на рис. 4.6 изобразим орграф композиции.

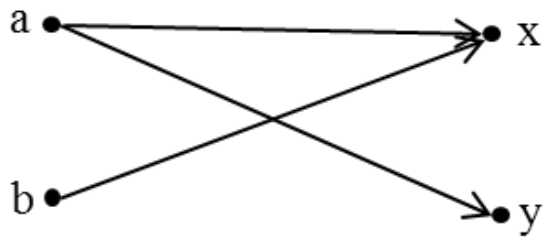


Рисунок 4.6. Орграф отношения S°R

Такой же граф можно построить, используя графы из рис.4.5, если соединить вершины *a* и *y* через общую вершину *1*, *a* и *x* через общую вершину *2* и *b* и *x* через общую вершину *3*.

Композицию бинарных отношений можно вычислить и с помощью их матриц смежности. Покажем, что матрица смежности композиции двух отношений равна логическому или булевому произведению их матриц смежности.

Рассмотрим три множества:

$A = \{a_1, a_2, ..., a_n\}$, $B = \{b_1, b_2, ..., b_m\}$ и $C = \{c_1, c_2, ..., c_p\}$.

Предположим, что R – отношение между A и B, а S – отношение между B и C. Напомним, что матрица M отношения R определяется условием:

$M(i, j)$ = И, если $(a_i, b_j) \in R$,

$M(i, j)$ = Л, если $(a_i, b_j) \notin R$.

Аналогично, матрица N отношения S заполняется по правилу:

$N(i, j)$ = И, если $(b_i, c_j) \in S$,

$N(i, j)$ = Л, если $(b_i, c_j) \notin S$.

Если найдется такой элемент $b_k \in B$, что $a_i R b_k$ и $b_k S c_j$, то в i-ой строке матрицы M на k-ом месте стоит *И*. Кроме того, в j-ом столбце матрицы N на k-ом месте тоже будет стоять значение *И*. С другой стороны, поскольку по определению композиции отношений $a_i(S \circ R)c_j$, то значение $P(i, j)$ логической матрицы P отношения $S \circ R$ тоже равно *И*. Если же в i-ой строке матрицы M нет значений *И*, соответствующих такому же значению в j-ом столбце матрицы N,

то $P(i, j)$ = Л.

Таким образом, логическая матрица P композиции $S°R$ заполняется по следующему правилу:

$P(i, j)$ = [$M(i, 1)$ **и** $N(1, j)$)]

или [$M(i, 2)$ **и** $N(2, j)$]

……

или [$M(i, n)$ **и** $N(n, j)$],

т.е. равна булевому произведению матриц M и N, которое будем записывать в виде $P = M \times N$.

Пример 4.13. Пусть R и S – отношения из примера 4.12. Вычислите логическую матрицу отношения $S°R$ с помощью булева произведения логических матриц отношений R и S.

Решение. Отношение R между $A = \{a, b\}$ и $B = \{1, 2, 3\}$ задается матрицей

$$M = \begin{bmatrix} И & И & И \\ Л & И & Л \end{bmatrix},$$

строки и столбцы которой помечены элементами множеств A и B в том порядке, в котором они выписаны.

Аналогично, S – это отношение между множествами $B = \{1, 2, 3\}$ и $C = \{x, y\}$, заданное матрицей

$$N = \begin{bmatrix} Л & И \\ И & Л \\ И & Л \end{bmatrix}.$$

Значит, логическая матрица P отношения $S°R$ равна булеву произведению матриц M и N:

$$P = \begin{bmatrix} И & И & И \\ Л & И & Л \end{bmatrix} \times \begin{bmatrix} Л & И \\ И & Л \\ И & Л \end{bmatrix}.$$

В матрице M есть две строки, а в матрице N – два столбца. По правилу произведения матрица P состоит из двух строк и двух столбцов. Ячейка $P(1, 1)$ заполняется по первой строке матрицы M и первому столбцу матрицы N:

$$P(1,1) = [\text{И} \quad \text{И} \quad \text{И}] \times \begin{bmatrix} \text{Л} \\ \text{И} \\ \text{И} \end{bmatrix} = (И \textbf{ и } Л) \textbf{ или } (И \textbf{ и } И) \textbf{ или } (И \textbf{ и } И) =$$

$$= Л \textbf{ или } И \textbf{ или } И = И.$$

Заметим, что в первой строке матрицы *M* на втором и третьем местах стоит И, так же как и в первом столбце матрицы *N*. Этого достаточно для обоснованного заключения: *P*(1,1) = И.

Сравнивая первую строку матрицы *M* со вторым столбцом матрицы *N*, мы видим, что в обоих случаях на первом месте стоит значение И. Следовательно, *P*(1,2) = И.

Таким же способом, глядя на вторую строку матрицы *M* и первый столбец матрицы *N*, определяем *P*(2,1) = И.

Наконец, *P*(2,2) = Л, т.к. вторая строка матрицы *M* и второй столбец матрицы *N* не имеют значения И на одинаковых местах. Итак, $P = \begin{bmatrix} \text{И} & \text{И} \\ \text{И} & \text{Л} \end{bmatrix}$, что соответствует орграфу, изображенному на рис. 4.6.

Пример 4.14. Отношение *R* на множестве *A* = {1, 2, 3, 4, 5} задается матрицей

$$\text{M} = \begin{bmatrix} \text{Л} & \text{Л} & \text{И} & \text{И} & \text{Л} \\ \text{Л} & \text{И} & \text{Л} & \text{Л} & \text{И} \\ \text{Л} & \text{Л} & \text{Л} & \text{И} & \text{Л} \\ \text{Л} & \text{Л} & \text{И} & \text{Л} & \text{Л} \\ \text{Л} & \text{И} & \text{Л} & \text{Л} & \text{Л} \end{bmatrix}.$$

Вычислить матрицу композиции *R*°*R* и объяснить, почему отношение *R* не обладает свойством транзитивности.

Решение. Матрица композиции *R*°*R* равна

$$\text{M}^2 = \begin{bmatrix} \text{Л} & \text{Л} & \text{И} & \text{И} & \text{Л} \\ \text{Л} & \text{И} & \text{Л} & \text{Л} & \text{И} \\ \text{Л} & \text{Л} & \text{Л} & \text{И} & \text{Л} \\ \text{Л} & \text{Л} & \text{И} & \text{Л} & \text{Л} \\ \text{Л} & \text{И} & \text{Л} & \text{Л} & \text{Л} \end{bmatrix} \times \begin{bmatrix} \text{Л} & \text{Л} & \text{И} & \text{И} & \text{Л} \\ \text{Л} & \text{И} & \text{Л} & \text{Л} & \text{И} \\ \text{Л} & \text{Л} & \text{Л} & \text{И} & \text{Л} \\ \text{Л} & \text{Л} & \text{И} & \text{Л} & \text{Л} \\ \text{Л} & \text{И} & \text{Л} & \text{Л} & \text{Л} \end{bmatrix} = \begin{bmatrix} \text{Л} & \text{Л} & \text{И} & \text{И} & \text{Л} \\ \text{Л} & \text{И} & \text{Л} & \text{Л} & \text{И} \\ \text{Л} & \text{Л} & \text{И} & \text{Л} & \text{Л} \\ \text{Л} & \text{Л} & \text{Л} & \text{И} & \text{Л} \\ \text{Л} & \text{И} & \text{Л} & \text{Л} & \text{И} \end{bmatrix}$$

Элементы композиции *R*°*R* имеют вид *(x, z)*, где *xRy* и *yRz* для какого-

нибудь $y \in A$. Поэтому в случае транзитивности R композиция $R \circ R$ должна быть подмножеством R. Однако из расположения значения И в матрицах, выписанных выше, видно, что $R \circ R$ содержит пары, которые не лежат в R. Именно поэтому отношение R не транзитивно.

Чтобы построить замыкание отношения по транзитивности, нужно умножать матрицу смежности саму на себя до тех пор, пока $M^n \neq M^{n+1}$. Достигнув равенства матриц $M^n \neq M^{n+1}$, можно построить матрицу смежности замыкания отношения по транзитивности:

$$M_T^* = M \cup M^2 \cup \ldots \cup M^n.$$

Рассмотренные в этом параграфе две операции (обратные отношения и композиции отношений) особенно важны при изучении функций.

4.5. Реляционная алгебра

Реляционная модель данных и построенные на ее основе программные средства обработки массивов данных является одной из самых успешных современных информационных технологий. В основе ее лежит простая идея – все данные упорядочены и хранятся в двумерных таблицах. Хранящийся в таблице набор данных (строк) можно рассматривать как математическое отношение, арность которого равна количеству столбцов таблицы. При этом наборы данных в строках таблицы соответствуют кортежам множества, на котором построено такое отношение.

Действительно, можно считать, что в таблицу, изображенную на рис.4.7, внесены данные, содержащиеся в четырех множествах – П№ = {П1, П2, П3, П4, П5}, Имя_П = {Бык, Волк, Заяц, Лев, Лиса}, Статус = {10, 20, 30} и Город = {Брест, Гродно, Минск}.

Поставщики

П№	**Имя_П**	**Статус**	**Город**
П1	Волк	20	Брест
П2	Заяц	10	Минск

П3	Лев	30	Гродно
П4	Лиса	20	Минск
П5	Бык	30	Брест

Рис. 4.7. Таблица с данными о поставщиках

В математике *декартовым произведением* (или сокращенно *произведением)* N множеств является множество всех таких упорядоченных наборов из N элементов, которые называются N-арными кортежами, что в каждом из них первый элемент берется из первого множества, второй – из второго и т.д. Подмножество такого множества, кортежи которого удовлетворяют заданному условию, называется N-арным отношением. Следовательно, результатом произведения четырех множеств П№×Имя_П×Статус×Город будет множество, состоящее из 4-арных кортежей, представляющих собой всевозможные упорядоченные наборы элементов наших четырех множеств. Выбрав из этого множества кортежи, которые соответствуют наборам данных в строках таблицы, мы получим подмножество, которое представляет собой математическое отношение. Если по каким-то причинам отсутствуют данные в некоторых ячейках таблицы, то в соответствующие множества (и в пустые ячейки таблицы) можно добавить элемент, который рассматривается как метка, указывающая на отсутствие данных. В теории баз данных его принято называть Null-значением. Таким образом, наборы данных в строках двумерной таблицы могут рассматриваться как математическое отношение, к которому могут быть применены соответствующие правила. Поэтому в учебном пособии для наглядности мы будем часто использовать термин «таблица» вместо более корректного термина «математическое отношение».

Идея реляционной модели была предложена американским математиком Э. Коддом и подробно представлена в статье E.F. Codd. Relational Completeness of Data Base Sublanguages // Randall J. Rustin (ed.). Data Base Systems, Courant Computer Science Symposia Series 6. – Englewood Cliffs, N.J.: Prentice-Hall,

1972.

Часть реляционной модели, которая связана с применением операторов для обработки отношений, основана на реляционной алгебре. Операторы в реляционной алгебре используют отношения в качестве операндов и возвращают отношения в качестве результата. Это реляционное свойство называется свойством *замкнутости*. Благодаря замкнутости, результат одной операции может использоваться в качестве исходных данных для другой. Это, в свою очередь, дает возможность формировать *вложенные выражения*, т.е., выражения, в которых операнды сами представлены выражениями (по аналогии, в арифметике, благодаря свойству замкнутости, результатом арифметических операций с числами получаем также число, к которому могут быть применены арифметические операторы, то есть, могут быть сформированы вложенные выражения).

Это же свойство замкнутости обусловливает необходимость принятия *правил наследования имен атрибутов* для того, чтобы можно было предсказывать имена атрибутов на выходе произвольной реляционной операции. Задав такие правила для всех операций, можно гарантировать, что для выражения любой сложности будет вычисляться отношение, имеющее вполне определенный набор атрибутов.

По определению Е. Кодда, основу реляционной алгебры составляют восемь операторов, которые разделены на две группы – *традиционные и специальные реляционные операторы*. Количество операторов могло быть иным. Например, не все предложенные операторы являются примитивными. Оператор соединения можно заменить двумя другими – умножением и выборкой. С другой стороны, можно добавить новые операторы, если они не будут нарушать требования реляционной алгебры.

Некоторые из операторов реляционной алгебры могут выполняться только при определенных условиях, в отличие от подобных операторов, действующих на множествах. Например, в математике объединение двух

множеств – это множество всех элементов, принадлежащих или обоим, или одному из исходных множеств. Объединение множества кортежей в приведенном выше отношении *Поставщики* и множества, например, кортежей в отношении, содержащим данные о свойствах деталей, является множеством, но не является отношением – отношения не могут содержать смесь кортежей разных типов. В реляционной алгебре для объединения требуется, чтобы два исходных отношения имели один и тот же тип или, другими словами, были совместимы по типу. *Отношения совместимы по типу*, если каждое из них имеет одно и то же множество имен атрибутов и соответствующие атрибуты определены на одном и том же домене (имели одинаковый смысл).

Традиционные реляционные операции

К традиционным операциям над множествами относятся операторы объединения, пересечения, вычитания. и произведения. Эти операторы подобны соответствующим операторам алгебры множеств, но имеют некоторые ограничения, которые отражены в определениях.

Объединением двух совместимых по типу отношений A и B (A UNION B) называется отношение с тем же заголовком, как и в отношениях A и B, и с телом, состоящим из множества всех кортежей t, принадлежащих A или B или обоим отношениям. При этом совпадающие кортежи записываются один раз. Пример операции объединения показан на рис. 4.8.

П1

П№	**Имя_П**	**Город**
П1	Волк	Брест
П2	Заяц	Минск
П3	Лев	Гродно

П2

П№	**Имя_П**	**Город**
П2	Заяц	Минск
П4	Лиса	Минск
П5	Бык	Брест

П1 UNION П2

П№	**Имя_П**	**Город**
П1	Волк	Брест
П2	Заяц	Минск
П3	Лев	Гродно
П4	Лиса	Минск
П5	Бык	Брест

Рис. 4.8. Пример объединения двух таблиц

Пересечением двух совместимых по типу отношений A и B

(A INTERSECT B) называется отношение с тем же заголовком, как и в отношениях A и B, и с телом, состоящим из множества всех кортежей t, которые принадлежат одновременно обоим отношениям A и B. Пример операции пересечения показан на рис. 4.9.

П1

П№	**Имя_П**	**Город**
П1	Волк	Брест
П2	Заяц	Минск
П3	Лев	Гродно
П4	Лиса	Минск
П5	Бык	Брест

П2

П№	**Имя_П**	**Город**
П1	Волк	Брест
П3	Лев	Гродно
П4	Лиса	Минск
П7	Зубр	Брест

П1 INTERSECT П2

П№	**Имя_П**	**Город**
П1	Волк	Брест
П3	Лев	Гродно
П4	Лиса	Минск

Рис. 4.9. Пример пересечения двух таблиц

Вычитанием двух совместимых по типу отношений A и B (A MINUS B) называется отношение с тем же заголовком, как и в отношениях A и B, и с телом, состоящим из множества всех кортежей t, принадлежащих отношению A и не принадлежащих отношению B. Пример операции вычитания показан на рис. 4.10.

П1

П№	**Имя_П**	**Город**
П1	Волк	Брест
П2	Заяц	Минск
П3	Лев	Гродно
П4	Лиса	Минск
П5	Бык	Брест

П2

П№	**Имя_П**	**Город**
П1	Волк	Брест
П3	Лев	Гродно
П4	Лиса	Минск
П7	Зубр	Брест

П1 MINUS П2

П№	**Имя_П**	**Город**
П2	Заяц	Минск
П5	Бык	Брест

Рис. 4.10. Пример вычитания двух таблиц

Декартово произведение двух отношений A и B (A TIMES B), где A и B не имеют общих имен атрибутов, определяется как отношение с заголовком, который представляет собой сцепление (конкатенацию) двух заголовков исходных отношений A и B, и телом, состоящим из множества всевозможных кортежей t таких, что t представляет собой сцепление кортежа a, принадлежащего отношению A, и кортежа b, принадлежащего отношению B. Кардинальное число нового отношения равняется произведению кардинальных чисел исходных отношений, а степень равняется сумме их степеней (см. рис. 4.11.).

A
a_1
a_2
a_3

TIMES

B
b_1
b_2

=

A	B
a_1	b_1
a_1	b_2
a_2	b_1
a_2	b_2
a_3	b_1
a_3	b_2

Рис. 4.11. Пример реляционной операции декартова умножения

Специальные реляционные операции

К специальным реляционным операциям относятся выборка, проекция, соединение и деление.

Выборка или *сокращение* (WHERE) – это упрощенное название θ-выборки, где θ обозначает любой скалярный оператор сравнения (=, ≠, ≥, > и т.д.). θ-выборкой из отношения A по атрибутам X и Y (A WHERE XθY) (порядок учитывается!) называется отношение, имеющее тот же заголовок, что и отношение A, и тело, содержащее множество всех кортежей t отношения A, для которых проверка условия «XθY» дает значение истина. Атрибуты X и Y должны быть определены на одном и том же домене, а оператор сравнения θ должен иметь смысл для данного домена. Пример выборки из таблицы *Поставщики* (см. рис. 4.7) по условию *город='Минск' OR город='Гродно'* показан на рис. 4.12:

Поставщики

П№	Имя_П	Статус	Город
П2	Заяц	10	Минск
П3	Лев	30	Гродно
П4	Лиса	20	Минск

Рис. 4.12. Пример действия реляционной операции выборки:
Поставщики WHERE город=’Минск’ OR город=‘Гродно’

Проекцией (PROJECT) отношения A по атрибутам X, Y, …, Z, где каждый из атрибутов принадлежит отношению A, называется отношение с заголовком {X,Y,…,Z} и телом, содержащим множество кортежей с атрибутами, совпадающими с соответствующими атрибутами отношения A. Т.е., с помощью операции проекции получается вертикальное подмножество исходного отношения – подмножество, получаемое исключением всех атрибутов, не указанных в списке атрибутов, и последующим исключением дублирующих кортежей из того, что осталось. Пример проекции из таблицы Поставщики по атрибутам Имя_П и Город показан на рис. 4.13 (обозначение проекции: Поставщики [Имя_П, Город]).

Поставщики [Имя_П, Город]

Имя_П	**Город**
Волк	Брест
Заяц	Минск
Лев	Гродно
Лиса	Минск
Бык	Брест

Рис. 4.13. Пример проекции из таблицы Поставщики по атрибутам
Имя_П, Город

Соединение (JOIN) – это разновидность операции произведения, в которой сцепление кортежей основывается на задаваемом атрибуте или наборе атрибутов каждого из двух отношений. Значения указанных атрибутов сравниваются с целью наложения определенных ограничений на результат. Отношения можно соединять по атрибутам, имеющим либо общие домены, либо сопоставимые домены, когда значения данных из одного домена можно сравнить со значениями данных из другого домена.

Наиболее часто используется естественное или внутреннее соединение, когда отношения имеют общий атрибут и результат содержит только строки, в которых значения общего атрибута совпадают.

Естественным (внутренним) соединением отношений A и B (A JOIN B) с заголовками X,Y и Y,Z соответственно и с атрибутами Y, определенными на одном и том же домене, называется отношение с заголовком {X,Y,Z} и телом, содержащим множество кортежей с атрибутами, совпадающими с соответствующими атрибутами отношений A и B. Пример естественного соединения отношений Поставки и Поставщики показан на рис. 4.14.

Отношения имеют общий атрибут П№. В результирующем отношении присутствуют соединения кортежей, в которых значения общего атрибута совпадают. Таким образом, благодаря соединению в одном кортеже собраны данные о поставке и поставщике, осуществившим эту поставку. При этом строки, не имеющие одинаковых значений в общем атрибуте, в результирующее отношение не вошли (строки с П№=’П5’ из таблицы Поставки и строка с П№=’П7’ из таблицы Поставщики.

В случае внешнего соединения (Поставки OUTER JOIN Поставщики) кортеж, который невозможно соединить с кортежем соответствующей таблицы из-за отсутствия совпадающих значений, будет помещен в результирующую таблицу, а для присоединенных атрибутов значения определены не будут, т.е., им присвоят Null-значения (рис. 4.15).

Поставки

П№	**Д№**	**Пр№**	**Кол**
П1	Д1	Пр1	200
П1	Д1	Пр4	700
П2	Д3	Пр1	400
П2	Д3	Пр2	200
П2	Д3	ПР7	800
П2	Д5	ПР2	100
П3	Д3	ПР1	200
П3	Д4	ПР2	500
П4	Д6	ПР3	300
П4	Д6	ПР7	300
П5	Д2	ПР2	200
П5	Д6	ПР2	200

Поставщики

П№	**Имя_П**	**Статус**	**Гор**
П1	Волк	20	Брест
П2	Заяц	10	Минск
П3	Лев	30	Гродно
П4	Лиса	20	Минск
П7	Бык	30	Брест

Табл1 = Поставки JOIN Поставщики						
П№	**Д№**	**Пр№**	**Кол**	**Имя_П**	**Статус**	**Гор**
П1	Д1	Пр1	200	Волк	20	Брест
П1	Д1	Пр4	700	Волк	20	Брест
П2	Д3	Пр1	400	Заяц	10	Минск
П2	Д3	Пр2	200	Заяц	10	Минск
П2	Д3	ПР7	800	Заяц	10	Минск
П2	Д5	ПР2	100	Заяц	10	Минск
П3	Д3	ПР1	200	Лев	30	Гродно
П3	Д4	ПР2	500	Лев	30	Гродно
П4	Д6	ПР3	300	Лиса	20	Минск
П4	Д6	ПР7	300	Лиса	20	Минск

Рис. 4.14. Результат естественного (внутреннего) соединения таблиц

Большинство СУБД используют также разновидности внешнего соединения с операторами SQL – LEFT JOIN и RIGHT JOIN.

Соединение обладает свойствами ассоциативности и коммутативности. Если отношения A и B не имеют общих имен атрибутов, то естественное соединение превращается в декартово произведение.

Поставки

П№	Д№	Пр№	Кол
П1	Д1	Пр1	200
П1	Д1	Пр4	700
П2	Д3	Пр1	400
П2	Д3	Пр2	200
П2	Д3	ПР7	800
П2	Д5	ПР2	100
П3	Д3	ПР1	200
П3	Д4	ПР2	500
П4	Д6	ПР3	300
П4	Д6	ПР7	300
П5	Д2	ПР2	200
П5	Д6	ПР2	200

Поставщики

П№	Имя_П	Статус	Гор
П1	Волк	20	Брест
П2	Заяц	10	Минск
П3	Лев	30	Гродно
П4	Лиса	20	Минск
П7	Бык	30	Брест

Тавл1 = Поставки OUTER JOIN Поставщики

П№	Д№	Пр№	Кол	Имя_П	Статус	Гор
П1	Д1	Пр1	200	Волк	20	Брест
П1	Д1	Пр4	700	Волк	20	Брест
П2	Д3	Пр1	400	Заяц	10	Минск
П2	Д3	Пр2	200	Заяц	10	Минск
П2	Д3	ПР7	800	Заяц	10	Минск
П2	Д5	ПР2	100	Заяц	10	Минск
П3	Д3	ПР1	200	Лев	30	Гродно
П3	Д4	ПР2	500	Лев	30	Гродно
П4	Д6	ПР3	300	Лиса	20	Минск
П4	Д6	ПР7	300	Лиса	20	Минск
П5	Д2	ПР2	200			
П5	Д6	ПР2	200			
П7				Бык	30	Брест

Рис. 4.15. Результат внешнего соединения таблиц Поставки и Поставщики

Результатом *Деления* (DIVIDED BY) двух отношений, бинарного и унарного, является отношение, содержащее все значения одного атрибута бинарного отношения, которые сцеплены в другом атрибуте со всеми значениями в унарном отношении.

На рис. 4.16 показаны два примера деления отношений.

П№	Д№
П1	Д1
П1	Д2
П1	Д4
П2	Д1
П2	Д2

DIVIDED BY

Д№
Д2
Д4

=

П№
П1
П4

П№	Д№
П3	Д2
П4	Д1
П4	Д2
П4	Д4

DIVIDED BY

Д№
Д1

=

П№
П1
П2
П4

Рис. 4.16. Примеры деления бинарного отношения на унарное.

Приведенное выше определение *операции деления* можно развить для случая деления n-арного отношения на m-арное отношение (n>m), в результате которого получается (n – m)-арное отношение.

Дополнительные реляционные операции

Итак, Е. Кодд предложил восемь операторов для построения реляционных выражений. Традиционные операторы соответствуют операциям на множествах (с некоторыми ограничениями, которые учитывают специфику математических отношений), специальные операторы предложены для решения специфических задач, связанных с обработкой табличных данных. Однако могут возникать ситуации, когда этих операторов недостаточно для построения реляционного выражения, описывающего алгоритм обработки данных. В таких случаях предлагается создавать и использовать дополнительные операторы, действия которых не противоречат постулатам реляционной алгебры. Например, необходимо перемножить два отношения, имеющих одинаковые атрибуты, что запрещено правилами декартового произведения. Можно создать и применить оператор переименования атрибутов (Е. Кодд назвал его RENAME) и после этого произведение отношений будет возможно:

Поставки TIMES (Поставщики RENAME ПN№ AS П1№).

Описанные выше операторы реляционной алгебры не содержат средств для скалярных вычислений. Например, надо найти стоимость поставок деталей Д1, если одна деталь стоит 235 рублей. Для обеспечения таких возможностей предлагается *операция расширения* EXTEND. С помощью этой операции создается новое отношение, похожее на исходное, но содержащее дополнительный атрибут, значения которого получены посредством некоторых скалярных вычислений:

Extend A add expr as z.

Результатом этого выражения будет отношение с заголовком, эквивалентным заголовку отношения A, дополненному новым атрибутом z, который рассчитывается скалярным выражением expr для каждого кортежа отношения A. При этом отношение A не должно иметь атрибута z и выражение expr не должно ссылаться на атрибут z. Кардинальное число нового отношения равно кардинальному числу отношения A, степень равна степени отношения A плюс единица.

Сейчас, применив реляционное выражение

EXTEND (Поставки WHERE ДN№='Д1') ADD Кол*235 AS сумма,

получим таблицу поставок деталей Д1, содержащую в поле «сумма» стоимость каждой поставки (рис. 4.17):

П№	**Д№**	**Пр№**	**Кол**	**сумма**
П1	Д1	Пр1	200	47000
П1	Д1	Пр4	700	164500
П5	Д1	ПР4	100	23500

Рис. 4.17. Пример действия оператора EXTEND

Пример более сложного реляционного выражения:

Подсчитать количество поставок, сделанных каждым поставщиком.

EXTEND Поставщики ADD COUNT ((Поставки RENAME ПN№ AS X)
WHERE X= ПN№) AS Кол_П;

Результат действия этого выражения показан на рис. 4.18. Изменив в таблице Поставки название атрибута ПN№ на X, мы получили возможность для каждого поставщика из таблицы Поставщики отбирать соответствующие ему

строки в таблице Поставки ((Поставки RENAME П№ AS X) WHERE X=П№) и при помощи итоговой функции COUNT( ) подсчитывать их количество, которое равно числу поставок. Результат записывается в поле Кол_П, которое оператор EXTEND добавляет к таблице Поставщики.

П№	Имя_П	Статус	Гор	Кол_П
П1	Волк	20	Брест	6
П2	Заяц	10	Минск	2
П3	Лев	30	Гродно	1
П4	Лиса	20	Минск	3
П5	Бык	30	Брест	0

Рис. 4.18. Пример действия оператора EXTEND

Таким образом, операция расширения обеспечивает возможность горизонтальных или построчных, осуществляемых в пределах кортежа, вычислений.

Итоговые (или статистические) функции – это функции, которые в качестве аргумента используют множество значений и в результате возвращают одно значение. Кроме упомянутой ранее функции COUNT (подсчет строк в таблице), итоговыми функциями являются SUM (сумма), AVG (среднее значение), MAX (максимальное значение), MIN (минимальное значение), StDev (среднеквадратичное отклонение от среднего значения поля), Var (дисперсия значений поля) и некоторые другие. Если аргумент такой функции будет пустым множеством, то функции COUNT и SUM возвращают нуль, функции MAX и MIN возвращают максимальное и минимальное значения соответствующего домена.

Итоговые функции используются, как правило, для обработки группы строк или всех строк таблицы – для так называемых вертикальных вычислений. Для таких случаев предложен оператор агрегирования (или подведения итогов) SUMMARIZE:

SUMMARIZE A BY (a1,a2,...,an) ADD expr AS Z,

где a1, a2, …, an – отдельные атрибуты отношения A. Результатом этого выражения будет отношение с заголовком {a1, a2, …, an, z} и с телом,

содержащим все такие кортежи t, которые являются кортежами проекции отношения A по атрибутам a1, a2, …, an, расширенного значением для нового атрибута z. Такое новое значение z подсчитывается вычислением итогового значения expr по всем кортежам отношения A, которые имеют те же самые значения для атрибутов a1, a2, …, an, что и кортеж t.

Другими словами, оператор SUMMARIZE собирает в группы кортежи отношения A, имеющие одинаковые значения наборов атрибутов a1, a2, …, an и для них проводит вычисления по формуле expr. В результирующей таблице для каждого набора a1, a2, …, an приводится результат вычислений expr, записанный в поле z.

Кардинальное число результирующего отношения равно кардинальному числу проекции отношения A по атрибутам a1, a2, …, an, а степень равна степени такой проекции плюс единица. Например, рассчитанное в прошлом примере количество поставок, сделанных каждым поставщиком, может быть получено при помощи выражения

```
SUMMARIZE Поставки BY(П№) ADD COUNT AS Кол_П.
```

В отличие от результата действия оператора EXTEND, примененного в предыдущем решении, в этом случае результат вычислений не содержит сведений о поставках поставщика П5 (рис. 4.19). Причина состоит в том, что поставщика П5 нет в отношении Поставки.

П№	Кол_П
П1	6
П2	2
П3	1
П4	3

Рис. 4.19

Если группировка кортежей производится по пустому множеству атрибутов, то вычисление выполняется по всем кортежам отношения и итоговая функция выдает единственный результат для всей таблицы. Например, результатом действия реляционного выражения

```
SUMMARIZE Поставки BY( ) ADD COUNT AS Кол_П
```

будет таблица, содержащая одну ячейку с именем атрибута Кол_П.

Популярными дополнительными реляционными операциями являются также:

– операции реляционного присвоения, которые дают возможность «запоминать» результаты действия некоторых алгебраических выражений в базе данных и таким образом изменять состояние базы данных или, иначе говоря, обновлять базу данных, например:

  Поставки := Поставки MINUS (Поставки WHERE Кол = 0);

– операции вставки:

  INSERT (Поставщики WHERE Гор_П = Минск) INTO Temp;

(здесь выбранные данные вставлены в отношение Temp, которое должно быть совместимым по типу с отношением Поставщики);

– операции обновления данных:

  UPDATE (Поставщики WHERE Гор_П = Брест) СТАТУС := 40;

– операции удаления данных:

  DELETE Поставщики WHERE Статус < 20.

Подобные операции действуют на уровне множеств (например, операция DELETE удаляет множество кортежей из целевого отношения) и должны контролироваться с помощью предиката рассматриваемого отношения.

Дополнительные реляционные операции не ограничиваются приведенным здесь списком. Ранее говорилось о том, что при необходимости можно создавать новые реляционные операторы. Как правило, авторами предлагаются составные, построенные на восьми операторах, предложенных Е. Коддом. Например, оператор полусоединения A SEMIJOIN B, который после соединения таблиц A и B выполняет проекцию по атрибутам таблицы A, тем самым такой оператор удаляет из таблицы A кортежи, у которых значение атрибута соединения отсутствует в соответствующем атрибуте таблицы B. Очевидно, такие операторы всего лишь сокращают реляционное выражение.

Примеры использования реляционной алгебры

Для демонстрации примеров будем использовать базу данных Поставщики–Детали–Проекты, которая была предложена Дж. Дейтом, и широко используется для изучения систем баз данных. Во всех примерах этой главы использовались фрагменты подобной базы данных.

Схема базы данных и таблицы, содержащие учебные данные, приведены на рис. 4.20 и рис. 4.21 соответственно.

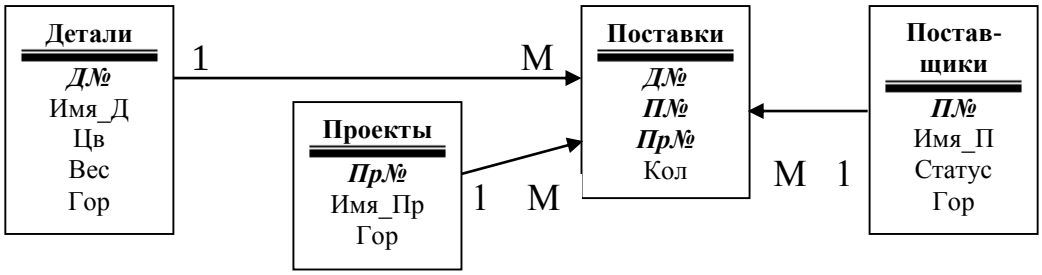


Рис. 4.20. Схема базы данных Поставщики–Детали–Проекты

Процесс проектирования баз данных, включающий анализ предметной области, разработку инфологической модели и проверку отношений на соответствие требованиям нормальных форм, подробно рассмотрен в курсе лекций по дисциплине «Базы данных».

Поставки

П№	Д№	Пр№	Кол
П1	Д1	Пр1	200
П1	Д1	Пр4	700
П2	Д3	Пр1	400
П2	Д3	Пр2	200
П2	Д3	ПР3	200
П2	Д3	ПР4	500
П2	Д3	ПР5	600
П2	Д3	ПР6	400
П2	Д3	ПР7	800
П2	Д5	ПР2	100
П3	Д3	ПР1	200
П3	Д4	ПР2	500
П4	Д6	ПР3	300
П4	Д6	ПР7	300
П5	Д2	ПР2	200
П5	Д2	ПР4	100
П5	Д5	ПР5	500
П5	Д5	ПР7	100
П5	Д6	ПР2	200
П5	Д1	ПР4	100
П5	Д3	ПР4	200
П5	Д4	ПР4	800
П5	Д5	ПР4	400
П5	Д6	ПР4	500

Детали

Д№	Имя_Д	Цв	Вес	Гор
Д1	Тестер	Черный	250	Минск
Д2	Дозиметр	Серый	700	Борисов
Д3	Радиометр	Черный	1400	Гродно
Д4	Часы	Желтый	140	Минск
Д5	Рулетка	Красный	200	Брест
Д6	Лом	Черный	5000	Варшава

Поставщики

П№	Имя_П	Статус	Гор
П1	Волк	20	Брест
П2	Заяц	10	Минск
П3	Лев	30	Гродно
П4	Лиса	20	Минск
П5	Бык	30	Брест

Проекты

Пр№	Имя_П	Гор
Пр1	Спутник	Минск
Пр2	Корунд	Минск
Пр3	Лес	Брест
Пр4	Шина	Бобруйск
Пр5	Азот	Гродно
Пр6	Электро	Молодечно
Пр7	Кристалл	Пинск

Рис. 4.21. База данных Поставщики–Детали–Проекты

Примеры

1. Получить имена поставщиков, которые поставляют деталь Д2.

Требуется построить таблицу с одним полем Имя_П, используя таблицы Поставки и Поставщики:

((Поставки JOIN Поставщики) WHERE Д№=’Д2’) [Имя_П];

2. Получить имена поставщиков, которые поставляют по крайней мере одну черную деталь.

Здесь также надо построить унарную таблицу с атрибутом Имя_П, но необходимые нам сведения находятся в трех таблицах – Детали, Поставки и Поставщики:

(((Детали WHERE Цв = ‘Черный’) JOIN Поставки) JOIN Поставщики) [Имя_П]

или

(((Детали WHERE Цв = ‘Черный’) [Д№] JOIN Поставки) [П№] JOIN Поставщики) [Имя_П].

Во втором выражении предлагается делать проекции после операций выборки и соединения, уменьшая при этом степень промежуточных таблиц. Благодаря этому повышается эффективность алгоритма.

3. Получить имена поставщиков, которые поставляют все детали.

Наиболее простое решение этой задачи получаем при использовании реляционной операции деления:

((Поставки [П№,Д№] DIVIDED BY Детали [Д№]) JOIN Поставщики) [Имя_П]

Разделив бинарную таблицу Поставки [П№,Д№] на унарную Детали [Д№], получим унарную таблицу с атрибутом П№ и телом, содержащим кортежи с теми номерами поставщиков П№, которые в бинарной таблице сцеплены со всеми значениями атрибута Д№ в таблице Детали [Д№].

4. Получить номера поставщиков, которые поставляют по крайней мере все те детали, которые поставляет поставщик П2.

Алгоритм решения похож на алгоритм прошлой задачи:

Поставки [П№,Д№] DIVIDEDBY (Поставки WHERE Имя_П=’П2’) [Д№]

5. Получить имена поставщиков, которые не поставляют деталь Д2.

В этой задаче можно легко найти номера поставщиков, поставляющих деталь д2, и вычесть их из таблицы, содержащей все номера поставщиков:

((поставщики [п№] MINUS (поставки WHERE д№=’д2’) [п№])
JOIN (поставщики) [имя_п]

6. Более сложный пример: Получить номера поставщиков, у которых максимальное значение поставок как минимум в 2 раза превышает среднее значение поставок деталей остальными поставщиками.

Итак, требуется построить унарную таблицу с атрибутом П№. Вся необходимая информация находится в таблице Поставки. Выбрать нужных поставщиков мы сможем, если построим таблицу, содержащую для каждого поставщика данные о величине его максимальной поставки и о средних поставках всех остальных. Тогда операцией выборки сможем отобрать поставщиков, у которых первая величина более чем в 2 раза превышает вторую.

На первом шаге строим таблицу с данными о максимальных поставках каждого поставщика:

SUMMARIZE поставки BY(П№) ADD MAX(Кол) AS m

Получили бинарную таблицу с атрибутами П№ и m. Далее добавим в эту таблицу новый атрибут, в котором для каждого поставщика будут находиться значения средних поставок всех остальных поставщиков. Для этого удобно использовать оператор расширения EXTEND:

```
EXTEND (SUMMARIZE поставки BY(п№) ADD MAX(Кол) AS m) ADD
AVG(((поставки RENAME П№ AS x) WHERE x<>П№) [кол]) AS ср
```

Далее в полученной таблице выбираем строки, в которых m>=2*ср и делаем проекцию по полю п№:

```
(EXTEND (SUMMARIZE поставки BY(п№) ADD MAX(кол) AS m) ADD
AVG(((поставки RENAME п№ AS x) WHERE x<>п№) [кол]) AS ср) WHERE
m>=2*ср)[п№]
```

Полученное реляционное выражение позволяет построить таблицу, удовлетворяющую условию задачи.

Задачу можно усложнить, если потребовать найти поставщиков, у которых максимальное (или минимальное) значение поставок *меньше* среднего. Если использовать предложенный алгоритм, то мы потеряем поставщиков, которые не поставляли детали. Решите такую задачу самостоятельно.

Итак, в главе очень кратко изложены основы реляционной алгебры. Современные системы управления базами данных построены, в основном, на реляционной модели данных и алгебра является фундаментом всех алгоритмов обработки данных. Для работы с данными используется язык программирования баз данных SQL, знакомству с которым посвящена следующая глава.

5. Функции

Функции используются для описания любых процессов, при которых элементы одного множества каким-то образом переходят в элементы другого. Такие преобразования элементов представляют собой основу многих вычислительных процессов.

Функции представляют собой специальный тип бинарных отношений. Мы рассмотрим некоторые свойства функций, имеющие особое значение для обработки данных, а также закон, известный как *принцип Дирихле*. Этот принцип позволяет решать не связанные друг с другом явным образом вычислительные задачи.

5.1. Свойства функций

Отношения применяются для описания связей между парами элементов, выбранных из двух множеств A и B. Функции – это частный случай бинарных отношений, на которые наложены дополнительные ограничения.

Функцией из множества A в множество B называется бинарное отношение, при котором *каждый* элемент множества A связан с *единственным* элементом множества B. Другими словами, для каждого $a \in A$ существует ровно одна пара из отношения вида (a, b).

В графических терминах определение функции описывается графом, у которого из *каждой* вершины, изображающей элементы множества A, выходит *ровно одна* стрелка.

Например, на рис. 5.1 изображен граф, представляющий функцию из множества $\{a, b, c\}$ в множество $\{1, 2\}$, состоящую из пар (a, 1), (b, 1) и (c, 2).

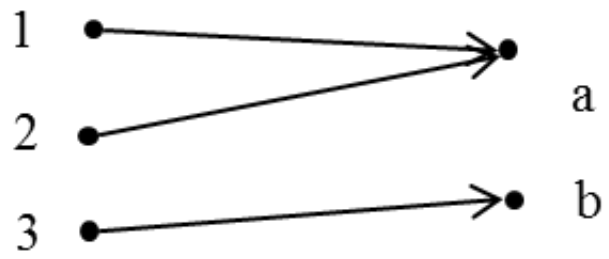


Рисунок 5.1 Пример графа функции

Пример 5.1. Определите, какие из следующих отношений между множествами $A = \{a, b, c\}$ и $B = \{1, 2, 3\}$ являются функциями из множества A в B:

а) $f = \{(a, 1), (a, 2), (b, 3), (c, 2)\}$;

б) $g = \{(a, 1), (b, 2), (c, 1)\}$;

в) $h = \{(a, 1), (c, 2)\}$.

Решение.

а) отношение f – не функция, поскольку элементу a соответствуют два разных элемента множества B: 1 и 2;

б) отношение g является функцией;

в) последнее отношение функцией не является, поскольку элементу b не соответствует ни одного элемента.

Пример 5.2. Какие из отношений являются функциями?

(а) «x – брат или сестра y» на множестве всех людей;

(б) отношение на множестве **Z**, заданное парами: $\{(x, x^2) : x \in Z\}$;

(в) отношение на множестве **R**, заданное парами: $\{(x, y) : x = y^2\}$.

Решение.

(а) Это не функция, поскольку есть люди с несколькими братьями и сестрами, а также бывают семьи с единственным ребенком.

(б) Это отношение – функция, поскольку по каждому целому числу x его квадрат x^2 определяется однозначно.

(в) Последнее отношение – не функция, т. к., например, обе упорядоченные пары: $(2,\sqrt{2})$ и $(2, -\sqrt{2})$ – ему принадлежат. Кроме того, в нем отсутствуют пары

(x, y) с отрицательными *x*.

Пусть *f* – функция из множества *A* в множество *B*. Поскольку для каждого элемента $x \in A$ существует единственным образом определенный элемент $y \in B$, такой, что $(x, y) \in f$, мы будем писать: $y = f(x)$ – и говорить, что функция *f* *отображает* множество *A* в множество *B*, а $f(x)$ называть *образом x* при отображении *f* или *значением f*, соответствующим аргументу *x*. Кроме того, можно написать $f: A \to B$, чтобы подчеркнуть, что функция *f* переводит элементы из *A* в элементы из *B*. Множество *A* принято называть *областью определения*, а *B – областью значений* функции *f* (часто говорят, что функция *f определена* или *задана* на множестве *A* со значениями в множестве *B*).

Для уточнения подмножества элементов, в которые переводятся элементы из *A* функцией *f*, вводят понятие «образ» или «множество значений функции». А именно, *множеством значений* функции *f* называется подмножество в *B*, состоящее из образов всех элементов $x \in A$. Оно обозначается как $f(A)$ и формально определяется так: $f(A) = \{f(x): x \in A\}$. Диаграмма Венна на рис. 5.2 служит удобной иллюстрацией функции, определенной на множестве *A* со значениями в множестве *B*.

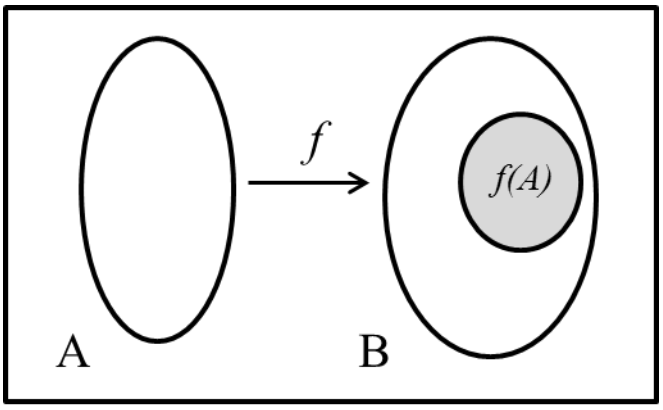


Рисунок 5.2. Диаграмма Венна функции $f: A \to B$

Когда мы работаем с функцией $f: A \to B$, где *A* и *B* – бесконечные множества, мы не можем нарисовать граф этого отношения. В этом случае используют традиционное графическое представление функции –*график*. Например, график функции $f: R \to R$, заданной формулой $f(x) = x^2$ и отображающей элементы

множества вещественных чисел в множество вещественных чисел, изображен на рисунке 5.3. Горизонтальная ось помечается знаком x и обозначает множество определения функции R. Вертикальная ось помечается знаком y и представляет область значений функции (в нашем случае тоже R).

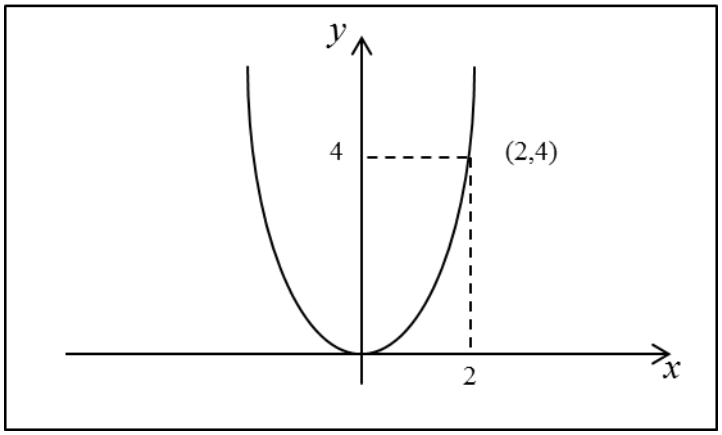


Рисунок 5.3. График функции y = f(x)

Кривая на рисунке, т.е. график функции, состоит из тех точек (x, y) прямого произведения $R \times R$, для которых $y = f(x)$. Например, $f(2) = 4$ и точка $(2, 4)$ лежит на этой кривой, что видно на рисунке.

Далее рассмотрим некоторые важные свойства функций.

Пусть f: $A \to B$ – функция. Функция называется *инъективной* (или *инъекцией,* или *взаимно однозначной*), если

$$f(a_1) = f(a_2) \Rightarrow a_1 = a_2 \text{ для всех } a_1, a_2 \in A.$$

Это определение логически эквивалентно тому, что

$$a_1 \neq a_2 \Rightarrow f(a_1) \neq f(a_2),$$

т.е. у инъективной функции нет повторяющихся значений. Иными словами, разные входные данные дают различные выходные данные.

Функция f называется *сюръективной*, или *сюръекцией*, если множество ее значений совпадает с областью значений. Это означает, что для каждого $b \in B$ найдется такой $a \in A$, что $b = f(a)$. Таким образом, каждый элемент области значений является образом какого-то элемента из области определения f.

Функция f называется *биективной*, или *биекцией,* если она инъективна и сюръективна.

Пример 5.3. Определите, какие из функций, изображенных на рис. 5.4, инъективны, а какие сюръективны. Перечислите все биекции.

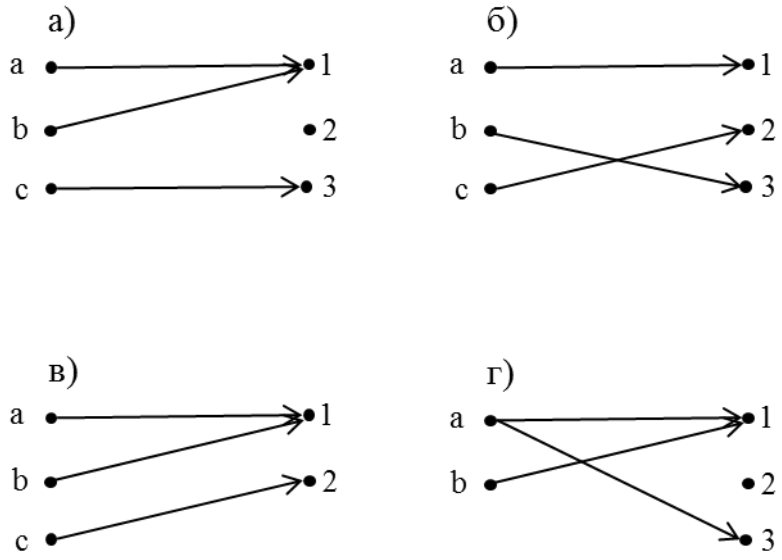


Рисунок 5.4. Графы функции для примера5.3.

Решение.

(а) Данная функция не инъективна, поскольку значение 1 соответствует как *a,* так и *b*. Она не является и сюръекцией, ввиду того что в элемент 2 ничего не отображается.

(б) Данная функция инъективна, т.к. не имеет повторяющихся значений. Она же и сюръективна, поскольку множество ее значений совпадает со всей областью определения.

(в) Значение 1 эта функция принимает как на *a*, так и на *b*. Следовательно, она не инъективна. Однако данная функция сюръективна, поскольку в ее множество значений входят все элементы области определения.

(г) Последняя функция инъективна, но не сюръективна.

Таким образом, только в случае (б) мы имеем биективную функцию.

Пример 5.4. Покажите, что функция h: Z→Z, заданная формулой $h(x) = x^2$, не инъективна и не сюръективна.

Решение. Заметим, что данная функция не совпадает с функцией f: R→R, заданной той же формулой $f(x) = x^2$, чей график был приведен выше. Несмотря на одинаковые формулы, области определения и значений функции h

ограничены только целыми числами. Фактически график *h,* изображенный на рис 5.5, состоит из серии изолированных точек.

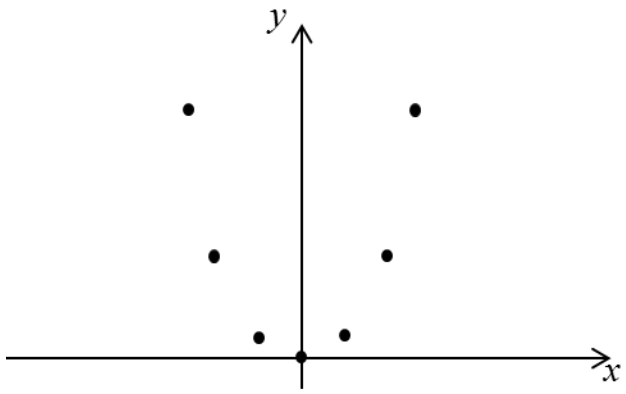


Рисунок 5.5. График функции h(x) = x2, определенной на множестве целых чисел Z.

Чтобы показать, что *h* не инъективна, достаточно найти такие разные целые числа $a_1 \neq a_2$, для которых $h(a_1) = h(a_2)$. На графике видно много таких пар целых чисел. Например, $a_1 = 2$ и $a_2 = -2$.

Что касается противоречия с сюръективностью, то можно найти такое число, которое содержится в области значений (**Z**), но не является значением функции *h* (не является квадратом целого числа). Если бы область определения и область значений функции были заданы на множестве рациональных чисел, то функция была бы сюръективной.

Пример 5.5. Покажите, что функция *k*: **R→R,** заданная формулой $k(x) = 4x + 3$, является биекцией.

Решение. Предположим, что $k(a_1) = k(a_2)$, т. е. $4a_1 + 3 = 4a_2 + 3$.

Из равенства следует, что $a_1 = a_2$. Значит, *k* – инъекция.

Пусть $b \in \mathbf{R}$. Покажем, что найдется такое вещественное число $a \in \mathbf{R}$, что $k(a) = b$. Ясно, что в качестве *a* можно взять $a = (b - 3)/4$. Итак, *k* – сюръективная функция. Поскольку *k* является одновременно и сюръекцией и инъекцией, то она – биективная функция.

5.2. Обратные функции и композиция функций

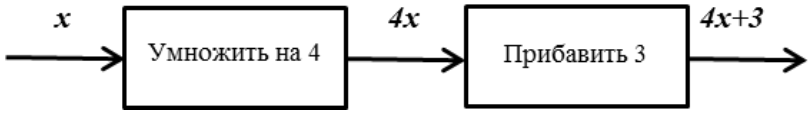
Напомним, что любая функция f: $A \to B$ – это бинарное отношение. Поэтому мы можем построить обратное отношение f^{-1}. Если при этом мы снова получим функцию, то исходную функцию будем называть *обратимой функцией,* а f^{-1}: $B \to A$ *– обратной функции.*

Функция f состоит из пар вида (a, b), где $b = f(a)$. Когда f обратима, обратная функция f^{-1} состоит из пар (b, a), где $a = f^{-1}(b)$. Значит, обратимая функция должна удовлетворять условию: если $f(a) = b$, то $f^{-1}(b) = a$. Другими словами, обратная функция переворачивает действие исходной.

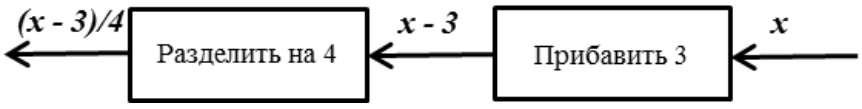
Пример 5.6. Какие из функций примера 5.4 обратимы?

Решение. Обратное отношение получается простым обращением стрелок в орграфе, его представляющем. Очевидно, только в случае (б) мы имеем обратимую функцию.

Идея переориентации стрелок может быть использована для обращения не очень сложных функций. Рассмотрим функцию k: $\mathbf{R} \to \mathbf{R}$, $k = 4x + 3$ (см. пример 5.5). Действие k можно разбить на несколько шагов:



Две элементарные команды: «умножить на 4» и «прибавить 3» – легко обращаются: «разделить на 4» и «вычесть 3» соответственно. Таким образом, обращение стрелок дает обратную функцию:



Итак, k^{-1}: $\mathbf{R} \to \mathbf{R}$ задается формулой $k^{-1} = (x - 3) / 4$.

Такой же ответ может быть получен и алгебраическим путем. Пусть $y = k(x)$, так что $x = k^{-1}(y)$. Нам известно, что $y = 4x + 3$. Выразив из этого равенства x, получим, что $x = (y - 3) / 4$. Следовательно, $k^{-1}(y) = (y - 3) / 4$ или, поскольку мы обычно используем x в качестве аргумента, $k^{-1}(x) = (x - 3) / 4$, что и было получено ранее.

Как мы видели в примерах, обе функции, обладающие свойством обратимости, были биективными. Это не случайное совпадение: обратимы только биекции. Сейчас мы докажем критерий обратимости для общей функции f: $A \to B$.

Теорема. Функция f обратима тогда и только тогда, когда она биективна.

Доказательство. Доказательство состоит из двух частей.

Сначала мы докажем, что биективная функция обратима. Пусть f: $A \to B$ – биекция. Как отношение, f можно определить с помощью предикатов: $f = \{(a, b): a \in A$ **и** $f(a) = b\}$. По определению обратного отношения имеем: $f^{-1} = \{(b,a): a \in A$ **и** $f(a) = b\}$.

Поскольку функция f сюръективна, для любого элемента $b \in B$ найдется такой $a \in A$, что $f(a) = b$. Кроме того, ввиду инъективности функции f такой элемент a определяется по b единственным образом. Следовательно, все пары отношения f^{-1} обладают тем свойством, что каждый элемент множества B соответствует единственному элементу множества A. А это, по определению, и означает, что f^{-1} является функцией, как и утверждалось.

Теперь покажем, что обратимая функция обязана быть биективной. Предположим, что обратное отношение f^{-1} – функция. Тогда для любого $b \in B$ существует единственный элемент $a \in A$, для которого $(b, a) \in f^{-1}$. Следовательно, $(a, b) \in f$, т.е. $b = f(a)$. Этим доказана сюръективность f.

Для проверки инъективности функции f поступим следующим образом. Предположим, что $f(a_1) = f(a_2)$. Тогда обе пары $(f(a_1),a_1)$ и $(f(a_2),a_2)$ лежат в f^{-1}. Так как f^{-1} является функцией, имеет место равенство: $a_1 = a_2$, так что f инъективна.

Таким образом, f является биекцией, как и утверждалось. Теорема доказана: обратимая функция биективна и, с другой стороны, биективная функция обратима.

Пример 5.7. Пусть $A = \{x: x \in \mathbf{R}$ **и** $x \neq 1\}$ и f: $A \to A$ задается формулой $f(x) = \frac{x}{x-1}$. Показать, что f биективна и найти обратную ей функцию.

Решение. Предположим, что $f(a_1) = f(a_2)$. Тогда

$$\frac{a_1}{a_1-1} = \frac{a_2}{a_2-1} \quad .$$

Значит, $a_1 = a_2$. Следовательно, f инъективна.

Пусть $b \in A$ – элемент области значений f. Найдем элемент a из множества A, удовлетворяющий условию: $f(a) = b$, т.е. $\frac{a}{a-1} = b$. Разрешая полученное уравнение относительно a, найдем $a = \frac{b}{b-1}$.

Нам удалось найти элемент $a = \frac{b}{b-1} \in A$, для которого $f(a) = b$. Это свидетельствует о сюръективности f. Итак, мы показали, что функция f как сюръективна, так и инъективна. Значит, она является биекцией.

Обратная функция определяется условием: $f^{-1}(b) = a$ всегда, когда $f(a) = b$. Но, как мы уже выяснили при доказательстве сюръективности f, $a = \frac{b}{b-1}$. Таким образом, $f^{-1}: A \to A$, $f^{-1}(x) = \frac{x}{x-1}$, т.е. функция f обратна сама себе.

Познакомимся с понятием композиции функций. Если $f: A \to B$ и $g: B \to C$ – функции, то композиция функций $g \circ f$ между A и C состоит из пар вида (a,c), где для некоторого $b \in B$ $(a,b) \in f$ и $(b,c) \in g$. Однако элемент $b = f(a)$ однозначно определяется по a, поскольку f – функция. Более того, элемент $c = g(b)$ также однозначно определяется по b (g тоже функция). Следовательно, элемент $c = g(f(a))$ единственным образом определяется элементом a и, стало быть, композиция функций f и g – снова функция.

Итак, композиция $g \circ f: A \to C$ является функцией, действующей по правилу $(g \circ f)(x) = g(f(x))$.

Пример 5.8. Рассмотрим две функции: $f: R \to R$, $f(x) = x^2$ и $g: R \to R$, $g(x) = 4x + 3$. Вычислить $g \circ f$, $f \circ g$, $f \circ f$ и $g \circ g$.

Решение. Все новые функции определены на R со значениями в R.

$(g \circ f)(x) = g(f(x)) = g(x^2) = 4x^2 + 3$;

$(f \circ g)(x) = f(g(x)) = f(4x + 3) = (4x + 3)^2 = 16x^2 + 24x + 9$;

$(f°f)(x) = f(f(x)) = f(x^2) = x^4$,

$(g°g)(x) = g(g(x)) = g(4x + 3) = 4(4x + 3) + 3 = 16x + 15$.

В современных языках программирования функции используются очень широко. Они дают нам возможность выделить отдельные вычисления в *подпрограммы*. В большинстве языков есть специальные библиотеки с наиболее часто применяющимися функциями, такими как $\sin x$, $\log x$, $|x|$ и т.д. Кроме того, в них легко создавать собственные функции.

В некоторых особенно мощных языках, известных как языки функционального программирования, основные операторы определены в терминах функций. Главная особенность таких языков – возможность построения новых, более сложных операторов из основных. Чтобы уметь это делать, необходимо в совершенстве овладеть композицией функций.

5.3. Языки функционального программирования

Языки функционального программирования оперируют символами, используя в качестве операторов основные примитивные функции. Такие языки успешно применяются для создания экспертных систем, моделирования общесмысловых рассуждений, построения естественных языковых интерфейсов и поддерживают исследования в области компьютерной речи и изображений.

Сила этих языков заключается в их способности строить сложные процедуры из простых, комбинируя основные примитивные функции. В результате созданные алгоритмы производят сложные вычисления, используя доступные и весьма примитивные основные функции. Здесь мы опишем некоторый функциональный алгоритм, который оперирует с текстом так, как это может происходить в простом текстовом редакторе.

Пусть C = {«а», «б», «в», …, «я»} – множество *литер* нижнего регистра клавиатуры компьютера с кириллицей, а P обозначает множество целых чисел {0, 1, 2, ...}. Обозначим через S множество *строк* (последовательностей) этих литер. Например, «мышь» – элемент множества S, как и пустая строка « ».

Допустим, что мы можем использовать следующие основные примитивные функции:

CHAR: S→C, где CHAR(S) – первая буква непустой строки s.

REST: S→S, где REST(S) – строка, полученная из непустой строки s удалением ее первой литеры.

ADDCHAR: C×S→S, где ADDCHAR(C,S) – строка, полученная из s добавлением к ее началу литеры C.

LEN: S→P, где LEN(S) – число литер в строке S.

Поскольку это наши специфические базисные функции, нас не должно волновать, как они устроены на более низком уровне. Можно считать, что доступ к ним осуществляется практически так же, как к обыкновенным функциям на калькуляторе – простым нажатием кнопки.

Пример 5.9. Вычислить:

CHAR(S),
LEN(REST(S)) и
ADDCHAR(CHAR(S), ADDCHAR(«л», REST(S))), если S = «сон».

Решение.

CHAR(S) = CHAR(«сон») = «с»;
LEN(REST(s)) = LEN(REST(«сон»)) = LEN(«он») = 2;
ADDCHAR(CHAR(s), ADDCHAR(«л», REST(«сон»))) =
= ADDCHAR(«с», ADDCHAR(«л», REST(«сон»))) =
= ADDCHAR(«с», ADDCHAR(«л», «он»)) = ADDCHAR(«с», «лон») = «слон».

Можно заметить, что ADDCHAR(CHAR(s), ADDCHAR(c, REST(s))), где S – произвольная непустая строка, c – любая литера, вставляет новую литеру «с» непосредственно после первой литеры строки S.

Пример 5.10. Функция THIRD: S→C нужна для определения третьей по счету литеры в строке из трех и более литер. Выразите функцию THIRD через CHAR и REST.

Решение. Третья литера произвольной строки *s* необходимой длины совпадает с первым символом строки, полученной из *s* удалением первых двух. Поэтому THIRD(s) = CHAR(REST(REST(s))).

Пример 5.11. Опираясь на базисные примитивные функции, найдите функцию REVERSE2: S→S, которая переставляет первые две литеры в строке

длины 2 и более.

Решение. Пусть *s* – вводимая строка. Первая литера выводимой строки равна CHAR(REST(s)), а вторая – CHAR(s). Остальные литеры остаются неизменными и совпадают с REST(REST(s)). Следовательно, значение функции REVERSE2(s) выражается следующим образом:

ADDCHAR(CHAR(REST(s)), ADDCHAR(CHAR(s), REST(REST(s)))).

Пример 5.12. Проследите следующий алгоритм, взяв в качестве вводной строки *s* = «клоп».

```
Input s
begin
    u:= « »;
    t:= s;
    i:=0;
    while i < LEN(s) do
    c:=CHAR(t);
t:=REST(t);
u:=ADDCHAR(c, u);
i:=i + 1;
end
Output u
```

Что делает этот алгоритм?

Решение. Проследим за изменением значений переменных c, *t, u* и *i* в течение работы цикла while и сведем полученную информацию в табл. 5.1.

Таблица 5.1 Действие алгоритма из примера 5.12.

Проход цикла	**c**	**t**	**u**	**i**	***i* < 4?**
0	–	«клоп»	« »	0	Да
1	«к»	«лоп»	«к»	1	Да
2	«л»	«оп»	«лк»	2	Да
3	«о»	«п»	«олк»	3	Да
4	«п»	« »	«полк»	4	Нет

Алгоритм переставляет литеры строки в обратном порядке.

В свою очередь, создав программы по алгоритмам, полученным в примерах 5.9–5.12, мы можем использовать их как примитивные функции.

Как можно заметить, некоторые из наших функций определены не для всех вводимых строк. Например, REST не определена на пустой строке *s* = « », а REVERSE2 не определена на строках длины меньше 2. Причина заключается в

том, что желательно ограничиться малым числом стандартных множеств, из которых берутся входные данные. Поэтому используются так называемые *частично вычислимые функции*. У них стандартные входные и выходные данные, но ограничены области определения и значений.

Например, частично вычислимая функция REST по существу определяется так: REST: S→S, область определения: $s \in$ S и $s \neq$ « », где REST(s) – строка, полученная из *s* удалением ее первой литеры.

Работая с композицией частично вычислимых функций, необходимо быть очень внимательным. Поскольку, например, функция REST(REST(s)) не определена на строках длины 1 или меньше, то область определения композиции REST°REST – $s \in$ S и LEN(s) > 1.

5.4. Принцип Дирихле

Пусть f: $A \to B$ – функция, причем как A, так и B, – конечные множества. Предположим, что A состоит из n элементов: a_1, a_2, ..., a_n. Принцип Дирихле гласит, что если $|A| > |B|$, то по крайней мере одно значение f встретится более одного раза. Проще говоря, найдется пара элементов $a_i \neq a_j$, для которых $f(a_i) = f(a_j)$. (Допуская некоторую вольность, принцип Дирихле можно переформулировать в легко запоминающейся форме: нельзя рассадить 10 зайцев в 9 клеток так, чтобы в каждой клетке сидел один заяц.)

Чтобы убедиться в истинности принципа, предположим, что для любой пары разных индексов $i \neq j$ мы имеем: $f(a_i) \neq f(a_j)$. Тогда множество B содержит по крайней мере n различных элементов: $f(a_1)$, $f(a_2)$,…, $f(a_n)$. И уж во всяком случае, $|B| \geq n$, что противоречит предположению: $n = |A| > |B|$. Следовательно, есть хотя бы два разных элемента a_i, $a_j \in A$, для которых $f(a_i) = f(a_j)$.

Пример 5.13. В аудитории присутствует 15 человек. Покажите, что по крайней мере у двоих из них день рождения в одном и том же месяце.

Решение. Множество людей в аудитории обозначим буквой A, а множество всех 12 месяцев обозначим через B. Рассмотрим функцию f: $A \to B$,

сопоставляющую каждому человеку в аудитории месяц его рождения. Так как $|A| = 15$, а $|B| = 12$, то $|A| > |B|$. По принципу Дирихле функция f должна иметь повторяющиеся значения, т.е. найдутся два человека с одним и тем же месяцем рождения.

Задачу из примера 5.13 легко решить и менее формальным рассуждением. Дано 15 человек и 12 месяцев. Поэтому совершенно очевидно, что хотя бы двое из них родились в один и тот же месяц. При этом трудно понять, зачем нам применять формальное рассуждение, как это было в предыдущем примере. Однако, как мы увидим дальше, более сложные задачи могут быть решены только с помощью принципа Дирихле, если, конечно, нам удастся обнаружить подходящую функцию. Поиск нужной функции – всегда самая трудная часть решения. Ключевая идея, на которой основан принцип Дирихле, состоит в том, что функция f размещает некоторое количество объектов (элементов множества A) в меньшее число клеток (элементы множества B). Поэтому по крайней мере два объекта попадут в одну.

Пример 5.14. Какое наименьшее число фамилий должно быть записано в телефонном справочнике, чтобы с гарантией можно было утверждать, что хотя бы две фамилии начинаются с одной и той же буквы и заканчиваются одинаковыми буквами?

Решение. Пусть A – множество фамилий в справочнике, а B – множество пар букв, выписанных из стандартного алфавита русского языка, насчитывающего 33 буквы. Обозначим через f: $A \to B$ функцию, которая каждой фамилии справочника ставит в соответствие пару букв: первую и последнюю буквы фамилии. Например, f(*Кузнецов*)=(к, в). Множество B содержит $33 \times 33 = 1089$ пар букв. Принцип Дирихле гарантирует нам, что если $|A| > |B| = 1089$, то найдется по крайней мере две фамилии, начинающиеся и оканчивающиеся на одинаковые буквы. Поэтому телефонный справочник должен содержать не менее 1090 фамилий (если учесть, что фамилии не могут начинаться с букв «Ы», «Ь» и «Ъ», то требуемый объем справочника окажется меньше).

Пример 5.15. Покажите, что какие бы пять цифр из 1, 2, 3, 4, 5, 6, 7 и 8 мы

ни выбрали, найдется хотя бы одна пара цифр, сумма которых равна 9.

Решение. Перечислим пары цифр, дающих в сумме 9.

{1, 8}, {2, 7}, {3, 6}, {4, 5}.

Обозначим через A множество выбранных пяти цифр (не важно каких конкретно), а через B следующее множество пар:

B= {{1, 8}, {2, 7}, {3, 6}, {4, 5}}.

Рассмотрим функцию f: $A \to B$, сопоставляющую каждой цифре из пятерки пару из множества B, которая в ней содержится. Например, $f(3) = \{3, 6\}$. По принципу Дирихле хотя бы две цифры из множества A попадут в одну и ту же пару. Значит, две из пяти цифр дадут в сумме 9.

Принцип можно обобщить следующим образом. Рассмотрим функцию f: $A \to B$, где A и B – конечные множества. Если $|A| > k|B|$ для некоторого натурального числа k, то найдется такое значение функции f, которое она будет принимать по крайней мере $k+1$ раз.

Это утверждение верно потому, что если каждое значение функция f принимает не более чем k раз, то все множество A состоит не более чем из $k|B|$ элементов.

Пример 5.16. Какое наименьшее число фамилий должно быть записано в телефонном справочнике, чтобы с гарантией можно было утверждать, что хотя бы пять фамилий начинаются с одной и той же буквы алфавита и заканчиваются одинаковыми буквами?

Решение. Пусть f: $A \to B$ – функция, которая каждой фамилии ставит в соответствие пару букв – первую и последнюю в фамилии (Пример 5.10). Как мы уже подсчитали, B состоит из 1089 элементов. Чтобы по крайней мере пять фамилий начинались и оканчивались одинаковыми буквами, нам нужно, чтобы $|A| > 4|B| = 4356$. Таким образом, телефонный справочник должен содержать не менее чем 4357 абонентов.

Пример 5.17. Покажите, что в любой группе из шести человек найдутся трое, знакомые друг с другом, или наоборот, не знающие друг друга.

Решение. Пусть x – один из шести людей, A – множество оставшихся пяти людей в группе и $B = \{0, 1\}$. Определим функцию f: $A \to B$ по правилу:

$$f(a) = \begin{cases} 0, \text{если } a \text{ не знаком с } x, \\ 1, \text{если } a \text{ знаком с } x. \end{cases}$$

Поскольку $5 = |A| > 2|B|$, то найдется три человека, которые либо все знакомы с x, либо все трое его не знают.

Предположим теперь, что три человека a, b и c знакомы с x. Если все три друг с другом не знакомы, то мы получаем решение задачи. В противном случае какая-то пара, скажем, a и b, знает друг друга. Но они же знакомы и с x. Значит, трое людей: a, b и x – хорошие знакомые. Аналогично разбирается случай, когда нашлась тройка людей, которые с x не знакомы.

Предыдущие примеры наглядно свидетельствуют, что применение принципа Дирихле требует аккуратной постановки задачи и, довольно часто, тонких логических рассуждений. Удачно, что в наших примерах было сравнительно несложно подсчитать количество элементов в множествах A и B. Так бывает далеко не всегда и в седьмой главе мы рассмотрим некоторые методы пересчета, которые предоставляют нам возможность определять мощность конечных множеств, чьи элементы выбираются определенными способами.

6. Булева алгебра

Булева алгебра – это область математики, изучающая методы логического анализа. Операции и законы булевой алгебры применяются к логическим символам так же, как обычная алгебра оперирует символами, представляющими численные величины.

В главе будут изучены основы булевой алгебры – множество {0, 1} с определенными на нем операциями дизъюнкции, конъюнкции и отрицания и законами преобразования выражений. Будет показана связь между булевой алгеброй и логикой высказываний с одной стороны, и алгеброй множеств – с другой. Мы покажем, как булевы выражения могут быть записаны в стандартной форме, носящей название «дизъюнктивная нормальная форма». После этого будет описан один из способов минимизации булевой функции, называемый «карта Карно».

6.1. Основы булевой алгебры

Простейшая булева алгебра состоит из множества $B = \{0, 1\}$ вместе с определенными на нем операциями *дизъюнкции* ($\vee$), *конъюнкции* ($\wedge$) и *отрицания* ($\bar{}$). Результат применения этих операций к элементам множества $B = \{0, 1\}$ показан в таблице 6.1.

Таблица 6.1. Операции булевой алгебры

p	**q**	**конъюнкция p∨q**	**дезъюнкция p∧q**	**отрицание** $\bar{p}$
0	0	0	0	1
0	1	1	0	
1	0	1	0	0
1	1	1	1	

Эта таблица напоминает таблицы истинности логических операций **не**, **или**, **и**, с которыми мы встретились, изучая логику. Действительно, мы можем легко трансформировать таблицу 9.1 в таблицы истинности, возникающие в логике высказываний, заменив переменные p и q на высказывания P и Q и используя

истинностные значения *Л* и *И* вместо 0 и 1 соответственно. Таким образом, $\bar{p}$ заменится на (не *P*), $p \vee q$ – на (*P* или *Q)*, а $p \wedge q$ – на (*P* и *Q)*. Поэтому и в контексте булевой алгебры мы будем называть такого сорта таблицы таблицами истинности. Мы можем комбинировать булевы переменные с помощью операций ($\vee$), ($\wedge$) и $(\bar{\ })$, получая *булевы выражения* так же, как мы строили составные высказывания из более простых, компонуя их с помощью логических операций.

Два булевых выражения называются *эквивалентными,* если они имеют одинаковые таблицы истинности. Вычисляя таблицы истинности, легко установить справедливость некоторых эквивалентностей, которые принято называть законами булевой алгебры (таблица 6.2).

Таблица 6.2. Законы булевой алгебры

Законы коммутативности:	$p \wedge q = q \wedge p$ $p \vee q = q \vee p$
Законы ассоциативности:	$p \wedge (q \wedge r) = (p \wedge q) \wedge r$ $p \vee (q \vee r) = (p \vee q) \vee r$
Законы дистрибутивности:	$p \wedge (q \vee r) = (p \wedge q) \vee (p \wedge r)$ $p \vee (q \wedge r) = (p \vee q) \wedge (p \vee r)$
Законы идемпотентности:	$p \wedge p = p$ $p \vee p = p$
Законы поглощения:	$p \wedge (p \vee q) = p$ $p \vee (p \wedge q) = p$
Законы де Моргана:	$\overline{(p \wedge q)} = \bar{p} \vee \bar{q}$, $\overline{(p \vee q)} = \bar{p} \wedge \bar{q}$

Пример 6.1. Докажите закон поглощения: $p \wedge (p \vee q) = p$ и $p \vee (p \wedge q) = p$.

Решение. Таблица истинности:

p	**q**	**p∨q**	**p∧(p∨q)**	**p∧q**	**p∨(p∧q)**
0	0	0	0	0	0
0	1	1	0	0	0
1	0	1	1	0	1
1	1	1	1	1	1

Из таблицы видна эквивалентность выражений *p*, $p \wedge (p \vee q)$ и $p \vee (p \wedge q)$, что доказывает закон поглощения.

Схожесть названий и форм законов булевой алгебры и соответствующих законов алгебры множеств не случайна. В таблице 6.3, приведенной ниже, устанавливается соответствие между булевыми операциями, логическими

операторами логики высказываний и операциями над множествами.

Таблица 6.3. Соответствие между разными математическими операторами

Логические операторы	**Операции над множествами**	**Булевы операции**
не	$\overline{}$	$\overline{}$
или	$\cup$	$\vee$
и	$\cap$	$\wedge$

Пример 6.2. Покажите, что булево выражение $\left(\overline{\bar{p} \wedge q}\right) \wedge (p \vee q)$ эквивалентно p.

Решение. Выполним преобразования:

$\left(\overline{\bar{p} \wedge q}\right) \wedge (p \vee q)$ = (закон де Моргана) =

$= \left(\overline{\bar{p}} \vee \bar{q}\right) \wedge (p \vee q) = (p \vee \bar{q}) \wedge (p \vee q)$ = (закон дистрибутивности) =

$= p \vee (\bar{q} \wedge q)$ = (т. к. $\bar{q} \wedge q = 0$) = p.

Булевой функцией от n булевых переменных $p_1, p_2, \ldots, p_n$ называется такая функция $f: B^n \rightarrow B$, что $f(p_1, p_2, \ldots, p_n)$ – булево выражение. Любая булева функция может быть записана в стандартном виде, который называется *дизъюнктивной нормальной формой.* Рассмотрим, например, булеву функцию $m(p, q, r)$ от булевых переменных p, q и r, чья таблица истинности дана ниже (таблица 6.4).

Таблица 6.4. Пример минтерма

p	***q***	***r***	***m***
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Функция m – пример *минтерма,* т. е. булевой функции, которая принимает значение 1 только на одном наборе значений аргументов. Так как согласно таблице истинности рассматриваемая функция $m(p, q, r) = 1$ только если $p = 0$, $q = 1$ и $r = 1$, то выполняется равенство $m(p, q, r) = \bar{p} \wedge q \wedge r$ (по определению конъюнкции функция $p \wedge q \wedge r$ принимает значение 1 только тогда, когда $p = q =$

$r = 1$. Поэтому $\bar{p} \wedge q \wedge r = 1 \Leftrightarrow p = 0,\ q = r = 1$). Выражение $\bar{p} \wedge q \wedge r$ называют *элементарной конъюнкцией*. Любой минтерм можно записать в виде элементарной конъюнкции, т.е. как конъюнкцию переменных p_i или их отрицаний.

Используя элементарные конъюнкции, можно записать любую булеву функцию как дизъюнкцию минтермов. Можно доказать, что такая запись (она называется *дизъюнктивной нормальной формой)* для каждой функции определена единственным образом. Рассмотрим булеву функцию трех переменных $f(p, q,$ r), чья таблица истинности приведена в таблице 6.5.

Таблица 6.5. Булева функция трех переменных

p	***q***	***r***	***m***
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Единицы последнего столбца в этой таблице соответствуют трем минтермам: $\bar{p} \wedge \bar{q} \wedge r;\ \bar{p} \wedge q \wedge r$ и $p \wedge \bar{q} \wedge \bar{r}$.

Таблица истинности функции f может быть получена наложением таблиц истинности выписанных минтермов.

Поскольку дизъюнкция с 1 «поглощает» все 0 (иными словами, $f_1 \vee f_2 \vee \ldots \vee f_n$ равно 1 тогда и только тогда, когда среди значений f_i найдется хотя бы одна 1), то наша функция f равна дизъюнкции трех минтермов:

$f(p,q,r) = (\bar{p} \wedge \bar{q} \wedge r) \vee (\bar{p} \wedge q \wedge r) \vee (p \wedge \bar{q} \wedge \bar{r})$.

Это и есть нормальная дизъюнктивная форма функции f. В той же форме можно записать произвольную булеву функцию с любым числом переменных.

Пример 6.3. Найти дизъюнктивную нормальную форму булевой функции $f = (p \wedge q) \vee (\bar{q} \wedge r)$.

Решение. Запишем таблицу истинности функции f.

Таблица 6.6. Булева функция трех переменных для примера 6.3

p	***q***	***r***	***m***

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

Выпишем минтермы: $\bar{p}\wedge\bar{q}\wedge r;\ p\wedge\bar{q}\wedge r;\ p\wedge q\wedge\bar{r};\ p\wedge q\wedge r$.

Следовательно, искомая дизъюнктивная нормальная форма:

$$f(p,q,r) = (\bar{p}\wedge\bar{q}\wedge r)\vee(p\wedge\bar{q}\wedge r)\vee(p\wedge q\wedge\bar{r})\vee(p\wedge q\wedge r).$$

Итак, любая булева функция может быть единственным образом представлена в виде дизъюнкции минтермов. Значит, каждая булева функция может быть выражена через две функции от двух аргументов (конъюнкцию и дизъюнкцию) и одной функции одной переменной (отрицание). Множество функций, через которые можно выразить любую булеву функцию, называется *полной системой функций.* Тогда множество булевых операторов {дизъюнкция, конъюнкция, отрицание} – полная система функций. Однако можно ограничиться и меньшим количеством функций. Например, по закону де Моргана $\left(\overline{p\vee q}\right)=\bar{p}\wedge\bar{q}$. Следовательно, дизъюнкцию можно заменить оператором конъюнкции и тремя операторами отрицания – $p\vee v=\overline{(\bar{p}\wedge\bar{v})}$. Значит, любую булеву функцию можно записать только с помощью двух операций, т.е. {конъюнкция, отрицание} – тоже полная система функций. Расплатой за уменьшение количества операторов, посредством которых записывается функция, становится громоздкость формул.

Пример 9.4. Функция **НЕ–И** определяется формулой: p **НЕ-И** $q=\left(\overline{p\vee q}\right)$.

Покажите, что {**НЕ-И**} – полная система функций.

Решение. Для решения задачи достаточно показать, что каждая из функций $\bar{p}$, $p\wedge q$ и $p\vee q$ может быть выражена через **НЕ-И**.

По закону идемпотентности $\bar{p}=\left(\overline{p\vee q}\right)=p$ **НЕ-И** p.

По закону де Моргана: $p\vee v=\overline{(\bar{p}\wedge\bar{v})}=\left(\overline{(p\ \text{НЕ}-\text{И}\ p)\wedge(q\ \text{НЕ}-\text{И}\ q)}\right)=$

$$= (p \text{ НЕ} - \text{И } p) \text{ НЕ} - \text{И } (q \text{ НЕ} - \text{И } q).$$

Наконец, используя найденное ранее отрицание,

$$p \wedge v = \overline{(\overline{p \wedge q})} = (\overline{p \text{ НЕ} - \text{И } q}) = (p \text{ НЕ} - \text{И } q) \text{ НЕ} - \text{И } (p \text{ НЕ} - \text{И } q).$$

Итак, {**НЕ-И**} – действительно полная система функций.

6.2. Карта Карно

Теперь научимся упрощать выражения для булевой функции. Под «упрощением» подразумевается эквивалентное выражение, использующее меньше символов, чем исходное. Метод состоит в упрощении дизъюнктивной нормальной формы булевой функции (при этом запись дизъюнктивной нормальной формы может оказаться более громоздкой, чем сама функция). Мы рассмотрим булевы функции не более чем от трех переменных. Используемый подход можно обобщить на функции с любым числом аргументов.

Прежде всего, записывая функцию, мы будем опускать символ конъюнкции $\wedge$ аналогично тому, как в обычной алгебре опускают символ умножения. Например: $\bar{p}\bar{q}r \vee \bar{p}qr \vee p\bar{q}\bar{r}$ вместо $(\bar{p} \wedge \bar{q} \wedge r) \vee (\bar{p} \wedge q \wedge r) \vee (p \wedge \bar{q} \wedge \bar{r})$.

Это выражение – дизъюнктивная нормальная форма. Его можно упростить следующим образом:

$\bar{p}\bar{q}r \vee \bar{p}qr \vee p\bar{q}\bar{r}$ = (по законам коммутативности и ассоциативности) =

$= (\bar{p}r\bar{q} \vee \bar{p}rq) \vee p\bar{q}\bar{r}$ = (по закону дистрибутивности) =

$= \bar{p}r(\bar{q} \vee q) \vee p\bar{q}\bar{r}$ = (т. к. $\bar{q} \vee q = 1$) $= \bar{p}r \vee p\bar{q}\bar{r}$.

Это выражение более простое, чем исходная функция. Оказывается, что последовательность шагов, которые мы совершили, упрощая функцию, можно делать почти автоматически. Заметим, что на первом шаге мы сделали перегруппировку: переставили и взяли в скобки два минтерма, отличающиеся только в одном символе. Закон дистрибутивности позволил нам на втором шаге вынести один минтерм за скобки, исключив из него булеву переменную q.

Упрощать функции можно с помощью *карты Карно* – метода, изобретенного в 1950-х гг. и применяемого для расчета логических схем.

Образно говоря, Карта Карно – это наглядная схема, предназначенная для обнаружения пар минтермов, которые можно сгруппировать и преобразовать в одно простое выражение.

В случае булевых функций трех переменных p, q и r карта Карно представляет собой таблицу с двумя строками и четырьмя столбцами (рис. 6.1). Столбцы обозначены конъюнкциями, которые можно получить из двух переменных p и q и их отрицаний, а строки – переменной r и ее отрицанием $\bar{r}$.

	pq	$\bar{p}q$	$\bar{p}\bar{q}$	$p\bar{q}$
r				
$\bar{r}$				

Рисунок 6.1 Карта Карно для функций трех переменных

Метки расставлены таким образом, что от столбца к столбцу в них происходит изменение ровно в одном символе. Ячейки карты Карно соответствуют восьми минтермам, которые можно построить из трех булевых переменных. Если нам дано булево выражение в дизъюнктивной нормальной форме, то в ячейки, соответствующие минтермам, участвующим в ней, мы записываем метку, например, цифру 1. Карта Карно булева выражения $\bar{p}\bar{q}r \vee \bar{p}qr \vee p\bar{q}\bar{r}$ изображена на рис. 6.2. Затем предлагается «группировать» пары минтермов, соответствующих помеченным «соседним» ячейкам в карте Карно (выделенные ячейки на рис. 6.2). Такая пара в нашем примере только одна. Она соответствует именно тем минтермам, которые мы объединили в сделанных ранее алгебраических преобразованиях.

	pq	$\bar{p}q$	$\bar{p}\bar{q}$	$p\bar{q}$
r		1	1	
$\bar{r}$				1

Рисунок 6.2. Карта Карно для булева выражения $\bar{p}\bar{q}r \vee \bar{p}qr \vee p\bar{q}\bar{r}$

Пример 9.5. Упростите булево выражение $pqr \vee \bar{p}\bar{q}\bar{r} \vee \bar{p}qr \vee pq\bar{r} \vee \bar{p}q\bar{r}$.

Решение. На рис. 6.3 изображена соответствующая карта Карно.

	pq	$\bar{p}q$	$\bar{p}\bar{q}$	$p\bar{q}$
r	1	1		
$\bar{r}$	1	1	1	

Рисунок 6.3. Карта Карно выражения $pqr \vee \bar{p}\bar{q}\bar{r} \vee \bar{p}qr \vee pq\bar{r} \vee \bar{p}q\bar{r}$

Из нее следует, что в данном выражении есть группа из четырех минтермов, которую мы обозначим через (А): $pqr \vee \bar{p}q\ r \vee pq\bar{r} \vee \bar{p}q\bar{r}$ и группа из двух минтермов, которую мы обозначим через (Б): $\bar{p}qr \vee \bar{p}\bar{q}\bar{r}$.

Сначала преобразуем группу (А):

$pqr \vee \bar{p}q\ r \vee pq\bar{r} \vee \bar{p}q\bar{r} = (p \vee \bar{p})\ q\ r \vee (p \vee \bar{p})\ q\bar{r} = qr \vee q\bar{r} = q(r \vee \bar{r}) = q$.

Теперь преобразуем группу (Б): $\bar{p}qr \vee \bar{p}\bar{q}\bar{r} = \bar{p}\bar{r}\ (q \vee \bar{q}) = \bar{p}\bar{r}$.

Таким образом, исходное выражение упрощается до $q \vee \bar{p}\bar{r}$.

Можно заметить, что минтерм $\bar{p}q\,\bar{r}$ используется дважды – в группах А и Б, хотя в исходном выражении он присутствует один раз. Этот искусственный прием основан на законе идемпотентности – $f = f \vee f$.

6.3. Функциональные схемы

Одно из основных приложений булевых функций лежит в области создания *функциональных схем,* которые можно реализовать в виде электронных устройств с конечным числом входов и выходов, причем на каждом входе и выходе могут быть только два значения – 0 или 1. Такие устройства собраны из *функциональных элементов* (вентилей), генерирующих основные булевы операции. Стандартные обозначения основных функциональных элементов показаны на рис. 6.4.

Соединяя вентили вместе, мы получаем *функциональную схему*. С ее помощью можно реализовать любую булеву функцию.

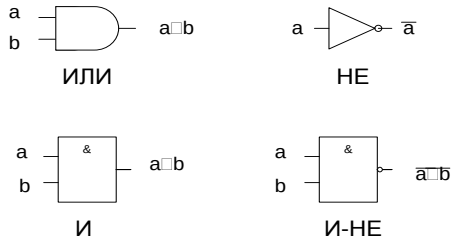


Рисунок 6.4. Стандартные обозначения функциональных элементов

Пример 9.6. Что получится на выходе функциональной схемы,

представленной на рис. 6.5?

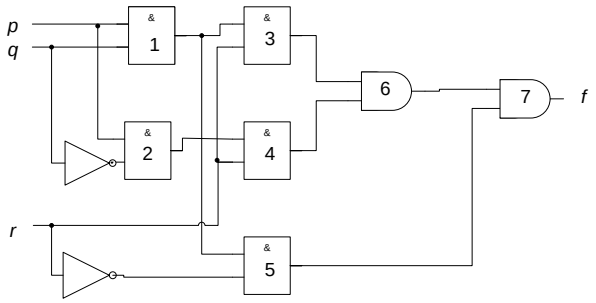


Рисунок 6.5. Функциональная схема для примера 9.6.

Решение. В табл. 6.7 перечислены входы и соответствующие выходы для каждого функционального элемента в соответствии с нумерацией из рис. 6.5.

Таблица 6.7. Значения на входе и выходе каждого вентиля в примере 6.6

Вентиль	**Вход**	**Выход**
1	p, q	pq
2	$p, \bar{q}$	$p\bar{q}$
3	pq, r	pqr
4	$p\bar{q}, r$	$p\bar{q}r$
5	$pq, \bar{r}$	$pq\bar{r}$
6	$pqr, p\bar{q}r$	$pqr \vee p\bar{q}r$
7	$pqr \vee p\bar{q}r, pq\bar{r}$	$pqr \vee p\bar{q}r \vee pq\bar{r}$

Таким образом, на выходе схемы получится функция $pqr \vee p\bar{q}r \vee pq\bar{r}$.

Диаграммы можно упростить, если использовать карту Карно для преобразования функции, полученной на выходе сложной схемы.

Пример 9.7. Упростим функцию, генерируемую схемой из примера 6.6, и найдем более простую функциональную схему, ее реализующую.

Решение. Карта Карно представлена на рис. 6.6.

	pq	$\bar{p}q$	$\bar{p}\bar{q}$	$p\bar{q}$
r	1			1
$\bar{r}$	1			

Рисунок 6.6. Карта Карно выражения $pqr \vee p\bar{q}r \vee pq\bar{r}$

Она имеет две пары минтермов для группировки (одна из них не видна при данном обозначении столбцов). Итак,

$pqr \vee p\bar{q}r = pq(r \vee \bar{r}) = pq$ и $pqr \vee p\bar{q}r = (q \vee \bar{q})pr = pr$.

Это сводит функцию к выражению $pq \vee pr$, которое благодаря свойству

дистрибутивности редуцируется к функции $p(q \vee r)$.

Более простая схема, реализующая функцию из примера 6.6, показана на рис. 6.7.

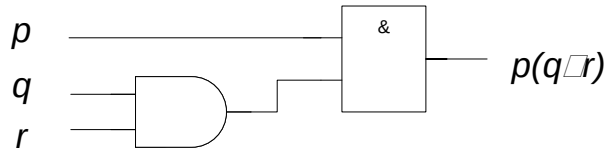


Рисунок 6.7. Эквивалентная схема из примера 6.6.

При разработке функциональных схем нет необходимости использовать все типы функциональных элементов. Как мы уже видели, множество $\{\vee, \bar{}\ \}$ является полной системой функций. Поэтому мы можем построить любую схему, ограничившись функциональными элементами **И** и **НЕ**.

Более того, если по той или иной причине нам неудобно использовать большое число компонент, мы могли бы использовать только функциональный элемент **НЕ-И**.

6.4. Проектирование сумматоров

Построим *полубитный сумматор,* предназначенный для сложения двух двоичных цифр. Ответ при этом представляется двузначным двоичным числом. Например, $1 +_2 1 = 10$.

Пусть *x* и *y* обозначают двоичные цифры, которые предстоит сложить, а *u* и *v* – двоичные цифры суммы, получающейся на выходе сумматора, как показано на рис. 6.8.

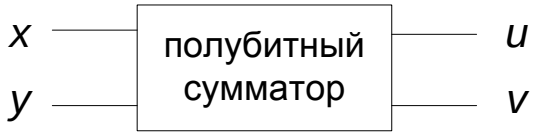


Рисунок 6.8. Модель полубитного сумматора

Таблица истинности (см. таблицу 6.8) проясняет связь между вводимыми и выводимыми цифрами. Следовательно, *u = xy (второй разряд, или разряд переноса)* и $v = \bar{x}y \vee x\bar{y}$ *(сложение по первому разряду)*.

Таблица 6.8. Таблица истинности для полубитного сумматора

x	***y***	***u***	***v***
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Полубитный сумматор реализуется следующей схемой (рис. 6.9):

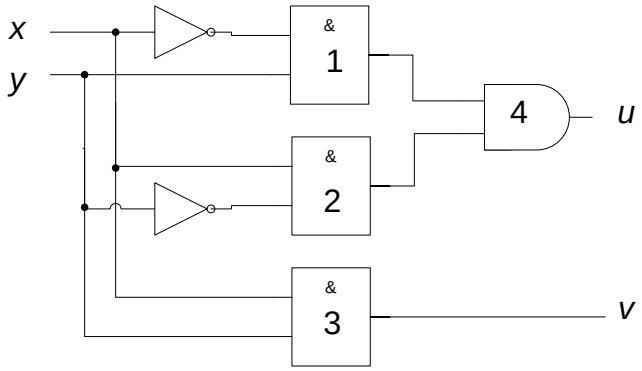


Рисунок 6.9. Схема полубитного сумматора

Выходными данными функциональных элементов 3 и 4 являются первый и второй разряды соответственно (табл. 6.9).

Таблица 6.9. Значения на входах и выходах вентилей полубитного сумматора

Логический элемент	**Ввод**	**Вывод**
1	$\bar{x}, y$	$\bar{x}y$
2	$x, \bar{y}$	$x\bar{y}$
3	x, y	xy
4	$\bar{x}y, x\bar{y}$	$\bar{x}y \vee x\bar{y}$

Сейчас спроектируем 2-битный сумматор. Это устройство, которое вычисляет сумму двузначных двоичных чисел, выдавая в качестве ответа трехзначное двоичное число. На входе 2-битный сумматор получает два двузначных двоичных числа, а на выходе у него оказывается трехзначное число, равное сумме вводимых чисел. Иными словами, 2-битный сумматор складывает числа в двоичной системе счисления, например: $11 +_2 10 = 101$.

Обозначим через a и b цифры второго и первого разрядов соответственно для первого вводимого в сумматор числа, а через c и d – цифры второго и первого разряда соответственно для второго числа (рис. 6.10). Пусть e, f и g – цифры третьего, второго и первого разрядов вычисляемой суммы.

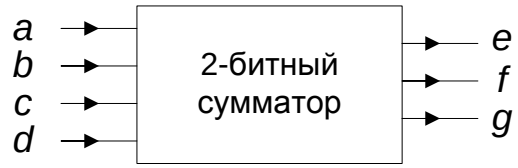


Рисунок 6.10. Модель двухбитного сумматора

Можно, как и в случае с полубитным сумматором, заполнить таблицы истинности цифр *e, f* и *g*, считая их булевыми функциями от вводимых цифр, упростить полученные выражения с помощью карты Карно и начертить функциональную схему. Однако можно поступить иначе: используем полубитный сумматор в качестве функционального элемента (вентиля). Схема, представленная на рис. 9.11, использует два полубитных сумматора для вычисления сумм: $a +_2 c$ и $b +_2 d$.

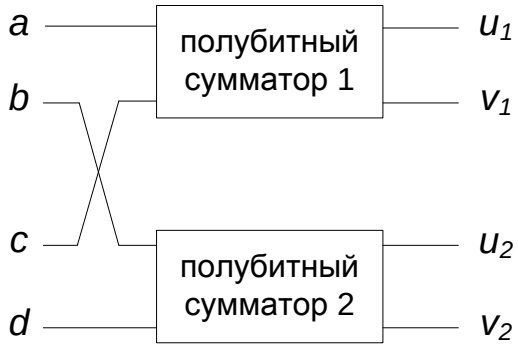


Рисунок 6.11. Схема поразрядного суммирования
двух двузначных чисел

Сумма цифр первого разряда (переменная v_2) дает нам цифру g. Складывая разряд переноса u_2 с v_1 с помощью третьего полубитного сумматора, мы получаем двузначное число с цифрами v_3, которое дает нам значение f, и u_3. Наконец, последняя цифра суммы – e – может быть получена из u_1 и u_3 с помощью функционального элемента ИЛИ.

Функциональная схема 2-битного сумматора представлена на рис. 6.12.

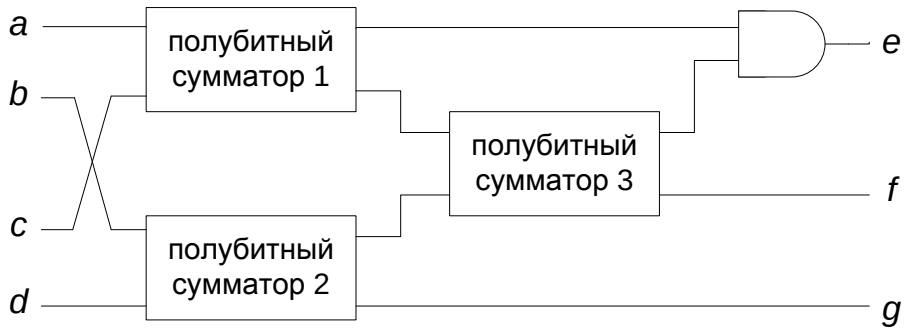


Рисунок 6.12. Функциональная схема двухбитного сумматора

На рис. 6.13 изображена функциональная схема 3-битного сумматора, складывающего два трехзначных двоичных числа с цифрами *a, b, c* и *d, e, f* соответственно. В качестве суммы получается четырехзначное число с цифрами *g, h, i, j*.

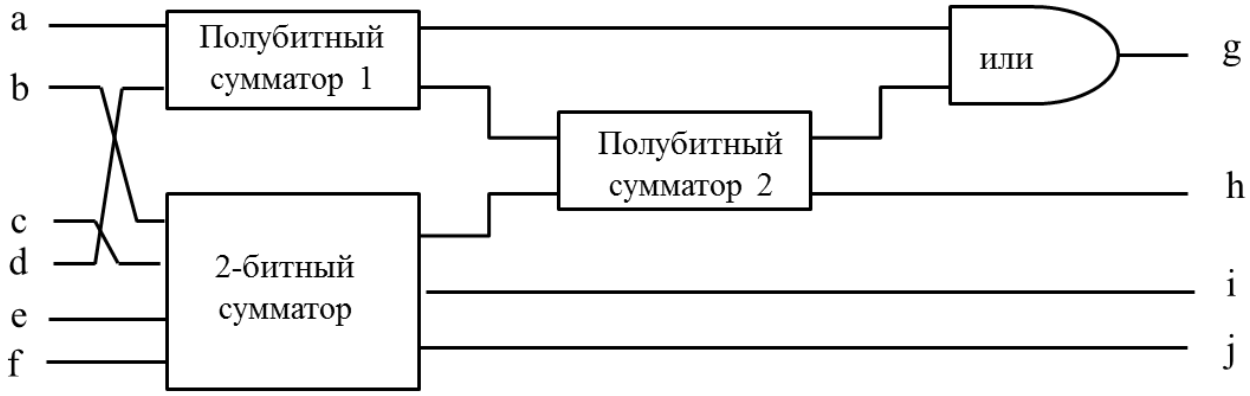


Рисунок 6.13. Функциональная схема трехбитного сумматора

7. Комбинаторика

Комбинаторика – это раздел математики, занимающийся подсчетом элементов конечных множеств. В этой главе будут рассмотрены задачи пересчета, которые решаются с помощью двух принципов – правил суммы и произведения.

Общие задачи пересчета связаны с выборкой некоторого числа элементов из заданного базисного множества. Такие задачи делятся на четыре типа в зависимости от того, как выбираются элементы: с повторением или без повторений, с учетом порядка выбора или без него. Будут выведены формулы для каждого из перечисленных типов задач.

Изучив методы пересчета, рассмотрим способы оценки эффективности алгоритмов. Это еще одно приложение функций к проблемам информатики, которое использует некоторые формулы комбинаторики.

7.1. Правила суммы и произведения

Рассмотрим решение нескольких простых задач.

Задача 1. В кондитерской к концу рабочего дня осталось несколько пирожных: четыре ванильных, два шоколадных и три фруктовых. Сколько пирожных может купить покупатель?

Эта задача решается простым подсчетом. Поскольку все пирожные различны, мы просто можем сложить их количества. Это дает 4+2+3 = 9 пирожных, из которых покупатель может сделать выбор.

Задача 2. Необходимо выбрать смешанную команду, которая будет представлять теннисный клуб на соревнованиях. В спортивном клубе состоят 6 женщин и 9 мужчин. Сколько различных пар можно выбрать для участия в соревнованиях?

В этой задаче у нас есть 6 женщин, из которых мы можем выбрать представительницу клуба, и для каждой из них мы можем подобрать партнера среди девяти мужчин. Таким образом, общее число различных пар, которые мы

можем составить, равно $6\times 9 = 54$.

Эти две задачи иллюстрируют два фундаментальных правила пересчета.

Правило суммы гласит, что если A и B – несвязанные события и существует n_1 возможных исходов события A и n_2 возможных исходов события B, то возможное число исходов события «A или B» равно сумме $n_1+ n_2$.

Правило произведения утверждает, что если дана последовательность k событий с n_1 возможными исходами первого, n_2 – второго и т.д., вплоть до n_k возможных исходов последнего, то общее число исходов последовательности k событий равно произведению $n_1\times n_2\times\ldots\times n_k$.

Правило суммы – это частный случай формулы включений и исключений. Действительно, если рассматривать A и B как множества исходов, то $|A|= n_1$, $|B|= n_2$; а поскольку события A и B не связаны друг с другом, то можно считать, что соответствующие множества не пересекаются (т. е., $|A\cap B|= \varnothing$). По формуле включений и исключений, $|A\cup B|= |A|+ |B|$, иными словами, множество $A\cup B$ содержит $n_1+ n_2$ элементов. Это означает, что существует $n_1+ n_2$ возможных исхода события «A или B».

Правило произведения тоже можно сформулировать на языке теории множеств. Пусть A_1 обозначает множество n_1 исходов первого события, A_2 – множество n_2 исходов второго, и т. д. Тогда любую последовательность k событий можно рассматривать как элемент декартова произведения $A_1\times A_2\times\ldots\times A_k$, чья мощность равна $|A_1|\times |A_2|\times\ldots\times |A_k|$.

Используя эти правила, решим третью задачу.

Задача 3. Сколько трехзначных чисел начинаются с 3 или 4?

Для решения этой задачи будем использовать и правило суммы, и правило произведения. Трехзначные числа, о которых идет речь в задаче, естественным образом разбиваются на два непересекающихся класса. К одному из них относятся числа, начинающиеся с 3, а ко второму – с 4. Для подсчета чисел в первом классе заметим, что существует один возможный исход для первой цифры (она должна быть равна 3), 10 исходов для второй и 10 исходов для последней цифры. По правилу произведения получаем, что всего чисел в первом

классе насчитывается ровно 1×10×10 = 100. Аналогично можно подсчитать количество чисел во втором классе. Оно также равно 100. Наконец, по правилу суммы получаем, что существует 100+100 = 200 трехзначных чисел, начинающихся с 3 или 4.

Пример 7.1. Есть три банана, четыре яблока и две груши. Надо выбрать два фрукта разного вида. Сколькими способами можно сделать такой выбор?

Решение. Если собираемся взять один из трех бананов и одно из четырех яблок, то сделать это можно 3×4 = 12 различными способами. Банан и грушу можно взять 3×2 = 6 возможными способами. Наконец, грушу и яблоко можно выбрать 4×2 = 8 различными способами. Поскольку все три множества возможностей различны, то всего количество способов, которыми можно выбрать два фрукта, равно 12+6+8=26.

Пример 7.2. Государственный регистрационный знак легкового автомобиля состоит из четырех цифр и двух букв белорусского алфавита, присутствующих и в латинском алфавите (всего 12 букв), а также 7 цифр, указывающих на территориальную принадлежность транспорта. В номере можно использовать любую последовательность букв и цифр. Сколько различных автомобильных номеров может выдать ГАИ?

Решение. Каждую из двух букв номера можно выбрать из 12 букв алфавита. По правилу произведения число различных последовательностей из двух букв равно 12×12= 144. Аналогично, число последовательностей четырех цифр равно 10×10×10×10 = 10000. Наконец, поскольку каждый из автомобильных номеров состоит из двух букв и четырех цифр, правило произведения дает 1 440 000 различных автомобильных номеров в одной территориальной единице. Всего в стране ГАИ может выдать 1 440 000×7 = 10 080 000 номеров.

7.2. Комбинаторные формулы

Допустим, что ребенку предложили мешок с конфетами трех наименований: «Крыжачок» (*A*), «Грильяж» (*B*) и «Коровка» (*C*). Сколькими способами ребенок

может выбрать две конфеты из мешка? На этот вопрос можно дать несколько ответов в зависимости от уточнения его формулировки. Ставя задачу, мы не уточнили, можно ли взять две конфеты одного наименования или нет. Например, можно ли взять две конфеты «Крыжачок», т.е., сделать выбор *AA*? Кроме того, имеет ли значение порядок выбора? Иными словами, отличается ли выбор *AB* от *BA* или нет?

Таким образом, мы имеем четыре разных типа пересчета:

1. Повторения разрешены, и порядок выбора существенен. В этом случае у нас есть 9 возможностей: AA, AB, AC, BA, BB, BC, CA, CB и CC.
2. Запрещено брать конфеты одного наименования, но порядок существенен. В этой ситуации есть 6 случаев: AB, AC, BA, BC,CA и CB.
3. Повторения разрешены, но порядок выбора не имеет значения. Тогда ответ – тоже 6 возможностей: AA, AB, AC, BB, BC и CC.
4. И, наконец, если нельзя брать одинаковые конфеты, а порядок не имеет значения, то у ребенка есть только три варианта выбора: AB, AC и BC.

При решении конкретных задач на подсчет количества способов необходимо четко понимать, о каком типе пересчета идет речь. Чтобы различать на терминологическом уровне тип конкретной задачи, введем несколько определений.

Предположим, что мы берем элементы $x_1, x_2, \ldots, x_k$ из множества X мощности n. Каждый такой набор принято называть *выборкой* объема k из n элементов, или (n, k)-выборкой. Выборка называется *упорядоченной,* если порядок следования элементов в ней задан. При этом две упорядоченные выборки, различающиеся лишь порядком следования элементов, считаются разными. Если же порядок следования элементов в выборке не имеет значения, то выборка называется *неупорядоченной*. Теперь введем основные термины в соответствии с типом уточнений, приведенных выше:

- (n,k)-*размещением с повторениями* называется упорядоченная (n,k)-выборка, элементы в которой могут повторяться;
- (n,k)-*размещением без повторений* называется упорядоченная

(n,k)-выборка, элементам в которой повторяться запрещено;

– (n,k)-*сочетанием с повторениями* называется неупорядоченная (n,k)-выборка с повторяющимися элементами;
– (n,k)-*сочетанием без повторений* называется неупорядоченная (n,k)-выборка без повторяющихся элементов.

Подсчитаем количество всех различных (n,k)-размещений с повторениями. На первое место выборки мы можем поставить любой из n элементов множества. Поскольку повторения разрешены, то на второе место мы опять можем поставить любой элемент из этого же множества и т.д. Поскольку у нас k мест в выборке, то по правилу произведения получаем, что **число всех (n,k)-размещений с повторениями $\overline{A}(n,k) = n^k$**. В литературе можно встретить и такое обозначение: $\overline{A}_n^k$.

Пример 7.3. Целые числа в компьютере представляются строчкой из N двоичных знаков. Первый из них отведен для знака (+ или –), а остальные $N – 1$ отвечают за модуль целого числа. Сколько различных целых чисел может использовать компьютер?

Решение. Двоичный знак – это 0 или 1. Для записи числа используется N таких цифр. Двоичные строки, представляющие числа, могут иметь повторяющиеся цифры и порядок их следования существенен для данной задачи. Поэтому мы имеем дело с (2, N)-размещениями с повторениями. По выведенной формуле получаем, что общее количество таких строк равно 2^N. Практически всегда различные размещения изображают различные числа, за исключением двух строк: –000000…00 и + 000000…00, которые изображают 0. Значит, компьютер может оперировать $(2^N – 1)$ целыми числами.

Для числа всех (n, k)-размещений без повторений используется специальное обозначение – $P(n, k)$ (в русскоязычной литературе часто используется обозначение A_n^k). Подсчитаем это число. На первое место выборки можем поставить любой из n элементов. Поскольку здесь нам не разрешены повторения,

то для второго места можем выбрать любой из (n – 1) оставшихся элементов. На третье – из (n – 2) и т. д., вплоть до k-го места, куда можно записать любой из *(n* – k + 1) элементов. Теперь для окончательного ответа нам нужно применить правило произведения. Имеем

$P(n, k) = n(n-1)(n-2)\ldots(n-k+1)$.

Выразим значение $P(n, k)$, используя факториал:

$$P(n,k) = n(n-1)(n-2)\ldots(n-k+1) =$$

$$= n(n-1)(n-2)\ldots(n-k+1) \times \frac{(n-k)(n-k-1)\ldots 2\cdot 1}{(n-k)(n-k-1)\ldots 2\cdot 1} =$$

$$= \frac{n(n-1)(n-2)\ldots(n-k+1)(n-k-1)\ldots 2\cdot 1}{(n-k)(n-k-1)\ldots 2\cdot 1} = \frac{n!}{(n-k)!}.$$

Итак, **число различных (n, k)-размещений без повторений равно**

$$\boldsymbol{P(n,k) = \frac{n!}{(n-k)!}}.$$

Пример 7.4. Сколько различных четырехбуквенных «слов» можно написать, используя буквы *a, c, n, o, p, e*, если под «словом» мы будем понимать любую последовательность неповторяющихся букв, даже если эта последовательность не несет в себе никакого смысла?

Решение. Как договорились, под «словом» мы понимаем любую последовательность четырех разных букв, которые можно выбрать из шести данных. Значит, мы имеем дело с подсчетом числа размещений без повторений $P(6, 4)$. Следовательно, $P(6, 4) = \frac{6!}{(6-4)!} = \frac{6!}{2!} = 360$.

Теперь займемся сочетаниями без повторений, т.е. выборками, в которых порядок не существенен и повторы запрещены. Число всех (n, k)-сочетаний без повторений обозначается символом $C(n, k)$ (это число также может обозначаться как C_n^k или $\binom{k}{n}$). Найдем его.

Мы воспользуемся уже известным нам фактом: число всех (n, k)-размещений без повторений равно $P(n, k) = n!/(n-k)!$. Поскольку размещение

без повторений отличается от сочетания без повторений наличием порядка, то число $P(n, k)$ естественно больше, чем $C(n, k)$. Если мы найдем соотношение между ними, то получим нужную формулу.

Проведем эксперимент. Пусть $n = 4$, а $k = 3$. Рассмотрим множество $A = \{1, 2, 3, 4\}$, из которого будем выбирать элементы. Каждое (4, 3)-сочетание без повторений – это выбор последовательности трех разных цифр из четырех данных, причем порядок, в котором будут идти выбранные цифры, значения не имеет. Например, подмножество {1, 2, 3} является (4, 3)-сочетанием без повторений. Перемешав цифры в выбранном подмножестве {2, 1, 3}, мы получим то же самое сочетание (порядок не важен), но совершенно другое размещение (порядок существенен). Узнаем, сколько различных размещений можно получить из одного сочетания? В данном конкретном случае ($n = 4$, $k = 3$) ответ легко получить, перечислив вручную все варианты. Нам же надо разобраться с общим случаем. Сформулируем его более четко: *дано* (n, k)-*сочетание без повторений, т.е. выбрано подмножество* $B \subset A$, *где* $|B| = k$ и $|A| = n$. *Сколько из него можно получить разных* (n, k)-*размещений без повторений?*

Фактически, нам нужно подсчитать количество (k, k)-размещений без повторений. Но это число мы знаем (учитываем, что 0! = 1):

$$P(k, k) = \frac{k!}{(k-k)!} = k!$$

Таким образом, на каждое **(n, k)-сочетание без повторений** приходится $k!$ различных (n, k)-размещений без повторений. Значит,

$$\boldsymbol{C(n, k) = \frac{P(n, k)}{k!} = \frac{n!}{(n-k)!\, k!}}.$$

Пример 7.5. Меню в ресторане дает возможность выбрать ровно три из семи главных блюд. Сколькими способами вы можете сделать заказ?

Решение. Здесь мы имеем дело с (7, 3)-сочетаниями без повторений. Поэтому ответ получить легко:

$$C(7,3) = \frac{7!}{(7-3)!\, 3!} = 35.$$

Итак, имеется 35 возможностей для различных заказов.

Последнее, что мы исследуем – это сочетания с повторениями. Напомним, что это выборки, в которых порядок не важен, а вот повторы элементов допускаются. Поскольку порядок в наших выборках значения не имеет, а повторы разрешены, мы можем сгруппировать вместе одинаковые элементы, разделив группы какими-нибудь метками.

Предположим, например, что мы сделали выборку, состоящую из пяти букв, каждая из которых может быть одной из *а, б* и *в*. Выборку, состоящую из двух «*а*», одной «*б*» и двух «*в*», можно записать как *аа|б|вв,* а выборка из одной буквы «*а*» и четырех букв «*в*» будет выглядеть так: *а||вввв.* Договоримся, что слева от первой метки либо стоят буквы «*а*», либо ничего, справа от второй метки – «*в*» либо ничего, а буквы «*б*», если они присутствуют в выборке, стоят между метками. Таким образом, можно считать, что мы всегда смотрим на *семь* ячеек (пять букв и две метки), причем различные выборки будут отличаться ячейками, в которых стоят метки. Значит, число всех таких сочетаний с повторениями совпадает с количеством способов, которыми мы можем поместить две метки в семь ячеек. Осталось понять, что это количество есть не что иное, как число всех (7, 2)-сочетаний без повторений, т.е. равно $C(7, 2)$. Действительно, первую метку можно поставить в любую из семи ячеек, а вторую – в любую из шести, поскольку одна ячейка уже занята. Это дает нам 7 × 6 возможностей. Заметим теперь, что, поменяв расставленные метки местами, мы получим то же самое заполнение ячеек. Значит, 7 × 6 нужно разделить на 2. Итак, количество способов равно

$$\frac{7 \cdot 6}{2} = \frac{7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1}{(2 \cdot 1)(5 \cdot 4 \cdot 3 \cdot 2 \cdot 1)} = \frac{7!}{5! \cdot 2!} = \frac{7!}{(7-2)! \cdot 2!} = C(7,2).$$

Возвращаясь к общему случаю (n, k)-сочетаний с повторениями (k объектов из n данных), заметим, что нам потребуется $n - 1$ метка и k объектов. Таким образом, у нас будет $(n - 1) + k$ ячеек для заполнения. Значит, число (n, k)-сочетаний с повторениями $F(n, k)$ совпадает с количеством способов расположения $(n - 1)$ метки в $(n + k - 1)$ ячейку. Итак, общее **число (n, k)-**

сочетаний с повторениями равно:

$$\boldsymbol{F(n,\,k) = C(n+k-1,\,n-1)} = \frac{(n+k-1)!}{\big(n+k-1-(n-1)\big)!(n-1)!} = \frac{\boldsymbol{(n+k-1)!}}{\boldsymbol{k!(n-1)!}}.$$

Пример 7.6. Сколько различных вариантов можно получить, бросая пять игральных костей?

Решение. На каждой из костей может выпасть от одного до шести очков, т. е. каждая кость дает шесть вариантов. Если бросили пять костей, то каждый вариант можно рассматривать как неупорядоченный набор пяти объектов (для каждого из которых есть 6 возможностей) с повторениями, т. е. (6, 5)-сочетание с повторениями. Согласно общей формуле, общее число исходов равно C (6 + 5 – 1, 6 – 1) = C (10, 5) = 10!/(5!·5!) = 252.

В табл. 7.1 собраны вместе все формулы подсчета количества выборок k элементов из n-элементного множества, которые мы вывели в этом параграфе.

Таблица 7.1. Количество выборок k-элементов из n-элементного множества

Порядок существенен		**Порядок не существенен**	
Размещения с повторениями	n^k	Сочетания с повторениями	$\dfrac{(n+k-1)!}{k!\,(n-1)!}$
Размещения без повторений	$\dfrac{n!}{(n-k)!}$	Сочетания без повторений	$\dfrac{n!}{(n-k)!\,k!}$

Разберем еще несколько примеров, которые наглядно показывают, что нужно быть очень внимательным при выборе комбинаторной формулы для решения конкретной задачи.

Пример 7.7. В лотерее «Спортлото» случайным образом выбирается шесть разных номеров из первых 49 натуральных чисел. Любой, кто угадает все шесть выпавших номеров, выиграет главный приз – крупное денежное вознаграждение. Все участники, угадавшие 3 номера, получают минимальное вознаграждение.

Подсчитаем шансы выигрыша главного приза. Шестерка выигрышных номеров – это неупорядоченная выборка шести чисел из 49 возможных. Поскольку общее количество (49, 6)-сочетаний без повторений равно

49!/((49! – 6)!·6!) = 49!/(43! 6!) = 13 983 816, то шансов выиграть практически нет: 1 из 13 983 816. Тем, кто угадал пять, четыре или три номера, тоже

присуждается приз, но значительно меньший.

Подсчитаем вероятность выигрыша минимального приза.

Итак, вы можете рассматривать свои шесть номеров как объединение двух несвязанных событий: *выборка трех правильных номеров* и в*ыборка трех неверных номеров*. Существует $C(6, 3)$ возможностей назвать три из шести выигрышных номеров и $C(43, 3)$ возможности неудачного выбора. Тогда общее число комбинаций, выигрывающих три номера, –

$$C(6,3) \times C(43,3) = \frac{6!}{(3! \cdot 3!)} \times \frac{43!}{(40! \cdot 3!)} = 246820.$$

Вероятность выигрыша – это доля удачно заполненных карточек ко всем возможным заполнениям, т.е. $246\,820/13\,983\,816 \approx 1/7 \approx 0{,}018$.

Пример 7.8. Двенадцать человек, включая Марию и Петра, являются кандидатами в комитет пяти. Сколько разных комитетов можно набрать из 12 кандидатов? Сколько из них:

(а) включают как Марию, так и Петра;

(б) не включают ни Марию, ни Петра;

(в) включают либо Марию, либо Петра, но не обоих?

Решение. Существует $C(12, 5)=12!/(7!\times5!) = 792$ возможных комитета.

(а) Если Мария и Петр уже выбраны в комитет, нам остается отобрать в него только трех членов из оставшихся десяти кандидатов. Это можно сделать $C(10, 3) = 10!/(7!\times3!) = 120$ способами. Значит, Мария и Петр могут быть членами 120 разных комитетов.

(б) Если Мария и Петр не участвуют в комитете, то мы выбираем всех его членов из десяти кандидатов. Поэтому у нас есть $C(10,5)= 10!/(5!\times5!) = 252$ возможности для разных комитетов, не включающих ни Марию, ни Петра.

(в) Один из способов дать ответ на третий вопрос заключается в подсчете комитетов, включающих Марию, но без Петра. Их ровно $C(10, 4)$. То же число комитетов включают Петра, но без Марии. Значит, $2\times C(10, 4)$ комитета имеют в качестве члена либо Марию, либо Петра, но не обоих сразу.

Альтернативный подход к решению основан на том, что каждый из 792

возможных составов комитета можно отнести в точности к одной из категорий: (а), (б) или (в). Значит, число комитетов, относящихся к последней, равно 792 – 120 – 252 = 420.

7.3. Бином Ньютона

Числа $C(n, k)$ возникают как коэффициенты при раскрытии скобок в биноме $(a + b)^n$. Например, $(a + b)^3 = (a + b)(a + b)(a + b) =$

$= aaa + aab + aba + abb + baa + bab + bba + bbb = a^3 + 3a^2b + 3ab^2 + b^3$.

Каждое из восьми слагаемых, стоящих во второй строке наших преобразований, получается при умножении трех переменных, выбираемых по одной из каждой скобки. Мы видим, в частности, что ровно три слагаемых содержат одну переменную a и две b. Это происходит потому, что у нас есть $C(3, 2) = 3$ способа выбора двух скобок из трех, откуда мы возьмем переменную b (а из оставшейся берем a).

Аналогично получаются и остальные коэффициенты этого выражения: $C(3, 0) = 1$, $C(3, 1) = 3$, $C(3, 2) = 3$ и $C(3, 3) = 1$. Чтобы согласовать полученные числа с формулой для $C(n, k)$, выведенной в предыдущем параграфе, принято считать, что $0! = 1$. Иначе говоря, существует единственная возможность не сделать никакого выбора из конечного множества объектов, а также единственная возможность выбора k объектов из k элементов.

В общем случае, раскрывая скобки в биноме $(a + b)^n$, при перемножении b, взятых из k скобок, и a, взятых из оставшихся $(n – k)$ скобок мы будем получать члены вида $a^{n-k}b^k$, где k принимает все значения от 0 до n). Так как есть ровно $C(n, k)$ способов выбора k скобок из n, то у нас будет в точности $C(n, k)$ членов вида $a^{n-k}b^k$ при $k = 0, 1, ..., n$. Следовательно,

$$(a+b)^n = C(n, 0)a^n + + C(n, 1)a^{n-1}b + C(n, 2)a^{n-2}b^2 + \ldots + C(n, n)b^n.$$

Такое же количество способов выбора скобок, из которых будем брать a:

$$C(n, n–k) = \frac{n!}{(n-k)!\times\big(n-(n-k)\big)!} = C(n, k).$$

Эта формула называется *биномом Ньютона*. Именно поэтому

коэффициенты *C(n, k)* часто называют *биномиальными коэффициентами.* Биномиальные коэффициенты полезно выстроить в так называемый *треугольник Паскаля* (см. рис. 7.1 и 7.2).

Каждая (*n* + 1)-я строка этого треугольника состоит из биномиальных коэффициентов, получающихся при раскрытии скобок в выражении $(a + b)^n$.

$$
\begin{array}{ccccccccccc}
 & & & & & C(0,0) & & & & & \\
 & & & & C(1,0) & & C(1,1) & & & & \\
 & & & C(2,0) & & C(2,1) & & C(2,2) & & & \\
 & & C(3,0) & & C(3,1) & & C(3,2) & & C(3,3) & & \\
 & C(4,0) & & C(4,1) & & C(4,2) & & C(4,3) & & C(4,4) & \\
C(5,0) & & C(5,1) & & C(5,2) & & C(5,3) & & C(5,4) & & C(5,5) \\
 & & & & & \ldots\ldots & & & & & \\
C(n,0) & & C(n,1) & & & \ldots\ldots & & & C(n,n-1) & & C(n,n)
\end{array}
$$

Рисунок 7.1. Треугольник Паскаля

Вычислив несколько первых коэффициентов треугольника Паскаля, мы получим:

$$
\begin{array}{ccccccccccc}
 & & & & & 1 & & & & & \\
 & & & & 1 & & 1 & & & & \\
 & & & 1 & & 2 & & 1 & & & \\
 & & 1 & & 3 & & 3 & & 1 & & \\
 & 1 & & 4 & & 6 & & 4 & & 1 & \\
1 & & 5 & & 10 & & 10 & & 5 & & 1 \\
 & & & & & \ldots & & & & &
\end{array}
$$

Рисунок 7.2. Численные значения треугольника Паскаля

Так как $C(n, 0) = C(n, n) = 1$, то на внешних сторонах треугольника Паскаля всегда стоят единицы. Симметрия относительно вертикальной высоты треугольника следует из тождества $C(n, k) = C(n, n - k)$.

Есть и другие закономерности, которые легко заметить, изучая треугольник Паскаля. Например, сложив два последовательных числа, стоящих в строке треугольника, мы получим число из следующей строки, которое стоит между двумя сложенными. Это свойство известно как *формула Паскаля:*

$C(n - 1, k - 1) + C(n - 1, k) = C(n, k)$, справедливая при $0 < k < n$.

Доказательство формулы Паскаля состоит в последовательности преобразований:

$$C(n-1,k-1)+C(n-1,k)=\frac{(n-1)!}{(n-k)!\,(k-1)!}+\frac{(n-1)!}{(n-k-1)!\,k!}=$$

$$=\frac{(n-1)!}{(n-k-1)!\,(k-1)!}\left(\frac{1}{n-k}+\frac{1}{k}\right)=\frac{(n-1)!}{(n-k-1)!\,(k-1)!}=$$

$$=\frac{n!}{(n-k)!\,k!}=C(n,k).$$

Другая формула, которую мы применим в следующем доказательстве, также видна из треугольника Паскаля и доказывается аналогичным методом:

$$C(n, n)+C(n+1, n)+C(n+2, n)+\ldots+C(n+r, n) = C(n+r+1, n+1).$$

Закончим знакомство с комбинаторикой задачей о количестве перестановок. Например, сколько новых «слов» можно составить, переставляя буквы в слове «КОЛОБОК». Это слово состоит из семи букв, которые можно переставить 7! способами, т.к. перемещения эквивалентны размещению без повторений $P(7,7)$. Однако в нем есть три буквы «О» и две буквы «К». Поэтому, меняя между собой местами буквы «О» или буквы «К», мы не получим новых «слов». Так как количество перестановок трех элементов равно 3!, а двух – 2!, то мы можем получить всего $\frac{7!}{3!\times 2!}! = 420$ разных «слов» из слова «КОЛОБОК».

В общем виде справедлива следующая теорема о перестановках.

Теорема. Существует $n!/(n_1!, n_2!... n_r!)$ различных перестановок n объектов, n_1 из которых относятся к типу 1, n_2 – к типу 2 и т.д. вплоть до n_r объектов типа r.

Пример 7.9. Сколькими способами можно распределить 15 студентов по трем учебным группам по пять студентов в каждой?

Решение. У нас есть 15 объектов, которые нужно организовать в три группы по пять. Это можно сделать 15!/(5!5!5!) = 68 796 различными способами.

Коэффициенты $n!/(n_1!, n_2!... n_r!)$ носят название *мультиномиальных*. Они стоят при произведениях $x_1^{n_1},\ x_2^{n_2}\ldots x_r^{n_r}$ в разложении степени $(x_1+x_2+\cdots+x_r)^n$. В этом легко убедиться, поскольку член вида $x_1^{n_1},\ x_2^{n_2}\ldots x_r^{n_r}$ поручается, когда мы перемножаем переменные x_1, выбранные из n_1 скобок, x_2, выбранные из n_2 скобок, и т. д. Таким образом, коэффициент при $x_1^{n_1},\ x_2^{n_2}\ldots x_r^{n_r}$ в точности

равен числу перестановок n объектов, n_1 из которых относятся к первому типу, n_2 – ко второму и т. д.

Пример 7.10. Найдите:

(а) коэффициент при $x^3y^2z^4$ из разложения степени $(x + y + z)^9$;

(б) коэффициент при x^3y^2 из разложения степени $(x + y + 3)^7$.

Решение.

(а) Коэффициент при $x^3y^2z^4$ из разложения степени $(x + y + z)^9$ равен $9!/(3!2!4!) = 1260$.

(б) Коэффициент при $x^3y^2z^{(7-3-2)} = x^3y^2z^2$ из разложения степени $(x + y + z)^7$ равен $7!/(3!2!2!) = 210$.

Поэтому разложение степени $(x + y + z)^7$ содержит член $210x^3y^2z^2$. Положив $z = 3$, мы увидим, что в разложении степени $(x + y + 3)^7$ присутствует член $1890x^3y^2$. Итак, коэффициент при x^3y^2 из разложения степени $(x + y + 3)^7$ равен 1890.

7.4. Метод рекуррентных соотношений

Метод рекуррентных соотношений состоит в том, что решение комбинаторной задачи с n предметами выражается через решение аналогичной задачи с меньшим числом предметов с помощью некоторого соотношения, которое называется рекуррентным, или возвратным (от латинского *recurrere* – «возвращаться»). Пользуясь этим соотношением, искомую величину можно вычислить исходя из того, что для небольшого количества предметов (одного, двух) решение задачи легко находится.

Понятие рекуррентных соотношений проиллюстрируем классической проблемой, которая была поставлена около 1202 г. Леонардо Фибоначчи.

Пара кроликов приносит раз в месяц приплод из двух крольчат (самки и самца), причем новорожденные крольчата через два месяца после рождения уже приносят приплод. Сколько кроликов появится через год, если в начале года была одна пара кроликов?

Из условия задачи следует, что через месяц будет две пары кроликов. Через

два месяца приплод даст только первая пара кроликов и получится 3 пары. А еще через месяц приплод дадут и исходная пара кроликов, и пара кроликов, появившаяся два месяца тому назад. Поэтому всего будет 5 пар кроликов. Обозначим через $F(n)$ количество пар кроликов по истечении n месяцев с начала года. Ясно, что через $n+1$ месяцев будут эти $F(n)$ пар и еще столько новорожденных пар кроликов, сколько было в конце месяца $n–1$, т.е., еще $F(n–1)$ пар кроликов. Иными словами, имеет место рекуррентное соотношение $F(n+1) = F(n) + F(n–1)$.

Так как по условию $F(0) = 1$ и $F(1) = 2$, то последовательно находим

$F(2) = 3$, $F(3) = 5$ и т. д.

Полученные таким способом числа называются числами Фибоначчи.

Проиллюстрируем применение метода рекуррентных соотношений на конкретных примерах.

Пример 7.11. Найти число сочетаний из n по k с повторениями, т.е., число $F(n, k)$.

Решение. Рассмотрим множество $A = \{a_1, a_2, \ldots, a_n\}$. Пусть B – множество всех сочетаний из n по k с повторениями, тогда мощность $|B| = F(n, k)$. Представим B как объединение множества B_1 и множества B_2. B_1 – это множество сочетаний с повторениями, которые не содержат элемента a_1, тогда $|B_1| = F(n – 1, k)$. B_2 – множество сочетаний с повторениями, содержащих хотя бы один раз элемент a_1. Так как каждое такое сочетание может быть получено присоединением к a_1 некоторого сочетания из n по $k – 1$, то $|B_2| = F(n, k – 1)$. Очевидно, что пересечение множеств B_1 и B_2 является пустым множеством, поэтому $|B| = |B_1| + |B_2|$, т. е.,

$$F(n, k) = F(n, k – 1) + F(n – 1, k). \qquad (7.1)$$

Получили рекуррентное соотношение, используя которое можно найти $F(n, k)$. Последовательно, применяя (7.1), находим

$$F(n, k) = F(n, k – 1) + (F(n – 1, k – 1) + F(n – 2, k)) =$$

$$= F(n, k – 1) + F(n – 1 ,k– 1) +\ldots+ F(2, k – 1) + F(1, k – 1). \qquad (7.2)$$

Очевидно, что

$$F(n, 1) = n \text{ и } F(1, k) = 1. \qquad (7.3)$$

Тогда, полагая в (7.2) $k = 2$, получим:

$$F(n, 2) = n + (n - 1) + \ldots + 2 + 1 = n\,(n + 1) / 2 = C\,(n + 1, 2). \qquad (7.4)$$

При $k = 3$ из равенства (7.2), учитывая формулы (7.3), (7.4) и свойство треугольника Паскаля (см. рис. 7.2 в §7.3)

$$C(n, n) + C(n+1, n) + C(n+2, n) + \ldots + C(n+k, n) = C(n+k+1, n+1),$$

имеем

$$F(n, 3) = C(n+1, 2) + C(n, 2) + \ldots + C(3, 2) + C(2, 2) = C(n + 2, 3).$$

Полученные результаты позволяют выдвинуть гипотезу:

$$\forall k \in N\colon F(n, k) = C(n + k - 1, k)\ . \qquad (7.5)$$

Проверяем истинность гипотезы методом математической индукции. При $k = 1$ предикат (7.5) принимает значение истины (см. (7.3)). Теперь, предположив истинность предиката (7.5) при произвольном $k > 1$, находим $F(n, k+1) = F(n, k) + F(n - 1, k) + \ldots + F(2, k) + F\,(1, k) =$

$$= C(n + k - 1, k) + C(n + k - 2, k) + \ldots + C(k + 1, k) + C(k, k) = C(n + k, k + 1),$$

т.е., предикат (7.5) принимает значение истины и если вместо k подставить $(k+1)$. Следовательно, согласно принципу математической индукции, формула (7.5) верна для любого натурального k.

В заключении отметим, что эта формула тождественна формуле для $F(n, k)$, полученной в §7.2, т.к. $C(n, k) = C(n, n-k)$.

Пример 7.12. Имеются марки достоинством 4, 6 и 10 копеек. Нужно наклеить марки на сумму 18 копеек. Сколько существует вариантов наклейки?

Решение. Будем обозначать через $f(n)$ количество вариантов наклейки марок на сумму n копеек. Рекуррентный метод будет применен для того, чтобы понизить значение n до величины, когда $f(n)$ подсчитать легко. Если последней наклеить марку номиналом 4 копейки, то общее количество вариантов выбора марок $f(18)$ совпадает с количеством вариантов выбора $f(14)$. Если последней наклеить марку номиналом 6 копеек, то $f(18) = f(12)$, и если последней наклеенной будет марка номиналом 10 копеек, то $f(18) = f(8)$. Поскольку все три варианта не пересекаются, то общее количество вариантов наклейки

$f(18) = f(14)+f(12)+f(8)$. Легко подсчитать, что $f(0) = 1$ – это единственный способ, когда не используется ни одна марка. Далее, $f(1) = f(2) = f(3) = 0$; $f(4) = 1$; $f(5) = 0$; $f(6) = 1$; $f(7) = 0$; $f(8) = 1$; $f(9) = 0$; $f(10) = 3$.

Аналогичными действиями можно записать $f(14)$ и $f(12)$ через меньшие значения n:

$f(14) = f(10)+f(8)+f(4)$ и $f(12) = f(8)+f(6)+f(2)$.

Тогда

$$f(18) = f(14)+f(12)+f(8) = f(10)+f(8)+f(4)+ f(8)+f(6)+f(2)+f(8) =$$
$$= f(10)+3f(8)+f(4)+f(6)+f(2) = 3+3+1+1+0 = 8.$$

Итак, существует 8 вариантов наклейки марок.

7.5. Эффективность алгоритмов

Одна из центральных задач информатики – создание и анализ «эффективности» компьютерных алгоритмов. Для успешного выполнения такого анализа нам необходимо уметь измерять затраты алгоритма в терминах времени и компьютерной памяти. Для этого, в частности, мы оцениваем время, необходимое для вычисления значения числовой функции. Один из способов оценки заключается в подсчете элементарных операций, которые производятся при вычислениях.

Например, чтобы установить, есть ли данное слово X в словаре, содержащем n слов, мы могли бы применить *последовательный поиск.* Названный алгоритм сравнивает слово X с первым словом в словаре, затем со вторым и т. д., пока не найдем слово X в словаре или, в наихудшем случае, оно не будет найдено вообще. Очевидно, в наихудшем случае будет произведено n сравнений. С другой стороны, при *двоичном (дихотомическом) способе* слово X сравнивается со «средним» из словаря, а потом, учитывая лексикографическое упорядочение слов, принимается решение о том, в какой части словаря (до «среднего» слова или после него) продолжать поиск. Этот процесс повторяется в выбранной половине словаря и т. д. В наихудшем случае (слово отсутствует в словаре)

двоичный поиск сделает $1 + \log_2 n$ сравнений. Как видно из табл. 6.2, двоичный поиск куда более эффективен, чем последовательный.

Таблица 7.2. Сравнение эффективности последовательного и двоичного поисков

n	$1 + \log_2 n$
8	4
64	7
$2^{18} = 250{,}000$	19

Пример 7.13. На выполнение алгоритмов *A, B, C, D* и *E* требуется n, $3n^2$, $2n^2+4n$, n^3 и 2^n элементарных операций соответственно. Подсчитайте время, необходимое для работы алгоритмов при n = 1, 10, 100 и 1000, если одна элементарная операция совершается за 1 мс.

Решение.

Таблица 7.3. Сравнение эффективности различных алгоритмов поиска

	A	***B***	***C***	***D***	***E***
	n	$3n^2$	$2n^2+4n$	n^3	2^n
1	1 мс	3 мс	6 мс	1 мс	2 мс
10	10 мс	300 мс	240 мс	1 с	1 с
100	100 мс	30 с	20 с	0,28 ч	4×10^{17} века
1000	1с	0,83 ч	0,56 ч	11,6 дн	10^{176} веков

Как видно из табл. 7.3, существует огромная качественная разница между формулами, включающими в себя степени числа n (полиномиальные функции), и теми, в которых n выступает в качестве показателя степени (экспоненциальные функции). Полиномиальные функции отличаются друг от друга величиной старшей степени переменной n. Если функции имеют одну и ту же старшую степень n, то рабочие периоды соответствующих алгоритмов близки друг к другу (см. алгоритмы *B* и *C)*.

Предположим, что функции $f(n)$ и $g(n)$ измеряют эффективность двух алгоритмов, их обычно называют *функциями временной сложности.* Будем говорить, что *порядок роста* функции $f(n)$ не больше, чем у $g(n)$, если найдется такая положительная константа C, что $|f(n)| \le C|g(n)|$ для всех достаточно больших значений n. Этот факт обозначают как $f(n) = O(g(n))$.

Пример 7.14. Покажите, что $2n^2 + 4n = O(n^2)$.

Решение. Так как $n \le n^2$ при $n \ge 1$, мы получаем, что $2n^2 + 4n \le 2n^2 + 4n^2 = 6n^2$ для всех $n \in \boldsymbol{N}$. Следовательно, положив $C = 6$ в определении порядка роста, мы можем заключить, что $2n^2 + 4n = O(n^2)$.

Кроме того, легко заметить, что $n^2 \le 2n^2 + 4n$ при $n \ge 1$. Другими словами, $n^2 = O(2n^2 + 4n)$. Две функции, такие как n^2 и $2n^2 + 4n$, каждая из которых имеет порядок роста не больше чем у другой, называются *функциями одного порядка роста*. Функции, ассоциированные с алгоритмами *B* и *C* в примере 7.13, имеют один и тот же порядок роста и, как следствие, соответствующие длительности работы алгоритмов близки.

Мы можем определить некоторую иерархическую структуру на множестве функций, каждая из которых имеет больший порядок роста, чем предыдущая. Один из примеров такой иерархии имеет вид:

Константа	Логарифм	Полином	Экспонента
1	$\log n$	$n\ n^2\ n^3 \ldots n^k \ldots$	2^n

Впоследствии при необходимости мы могли бы детализировать эту иерархию, вставив $(n \log n)$ между n и n^2, $(n^2 \log n)$ между n^2 и n^3 и т. д.

В этой иерархии важно то, что, двигаясь от ее левого края к правому, мы встречаем функции все большего порядка роста. Следовательно, чем правее в этом ряду стоит функция, тем быстрее растут ее значения по сравнению с ростом аргумента n. Сравнительные графики некоторых из перечисленных функций приведены на рис. 7.3.

Теперь любой функции временной сложности $f(n)$ мы можем сопоставить некоторую функцию $g(n)$ из описанной иерархии таким образом, что $f(n)$ будет иметь порядок роста не более чем $g(n)$, но больше, чем любая из функций иерархии, стоящая левее $g(n)$. Чтобы сделать это, мы с помощью нашей иерархии выделяем в данной функции наиболее быстро растущий член (его еще называют старшим членом) и сопоставляем нашей функции временной сложности соответствующую функцию в иерархии.

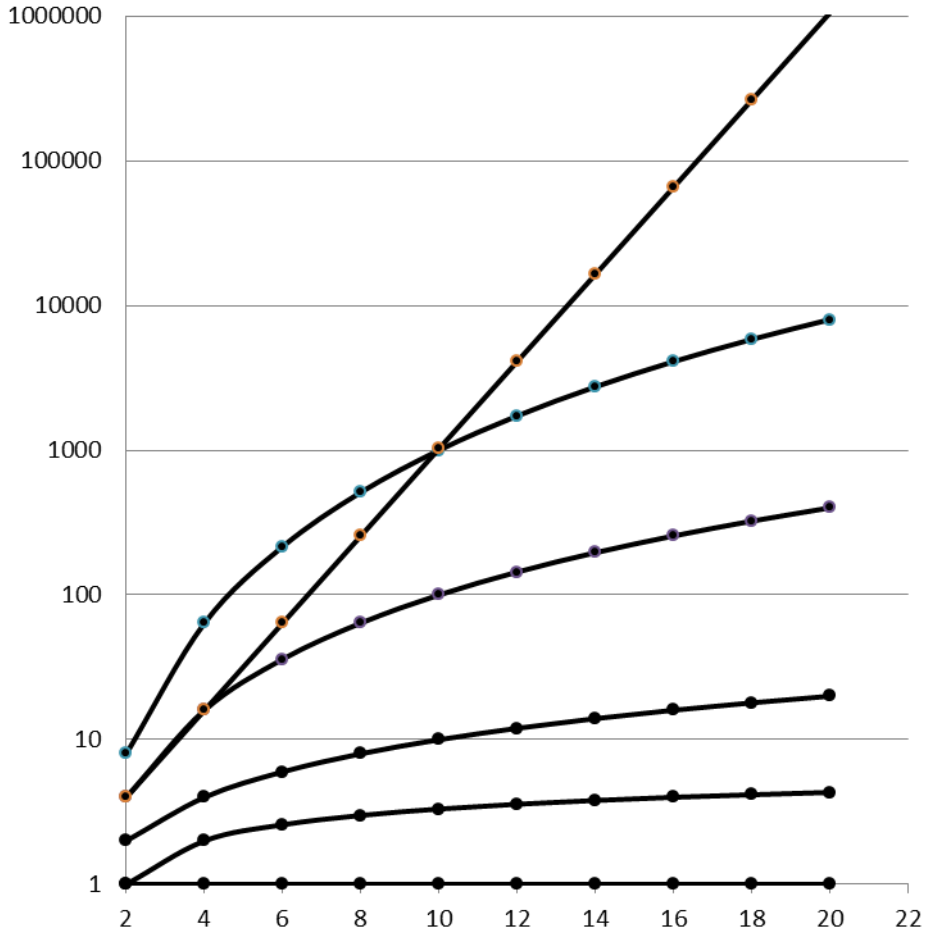


Рисунок 7.3. Относительный порядок роста функций.

Рассмотрим, например, функцию $f(n) = 9n+3n^6+7log_2n$. Ясно, что мультипликативные константы (числовые коэффициенты, на которые умножается то или иное выражение) не влияют на порядок роста функций. Поэтому $9n = O(n)$, $3n^6 = O(n^6)$ и $7log_2n = O(log_2n)$. Поскольку n и log_2n появляются в иерархии раньше, чем n^6, мы получаем, что как $9n$, так и $7log_2n$ принадлежат классу $O(n^6)$. Следовательно, $f(n) = O(n^6)$, и наиболее быстро растущим членом у $f(n)$ является член $3n^6$. Таким образом, значения функции $f(n)$ растут не быстрее, чем возрастают значения n^6. Фактически, в этом примере функция $f(n)$ имеет тот же порядок роста, что и n^6.

Пример 7.15. Используя иерархию функций, о которой шла речь выше, определите порядок роста у следующих функций временной сложности:

(а) n^4+2n^3+3;

(б) $6n^5+2^n$;

(в) $5n+n^2 \lg n$.

Решение.

(а) Функция принадлежит классу $O(n^4)$, поскольку n^4 – старший ее член.

(б) Функция принадлежит классу $O(2^n)$, поскольку 2^n – старший член.

(в) Старший член последней функции – это $n^2 \lg n$. Такой функции нет в иерархии, а если бы она была, то стояла бы между n^2 и n^3. Следовательно, можно утверждать, что данная функция растет не быстрее, чем функции, лежащие в классе $O(n^3)$.

При вычислении функции временной сложности любого алгоритма необходимо решить, что в данной задаче следует взять в качестве параметра n и какие элементарные операции стоит учитывать при расчетах.

Пример 7.16. Найдите функцию временной сложности следующего фрагмента алгоритма, написанного на псевдокоде, подсчитав количество операторов присваивания $x:= x+1$, которые в нем выполняются.

```
begin
  for i:= 1 to 2n do
        for j:= l to n do
            for k:= 1 to j do
                    x:=x + 1;
end
```

Внешний цикл (параметризованный переменной i) повторяется $2n$ раз. Цикл, помеченный переменной j, повторяется n раз. При каждом значении j операция $x:= x+1$ выполняется j раз. Следовательно, при каждом значении параметра внешнего цикла i операция $x:= x+1$ выполняется $1 + 2 + \ldots + n$ раз, что равно $0{,}5n(n+1)$. Значит функция временной сложности $T(n)$ определяется формулой:

$T(n) = 2n \times 0{,}5n(n+1) = n^2(n+1)$.

Итак, $T(n) = O(n^3)$.

8. Графы

В восемнадцатом столетии известный математик Леонард Эйлер впервые предложил использовать графы для решения ставшей впоследствии классической задачи о Кенигсбергских мостах. В Кенигсберге было два острова, соединенных семью мостами с берегами реки Преголь и друг с другом так, как показано на рис. 8.1. Задача состоит в следующем: осуществить прогулку по городу таким образом, чтобы, пройдя ровно по одному разу по каждому мосту, вернуться в то же место, откуда начиналась прогулка. Решая эту задачу, Эйлер изобразил Кенигсберг в виде графа, отождествив его вершины с частями города, а ребра – с мостами, которыми связаны эти части, и доказал, что искомого маршрута обхода города не существует.

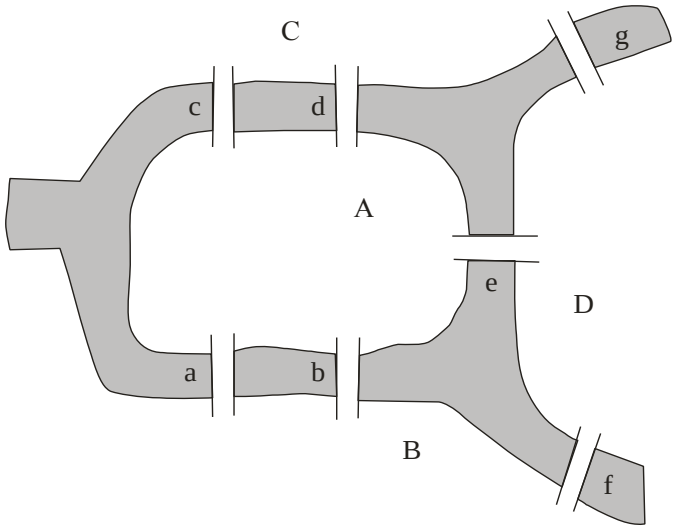


Рисунок 8.1. Схема мостов Кенигсберга

В этой лекции мы введем стандартную терминологию, используемую в теории графов, и рассмотрим, как с помощью графов решаются конкретные задачи. В частности, мы познакомимся с классом графов, называемым деревьями. Деревья – естественная модель, представляющая данные, организованные в иерархическую систему. Поиск по дереву для выделения отдельных предметов и сортировка данных в дереве представляют собой большой интерес для информатики.

8.1. Эйлеровы графы

На рис. 7.1 изображены семь мостов Кенигсберга так, как они были расположены в XVIII веке. В задаче, которую сформулировал Эйлер, спрашивается: можно ли найти маршрут прогулки, который проходит ровно один раз по каждому из мостов и начинается и заканчивается в одном и том же месте города? Модель задачи – это *граф,* состоящий из множества *вершин* и множества *ребер,* соединяющих вершины. Вершины *A, B, C* и *D* символизируют берега реки и острова, а ребра *a, b, c, d*, e, *f* и *g* обозначают семь мостов (рис. 8.2). Искомый маршрут (если он существует) соответствует обходу ребер графа таким образом, что каждое из них проходится только один раз. Проход ребра в этой модели соответствует пересечению реки по мосту.

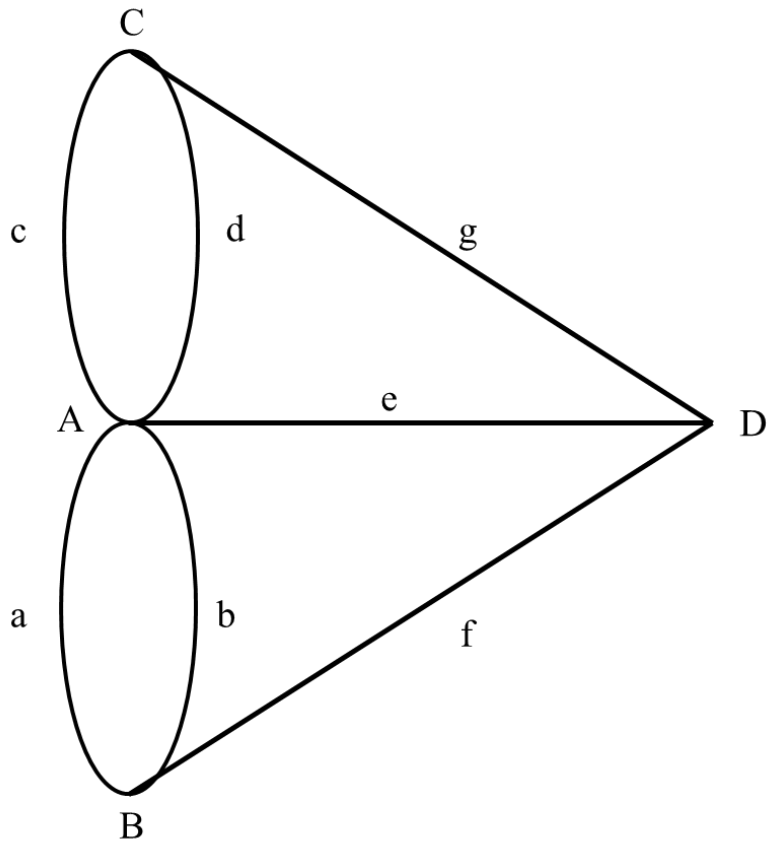


Рисунок 8.2. Графическая модель задачи о мостах Кенингсберга.

Граф, в котором найдется маршрут, начинающийся и заканчивающийся в одной вершине, и проходящий по всем ребрам графа ровно один раз, называется *эйлеровым графом.* Последовательность вершин (может быть и с повторениями),

через которые проходит искомый маршрут, как и сам маршрут, называется *эйлеровым циклом.* Легко заметить, что если в графе есть эйлеров цикл, то для каждого ребра, ведущего в какую-то вершину, должно найтись другое ребро, выходящее из этой вершины, и из этого простого наблюдения был сделан вывод: если в данном графе существует эйлеров цикл, то к каждой вершине должно подходить четное число ребер.

Кроме того, Эйлеру удалось доказать и противоположное утверждение. Граф, в котором любая пара вершин связана некоторой последовательностью ребер, является Эйлеровым тогда и только тогда, когда все его вершины имеют четную степень. *Степенью* вершины v называется число $\delta(v)$ ребер, ей *инцидентных* (если вершина v является концом ребра x, то говорят, что v и x инцидентны).

Очевидно, что в графе, моделирующем задачу о мостах Кенигсберга, эйлерова цикла найти нельзя. Действительно степени всех его вершин нечетны: $\delta(B) = \delta(C = \delta(D) = 3$ и $\delta(A) = 5$.

Графы, подобные тому, который мы исследовали при решении задачи о мостах, стали использоваться при решении многих практических задач, а их изучение привело к развитию большого раздела математики.

Простой граф определяется как пара $G = (V, E)$, где V – конечное множество вершин (*vertex* – угол, вершина угла), а E – конечное множество ребер (*edge* – край, грань), причем простой граф не может содержать *петель* (ребер, начинающихся и заканчивающихся в одной вершине) и *кратных ребер* (кратными называются несколько ребер, соединяющих одну и ту же пару вершин). Граф, изображенный на рис. 8.2, не является простым, поскольку, например, вершины A и B соединяются двумя ребрами.

Две вершины v и u в простом графе называются *смежными,* если они соединяются каким-то ребром e, про которое говорят, что оно *инцидентно* вершинам u и v. Таким образом, мы можем представлять себе множество E ребер как множество пар смежных вершин, определяя тем самым нерефлексивное, симметричное отношение на множестве V. Отсутствие рефлексивности связано

с тем, что в простом графе нет петель, т. е. ребер, оба конца которых находятся в одной вершине. Симметричность же отношения вытекает из того факта, что ребро *e*, соединяющее вершину *u* с *v*, соединяет и *v* с *u* (иначе говоря, ребра не ориентированы, т.е. не имеют направления). Единственное ребро простого графа, соединяющее пару вершин *u* и *v*, мы будем обозначать как *uv* (или *vu)*.

Логическая матрица отношения на множестве вершин графа, которое задается его ребрами, называется *матрицей смежности.* Симметричность отношения в терминах матрицы смежности *M* означает, что *M* симметрична относительно главной диагонали. А из-за нерефлексивности этого отношения на главной диагонали матрицы *M* стоит значение «Ложь».

Пример 8.1. Нарисуйте граф *G(V, E)* с множеством вершин *V*={*a, b,* c, *d, e*} и множеством ребер *E*={*ab*, *ae*, *bc*, *bd*, *ce*, *de*}. Выпишите его матрицу смежности.

Решение. Граф *G* показан на рис. 8.3.

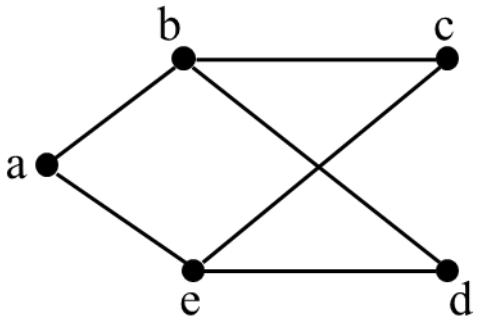


Рисунок 8.3. Граф для примера 8.1

Его матрица смежности имеет вид (строки и столбцы подписаны именами вершин для удобства, элементы самой матрицы смежности имеют значения Ложь или Истина):

	a	b	c	d	e
a	Л	И	Л	Л	И
b	И	Л	И	И	Л
c	Л	И	Л	Л	И
d	Л	И	Л	Л	И
e	И	Л	И	И	Л

Для восстановления графа достаточно только тех элементов матрицы смежности, которые стоят над главной диагональю.

Подграфом графа $G = (V, E)$ называется граф $G' = (V', E')$, в котором $E' \subset E$ и $V' \subset V$.

Пример 8.2. Найдите среди графов *H, K* и *L*, изображенных на рис. 8.4, подграфы графа G.

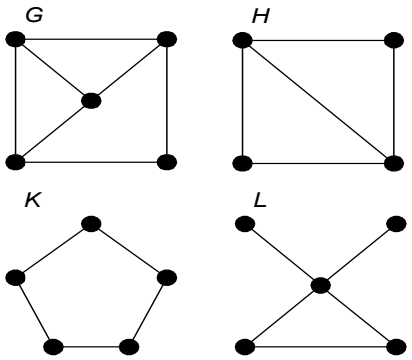


Рисунок 8.4. Графы для примера 8.2

Решение. Обозначим вершины графов *G*, *H* и *K*, как показано рис. 8.5. Тогда, учитывая определение подграфов, видно, что графы *H* и *K* являются подграфами в *G*. Граф *L* не является подграфом в *G*, поскольку у него есть вершина степени 4, а у графа *G* такой нет.

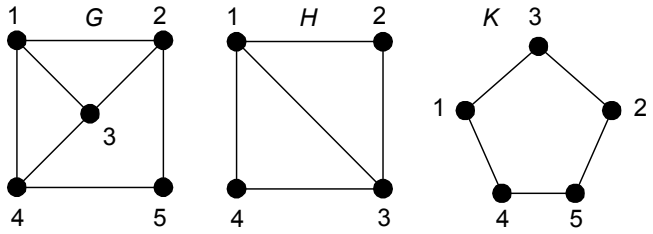


Рисунок 8.5. Граф *G* и его подграфы *H* и *K*

Маршрутом длины *k* в графе *G* называется такая последовательность вершин $v_0, v_1, \ldots, v_k$, что для каждого $i = 1, \ldots, k$ пара $v_{i-1}v_i$ образует ребро графа. Мы будем обозначать такой маршрут $v_0\, v_1 \ldots v_k$. Например, 1 4 3 2 5 – это маршрут длины 4 в графе *G* на рисунке 8.5.

Циклом в графе принято называть последовательность вершин $v_0, v_1, \ldots, v_k$, каждая пара которых является концами одного ребра, причем $v_0 = v_k$, а остальные вершины (и ребра) не повторяются. Иными словами, цикл – это замкнутый маршрут, проходящий через каждую свою вершину и ребро только один раз.

Пример 7.3. Найдите циклы в графе *G* из на рисунке 8.5.

Решение. В этом графе есть два разных цикла длины 5: 132541 и 125431. Мы можем пройти эти циклы как в одном направлении, так и в другом, начиная с произвольной вершины цикла. Кроме того, в графе есть три разных цикла длины 4: 12541, 12341 и 25432, а также два цикла длины 3: 1231 и 1341.

Граф, в котором нет циклов, называется *ациклическим.*

Граф называют *связным,* если любую пару его вершин соединяет какой-нибудь маршрут. Любой общий граф можно разбить на подграфы, каждый из которых окажется связным. Минимальное число таких связных компонент называется *числом связности* графа и обозначается через *c(G).* Вопросы связности имеют важное значение в приложениях теории графов к компьютерным сетям. Следующий алгоритм применяется для определения числа связности графа.

Алгоритм связности

Пусть *G = (V, E)* – граф. Алгоритм предназначен для вычисления значения *c = c (G)*, т. е. числа компонент связности данного графа *G*.

```
begin
  V':= V;
  c:= 0;
  while V'≠∅ do
        begin
        Выбрать y∈V';
        Найти все вершины, соединенные маршрутом с y;
        Удалить вершину y из V' и соответствующие ребра из E;
        c:= c + 1;
  end
end
```

Пример 8.4. Проследите за работой алгоритма связности на графе, изображенном на рис. 8.6.

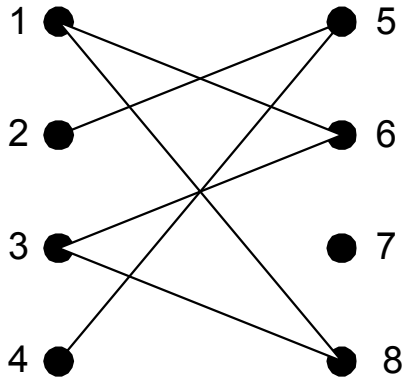


Рисунок 8.6. Граф для примера 8.4

Решение. См. табл. 8.1.

Таблица 8.1. Действие алгоритма связности для примера 8.4

	V'	c
Исходные значения	{1, 2, 3, 4, 5, 6, 7, 8}	0
Выбор $y = 1$	{2, 4, 5, 7}	1
Выбор $y = 2$	{7}	2
Выбор $y = 7$	∅	3

Итак, число связности $c(G) = 3$. Соответствующие компоненты связности приведены на рис. 8.7.

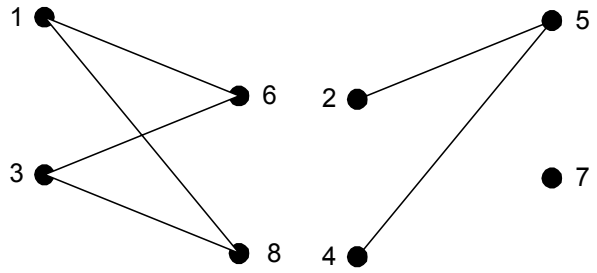


Рисунок 8.7. Компоненты связности для графа, изображенного на рис. 8.6.

8.2. Гамильтоновы графы

Мы рассмотрели эйлеровы графы, в которых существует замкнутый маршрут, проходящий по всем ребрам графа один раз. Похожая задача состоит в поиске цикла, проходящего через каждую вершину графа в точности один раз. Такой цикл, если он существует, называется *гамильтоновым,* а соответствующий граф – *гамильтоновым графом.*

Гамильтоновы графы используются при моделировании задачи о

составлении расписания движения поездов, для телекоммуникационных сетей и т. д. В отличие от задачи Эйлера, простой критерий гамильтоновости графа неизвестен. Поиск критерия остается одной из главных нерешенных задач теории графов. Многие графы являются гамильтоновыми. Предположим, что в каком-то графе любая пара вершин соединена ребром. Такой граф называется *полным* и обозначается через K_n, где n – число его вершин. Очевидно, в любом полном графе можно отыскать гамильтонов цикл.

Полный граф K_5 изображен на рис. 7.8. Его цикл *a b c d e a* является гамильтоновым. В нем есть и другие гамильтоновы циклы. Т. к. каждая вершина смежна с остальными, то начиная с вершины *a,* в качестве второй вершины цикла можно выбрать любую из четырех оставшихся. Далее будет три варианта для выбора третей вершины и два для четвертой, после чего вернемся в вершину *a.* Так, у нас есть 4 × 3 × 2 = 24 цикла. Поскольку каждый цикл можно проходить как в одном направлении, так и в другом, то реально в графе K_5 есть только 12 разных гамильтоновых циклов (две разные последовательности вершин: *a b c d e a* и *a e d c b a* задают один и тот же цикл).

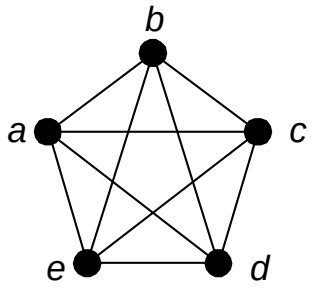


Рисунок 8.8. Полный граф K5

Поиск гамильтонова цикла (если он существует) в произвольном связном графе – задача не всегда простая.

Пример 8.5. Покажите, что граф, изображенный на рис. 8.9, не является гамильтоновым.

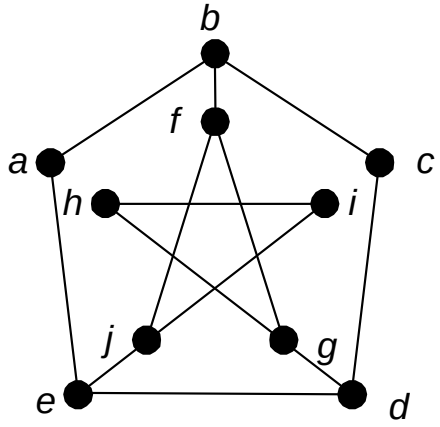


Рисунок 8.9. Пример не гамильтонова графа.

Решение. Предположим, что в связном графе найдется гамильтонов цикл. Каждая вершина *v* включается в гамильтонов цикл *C* выбором двух инцидентных с ней ребер, значит степень каждой вершины в гамильтоновом цикле (после удаления лишних ребер) равна 2. Степени вершин данного графа – 2 или 3. Вершины степени 2 входят в цикл вместе с обоими инцидентными с ними ребрами. Следовательно, ребра *ab*, *ae, cd*, *cb*, *hi*, *hg* и *ij* в том или ином порядке входят в гамильтонов цикл *C* (рис. 8.10).

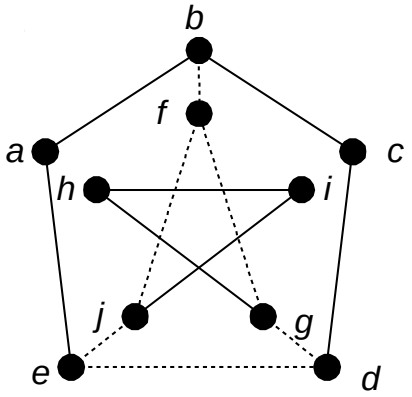


Рисунок 8.10. Ребра, входящие в гамильтонов цикл C

Ребро *bf* не может быть частью цикла *C,* поскольку каждая вершина такого цикла должна иметь степень 2. Значит, ребра *fj* и *fg* обязаны входить в цикл *C,* чтобы включить в него вершину *f*. Но тогда ребра *je* и *gd* никак не могут принадлежать циклу *C*, поскольку в противном случае в нем появятся вершины степени три. Это вынуждает нас включить в цикл ребро *ed*, что приводит нас к противоречию: ребра, которые мы были вынуждены выбрать, образуют два

несвязных цикла, а не один, существование которого мы предполагали. Значит, граф, изображенный на рис. 8.9, не гамильтоновый.

Гамильтоновы графы применяются для моделирования многих практических задач. Основой всех таких задач служит классическая задача коммивояжера: *коммивояжер должен совершить поездку по городам и вернуться обратно, побывав в каждом городе ровно один раз, сведя при этом затраты на передвижение к минимуму*.

Графическая модель задачи коммивояжера состоит из гамильтонова графа, вершины которого изображают города, а ребра – связывающие их дороги. Граф является *нагруженным* – его каждое ребро оснащено *весом*, обозначающим транспортные затраты, необходимые для путешествия по соответствующей дороге. Для решения задачи нам необходимо найти гамильтонов цикл минимального общего веса.

К сожалению, эффективный алгоритм решения данной задачи пока не известен. Для сложных сетей число гамильтоновых циклов, которые необходимо просмотреть для выделения минимального, непомерно огромно. Однако существуют алгоритмы поиска *субоптимального решения.* Субоптимальное решение необязательно даст цикл минимального общего веса, но найденный цикл будет, как правило, значительно меньшего веса, чем большинство произвольных гамильтоновых циклов. Один из таких алгоритмов мы сейчас и изучим.

Алгоритм ближайшего соседа

Этот алгоритм выдает субоптимальное решение задачи коммивояжера, генерируя гамильтоновы циклы в нагруженном графе с множеством вершин *V*. Цикл, полученный в результате работы алгоритма, будет совпадать с конечным значением переменной *маршрут*, а его общая длина – конечное значение переменной *w*.

```
begin
  Выбрать v∈V;
  маршрут:= v;
  w:= 0;
```

```
  v' := v;
  Отметить v';
  while остаются неотмеченные вершины do
       begin
          Выбрать неотмеченную вершину u, ближайшую к v';
          маршрут:=маршрут u
          w:= w + вес ребра v'u;
          v':= u;
          Отметить v';
       end
  маршрут:= маршрут v;
  w:= w + вес ребра v'v;
end
```

Пример 8.6. Примените алгоритм ближайшего соседа к графу, изображенному на рис. 8.11. За исходную вершину возьмите вершину *D*.

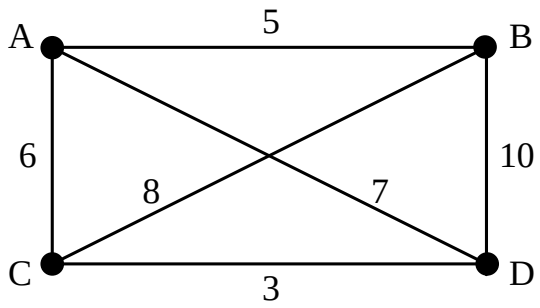


Рисунок 8.11.

Решение. См. табл. 8.2.

Таблица 8.2. Действие алгоритма ближайшего соседа для графа, изображенного на рис. 8.11

	u	***маршрут***	***w***	***v'***
Исходные значения	–	*D*	0	*D*
	C	*DC*	3	*C*
	A	*DCA*	9	*A*
	B	*DCAB*	14	*B*
Последний проход цикла	*B*	*DCABD*	24	*B*

В результате работы алгоритма был найден гамильтонов цикл *DCABD* общего веса 24. Делая полный перебор всех циклов в этом маленьком графе, можно обнаружить еще два других гамильтоновых цикла: *ABCDA* общего веса 23 и *ACBDA* общего веса 31. В полном графе с двадцатью вершинами существует приблизительно $6{,}1\times10^{16}$ гамильтоновых циклов, перечисление которых требует чрезвычайно много машинной памяти и времени.

8.3. Деревья

Известен класс графов, называемых деревьями, которые особенно интенсивно используются в вычислительных приложениях. Граф $G = (V, E)$ называется *деревом,* если он связен и ацикличен (т. е. не содержит циклов). Пусть $G = (V, E)$ – граф с n вершинами и m ребрами. Можно сформулировать несколько необходимых и достаточных условий, при которых G является деревом:

- любая пара вершин в G соединена единственным путем;
- G связен, и $m = n - 1$;
- G связен, а удаление хотя бы одного ребра нарушает связность графа;
- G ацикличен, но если добавить хотя бы одно ребро, то появится цикл.

Эквивалентность большинства из этих условий устанавливается без особого труда. Наиболее сложно разобраться со вторым из них. В следующем примере мы докажем, что дерево с n вершинами имеет $n - 1$ ребро.

Пример 8.7. Докажите с помощью математической индукции, что для дерева T с n вершинами и m ребрами выполнено соотношение $m = n - 1$.

Решение. Поскольку дерево с единственной вершиной вообще не содержит ребер, то доказываемое утверждение справедливо при $n = 1$.

Рассмотрим дерево T с n вершинами и m ребрами, где $n > 1$ и предположим, что любое дерево с $k < n$ вершинами имеет ровно $k - 1$ ребро.

Удалим ребро из T. По третьему свойству дерево T после этой процедуры превратится в несвязный граф. Получится ровно две компоненты связности, ни одна из которых не имеет циклов (в противном случае исходный граф T тоже содержал бы циклы и не мог бы являться деревом). Таким образом, полученные компоненты связности – тоже деревья.

Обозначим новые деревья через T_1 и T_2. Пусть n_1 – количество вершин у дерева T_1, а n_2 – у T_2. Поскольку $n_1 + n_2 = n$, то $n_1 < n$ и $n_2 < n$, следовательно, по предположению индукции, дерево T_1 имеет $n_1 - 1$ ребро, а T_2 имеет $n_2 - 1$ ребро. Следовательно, исходное дерево T насчитывало (с учетом одного удаленного) $(n_1 - 1) + (n_2 - 1) + 1 = n - 1$ ребро, что и требовалось доказать.

Несложно доказать, что в любом связном графе найдется подграф, являющийся деревом. Подграф в *G*, являющийся деревом и включающий в себя все вершины *G*, называется *остовным деревом*. Остовное дерево в графе *G* строится просто: выбираем произвольное его ребро и последовательно добавляем другие ребра, не создавая при этом циклов, до тех пор, пока нельзя будет добавить никакого ребра, не получив при этом цикла. Благодаря Примеру 7.7 мы знаем, что для построения остовного дерева в графе из n вершин необходимо выбрать ровно $n - 1$ ребро.

Пример 8.8. Найдите два разных остовных дерева в графе, изображенном на рис. 8.12.

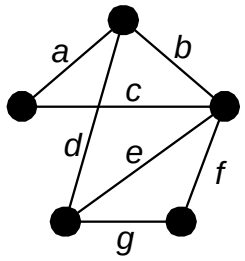


Рис. 8.12. Связный граф G

Решение. В этом графе существует несколько остовных деревьев. Одно из них получается последовательным выбором ребер: *a*, *b*, *d* и *f*. Другое – *b*, *c*, *e* и *g*. Названные деревья являются остовными и показаны на рис. 8.13.

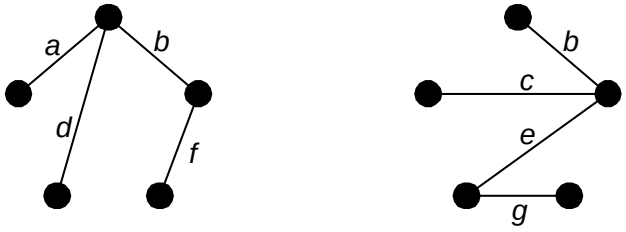


Рис. 8.13. Остовные деревья графа G.

Процесс, описанный в примере 8.8, можно приспособить для решения задачи поиска кратчайшего соединения: нужно построить железнодорожную сеть, связывающую некоторое число городов. Известна стоимость строительства отрезка путей между любой парой городов. Требуется найти сеть минимальной

стоимости.

На языке теории графов нам нужно в нагруженном графе найти остовное дерево наименьшего общего веса. Такое дерево принято называть минимальным остовным деревом или сокращенно – МОД. В отличие от задачи коммивояжера, здесь есть эффективные алгоритмы, которые находят действительно минимальное остовное дерево. Рассмотрим два из них. Первый –алгоритм Прима, с которым мы познакомились на первой лекции при решении задачи поиска кратчайшего соединения для набора из шести городов. Второй – алгоритм Краскала, или «жадный» алгоритм.

Алгоритмы поиска минимального остовного дерева

Алгоритм Прима

Включим в остов произвольную вершину связного графа G=(V,E). Выберем минимальное из всех инцидентных этой вершине ребер. Его вторую вершину включим в остов. Рассмотрим все ребра, исходящие из вершин, включенных в остов, и из них выберем ребро с минимальным весом. Его вторую вершину включим в остов. Повторяем этот пункт, пока не все вершины включены в остов. В этом алгоритме не нужно следить за образованием циклов.

Псевдокод алгоритма:

```
begin
    v:=произвольная вершина графа G=(V,E).
    M:={v}.
    V':=V\{v}.
    while V'≠∅ do
    begin
    v':=вершина из V', ближайшая к M.
    M:=M∪{v'}.
    V':=V\M
    end
end
```

Пример 8.9. В табл. 8.3 дано расстояние (в км) между пятью деревнями *A, B, C, D* и *E.* Найдите минимальное остовное дерево.

Таблица 8.3. Данные для примера 8.9

	A	***B***	***C***	***D***	***E***
A	–	13	3	9	9

B	13	–	11	11	13
C	3	11	–	9	7
D	9	11	9	–	2
E	9	13	7	2	–

Решение. Выберем, например, вершину B. Ближайшие вершины C и D. Выберем C. Далее добавляются последовательно вершины E и D. Построенное минимальное остовное дерево имеет вес 23.

Алгоритм Прима легко программируется. Выберем, например, вершину B и пометим строку с ее именем. Столбец B, в котором указаны все ребра, инцидентные этой вершине, удаляем из рассмотрения. Это ускорит дальнейший поиск. Выбираем в строке B одну из ближайших вершин, например C. Помечаем ее и удаляем из рассмотрения столбец C. В помеченных строках выбираем ближайшую вершину, инцидентную связанным вершинам, удаляем соответствующий столбец. Процесс продолжается, пока есть непомеченные вершины.

Алгоритм Краскала

Пусть $G = (V, E)$ – связный взвешенный граф. Алгоритм строит МОД в графе G, последовательно выбирая ребра наименьшего возможного веса до образования остовного дерева. МОД в памяти компьютера хранится в виде множества T ребер.

```
begin
  e:= ребро графа G с наименьшим весом;
  T:= {e};
  E':= E\{e}
  while E' ≠∅ do
        begin
           e':= ребро наименьшего веса из E';
           T := T ∪{e'};
           E':= множество ребер из E'\{T}, чье добавление к T не ведет к образованию
           циклов;
        end
end
```

Решение примера 8.9. Ребра выбираются следующим образом: первое – ребро DE веса 2; второе – AC веса 3; третье – CE веса 7. На этой стадии строящееся дерево выглядит так, как на рис. 8.14:

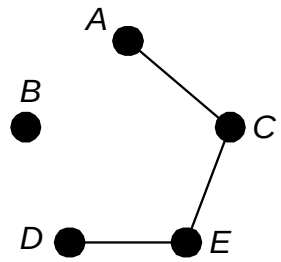


Рисунок 8.14. Вид дерева после трех шагов построения остовного дерева в примере 8.9.

Следующие по весу ребра – *AD, AE* и *CD,* каждое из которых имеет вес 9. Однако какое бы из них мы ни добавили, получится цикл. Поэтому перечисленные ребра следует исключить из числа доступных для построения. Далее идут ребра *BC* и *BD* веса 11. Можно присоединить любое из них, получив при этом два разных МОД: *{AC, BC, CE, DE}* или *{AC, BD, CE, DE}* веса 23 каждое.

Дерево с корнем

Часто в моделях используются деревья, представляющие информацию с учетом естественной иерархической структуры, такие как, например, генеалогическое древо. На рис. 8.15 показаны некоторые члены семьи Бернулли, каждый из которых был известным швейцарским математиком.

Генеалогическое древо можно изобразить и более сжато. Схема, приведенная на рис. 8.16, представляет собой пример так называемого дерева с корнем. *Деревом с корнем* называется дерево с одной выделенной вершиной, которая и является *корнем* дерева. Корень, в некотором смысле, можно назвать главной вершиной (например, родоначальник математиков Бернулли). Вершины дерева, лежащие непосредственно под данной, называются *сыновьями.* С другой стороны, вершина, стоящая непосредственно перед сыном, называется ее *отцом.*

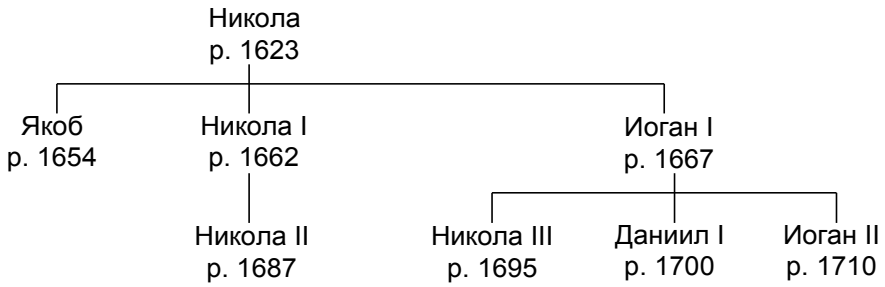


Рисунок 8.15. Династия швейцарских математиков Бернулли.

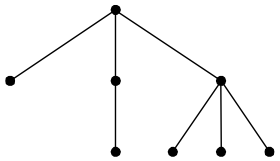


Рисунок 8.16. Схема генеалогического древа Бернулли

Дерево с корнем можно определить рекуррентным образом. Отдельная вершина является деревом с корнем (она же служит и корнем такого дерева). Если T_1, T_2, ..., T_k – несвязанные друг с другом деревья с корнями v_1, v_2, ...,v_k, то граф, получающийся присоединением новой вершины v к каждой из вершин v_1, v_2, ...,v_k отдельным ребром, является деревом T с корнем v. Вершины v_1, v_2, ...,v_k графа T – это сыновья корня v. Мы изображаем такое дерево с корнем, расположенным наверху, и сыновьями, стоящими ниже, непосредственно под корнем (рис. 8.17). Каждую вершину дерева с корнем можно рассматривать как корень другого дерева, которое «растет» из него. Мы будем называть его *поддеревом* дерева T.

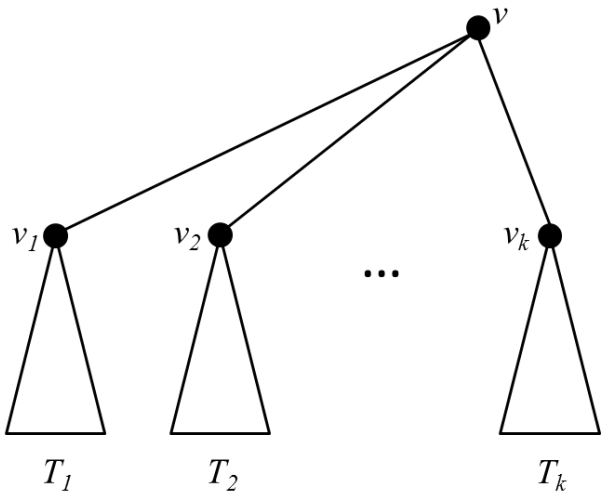


Рисунок 8.17. Рекуррентное определение дерева.

Как мы уже говорили, вершина на самом верху дерева – его корень. Вершины, которые находятся в самом низу дерева (и не имеют сыновей) принято называть *листьями*. Остальные вершины, отличные от корня и листьев, называют *внутренними*.

Область применения деревьев с корнем обширна – это и информатика, и биология, и менеджмент, и т.п.

Дерево решений

Возможности выбора при решении некоторой проблемы можно представить в виде дерева, где в корне – проблема, ребру соответствует один из вариантов решения проблемы, вершине – новая ситуация, возникающая в результате реализации приписанного ребру варианта. Такой трактовке соответствует граф типа дерева с корнем, получивший название «дерево решений».

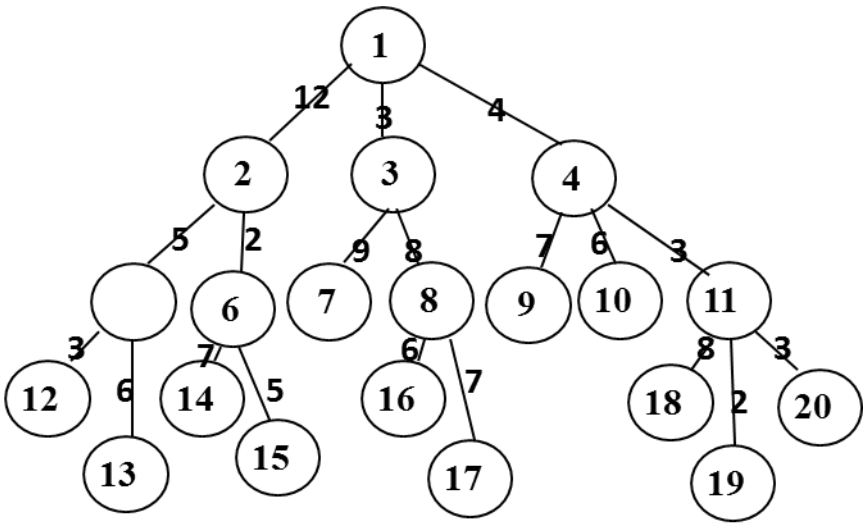


Рисунок 8.18. Пример дерева решений с нагруженными ребрами.

Предположим, что нужно оценить эффективность принятого выбора. Тогда возникает задача поиска оптимального решения среди возможных путей от корня (когда проблема поставлена) к одному из листьев (когда проблема решена).

Если дерево решений очень велико, то поиск путём полного перебора всех возможных решений невозможен. Рассмотрим две стратегии поиска, связанные с сокращённым перебором решений: метод ветвлений и метод ветвей и границ.

Метод ветвлений

Поиск начинается от корня дерева. Выбирают минимальное по весу исходящее ребро и по нему переходят к следующей вершине. Если эта вершина не является листом, снова переходят по минимальному исходящему ребру, пока не будет достигнут лист. Алгоритм очень просто реализуется, но решение может быть далёким от минимального.

Метод ветвей и границ

На каждом шаге выделяется множество вершин в качестве границы, веса вершин которой оцениваются длиной пути до неё от корня. На границе выбирается вершина с минимальной оценкой. Если эта вершина является листом, то решение найдено – путь до этого листа и будет минимальным решением, если нет, то строится новая граница заменой найденной вершины на

вершины, связанные с ней по исходящим ребрам, и поиск решения продолжается. Метод гарантирует получение минимального решения, однако часто связан с большим объёмом вычислений.

Пример 8.10. Определить по методу ветвлений и методу ветвей и границ минимальный путь в дереве, изображенном на рис. 8.18.

Решение.

Метод ветвлений.

Путь пройдет по вершинам 1,3,8,16 и равен 17.

Метод ветвей и границ.

Шаг 1. Границей является корень дерева, который заменяется вершинами, связанными с ним по исходящим ребрам – вершинами 2 (вес 12), 3 (3), 4 (4). В скобках приведены веса, приписываемые ребрам. Выбираем вершину 3 (3). Вершина 3 не является листом, поэтому она удаляется из границы, а вместо неё в границу вводятся вершины 7 (расстояние от корня 3+9=12) и 8 (3+8=11).

Шаг 2. Новая граница: 2(12), 7(12), 8(11), 4(4). Выбираем вершину 4, которую заменяют вершинами 9 (11), 10 (10) и 11 (7).

Шаг 3. Новая граница: 2 (12), 7 (12), 8 (11), 9 (11), 10 (10), 11 (7). Далее выбираем вершину 11, вместо которой вводятся связанные с ней по исходящим дугам вершины 18 (11), 19 (9), 20 (10).

Шаг 4. Новая граница состоит из вершин 2 (12), 7 (12), 8 (11), 9 (11), 10 (10), 18 (11), 19 (9) и 20 (10). Минимальный суммарный вес (или расстояние) у вершины 19 (9) – это лист. Минимальный путь: $1 \rightarrow 4 \rightarrow 11 \rightarrow 19$. Вес решения равен 9.

Метод ветвей и границ дает точное решение задачи.

Бинарные деревья

Для приложения к информатике наиболее важны так называемые *двоичные*, или *бинарные*, деревья с корнем. Двоичное дерево отличает от остальных то, что каждая его вершина имеет не более двух сыновей. В двоичном дереве с корнем от каждой вершины идет вниз не более двух ребер.

Таким образом, каждой вершине двоичного дерева с корнем соответствует не более чем два поддерева, которые называют *левым* и *правым* поддеревьями этой вершины. Если оказалось, что у какой-то вершины дерева отсутствует потомок слева, то ее левое поддерево называют *нулевым деревом* (т.е. нулевое дерево – это дерево без единой вершины). Аналогично, если у вершины отсутствует правый потомок, то ее правое поддерево будет нулевым.

Пример 8.11. Пусть *T* – двоичное дерево с корнем, изображенное на рис. 7.18.

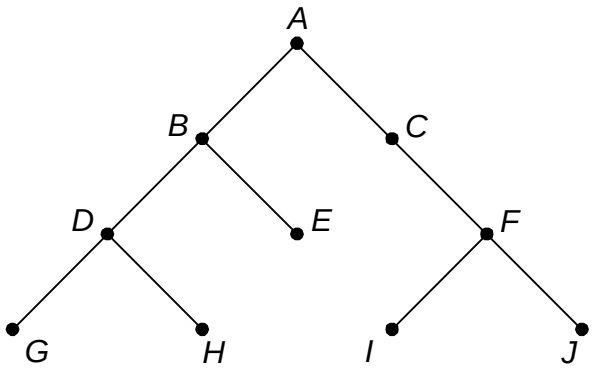


Рис. 8.19. Двоичное дерево *T* с корнем.

В этом примере корнем *T* является вершина *A*. Корнем левого поддерева вершины *B* является вершина *D*. Листьями *T* являются вершины *G, H, I, J*. Сыном вершины *C* является вершина *F*.

8.4. Сортировка и поиск

Двоичные деревья с корнем широко используются при решении задач о выборе, в частности, когда нужно классифицировать упорядоченные данные или вести в них поиск.

Упорядоченные данные, такие как множество чисел, упорядоченных по величине или множество строк литер, упорядоченных лексикографически (в алфавитном порядке), можно организовать в виде вершин двоичного дерева с корнем в соответствии с их порядком. При этом стремятся к тому, чтобы данные, стоящие в левом поддереве данной вершины *v* были бы меньше данных, соответствующих этой вершине, а данные, расположенные в правом ее

поддереве, – больше. Дерево данных, удовлетворяющее указанному условию, называют *двоичным деревом поиска*.

Достоинство организации упорядоченных данных в виде двоичного дерева поиска заключается в возможности создания эффективного алгоритма поиска каких-то конкретных данных, включения новых данных в дерево и печати всей информации, содержащейся в дереве в виде упорядоченного списка.

Предположим, что в университете хранится список студентов, упорядоченный в алфавитном порядке, в котором кроме фамилий имеются дополнительные сведения о студентах. Допустим также, что возникла необходимость найти какую-то информацию в списке или добавить новые записи к списку. Мы сейчас познакомимся с алгоритмами, которые осуществляют поиск конкретной информации, добавляют новых студентов к списку и выводят на печать все записи в алфавитном порядке.

Записи о студентах организованы в двоичное дерево поиска (каждая запись соответствует одной вершине), и наши алгоритмы будут исследовать вершины этого дерева. Поскольку каждая вершина является также и корнем некоторого двоичного дерева поиска, алгоритмы будут последовательно проверять левые и правые поддеревья вершин. Чтобы это осуществить, необходимо приписать каждой вершине некоторый *ключ* для идентификации и ссылок на ее левое и правое поддеревья (в структурах данных для этих целей используются так называемые дважды связанные списки). Из всех ключей организуется линейно упорядоченное множество (в нашей ситуации оно упорядочено лексикографически).

Алгоритм поиска

Алгоритм определяет, является ли данная запись *(ключ поиска)* вершиной в двоичном дереве поиска, сравнивая ключ поиска с ключом корня дерева, и, при необходимости, осуществляет аналогичные сравнения в левом или правом поддеревьях.

Поиск (дерево)
begin

```
  if дерево нулевое then
       поиск:= ложь;
  else
       if ключ поиска = ключ корня then
          поиск := истина;
       else
          if ключ поиска < ключ корня then
                  поиск:= поиск (левое поддерево);
          else
                  поиск:= поиск (правое поддерево);
end
```

Пример 8.12. Проследите за работой алгоритма, реализующего двоичное дерево поиска, изображенное на рис. 8.20. Известно, что ключ поиска – буква *R*, а ключи вершин упорядочены лексикографически.

Решение. Поскольку $R > K$, то поиск продолжается в правом поддереве вершины *K*. Так как $R < T$, процесс поиска переключается на левое поддерево вершины *T*. Наконец, ввиду неравенства $R \neq M$ и отсутствия поддеревьев у вершины *M*, алгоритм заканчивается и сообщает, что искомая вершина не была найдена.

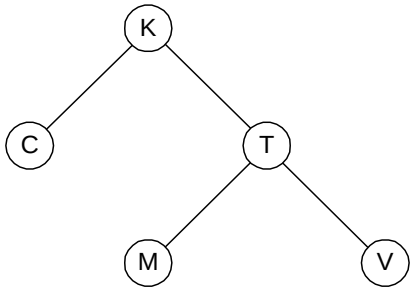


Рисунок 8.20.

Алгоритм вставки

Алгоритм добавляет новые вершины *(ключи вставок)* в двоичное дерево поиска, создавая при этом новую вершину слева или справа от уже существующей. Это делается таким образом, чтобы все ключи вершин в получившемся дереве подчинялись установившемуся порядку.

```
Вставка (запись, дерево)
begin
  if дерево нулевое then
       добавить новую вершину;
  else
```

```
        if ключ вставки= ключ корня then
        вывести на печать: «запись содержится в дереве»;
        else
            if ключ вставки < ключ корня then
            вставка:= вставка (запись, левое поддерево);
        else
            вставка:= вставка(запись, правое поддерево);
end
```

Пример 8.13. Проследите за работой алгоритма вставки вершин *R, B, D, F, H, J, I, G, E, A* и *L* в дерево из примера 8.12.

Решение. Поскольку *R* > *K*, мы применяем алгоритм вставки к правому поддереву вершины *K*. Далее мы видим, что *R* < *T*. Значит, алгоритм вставки переключается на левое поддерево вершины *T*. Так как *R* > *M* и правое поддерево вершины *M* нулевое, то мы ставим вершину *R* справа от *M*. Аналогично делаем вставку вершин *B, D, F, H, J, I, G, E, A* и *L* и получаем дерево, изображенное на рис. 8.21.

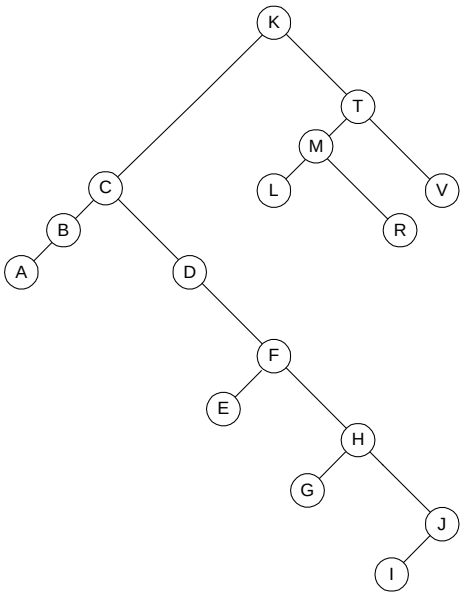


Рис. 8.21. Вставка новых элементов в двоичное дерево поиска

Алгоритм вставки можно использовать для создания двоичного дерева поиска, начиная с нулевого дерева и последовательно добавляя новые данные в удобном для нас порядке.

Алгоритм правильного обхода

Алгоритм выводит на печать в надлежащем порядке всю информацию,

содержащуюся в двоичном дереве поиска. При этом все вершины дерева осматриваются в определенном порядке. Алгоритм работает следующим образом. Для каждой вершины, начиная с корня, печатается вся информация, содержащаяся в вершинах левого поддерева. Затем выводится информация, хранящаяся в этой вершине и, наконец, информация, соответствующая вершинам правого поддерева.

Правильный обход (дерево)

```
begin
  if дерево нулевое then
        ничего не делать;
  else
        begin
           правильный обход (левое поддерево);
           напечатать корневой ключ;
           правильный обход (правое поддерево);
        end
end
```

Пример 8.14. Примените алгоритм правильного обхода к дереву, построенному в примере 8.13.

Решение. После работы алгоритма над указанным деревом выводится список: *A, B, C, D, E, F, G, H, I, J, K, L, M, R, T, V*.

Алгоритм обеспечивает обход дерева против часовой стрелки (рис 8.22) и печать информации, содержащейся в вершинах, как только проходит *под* вершиной.

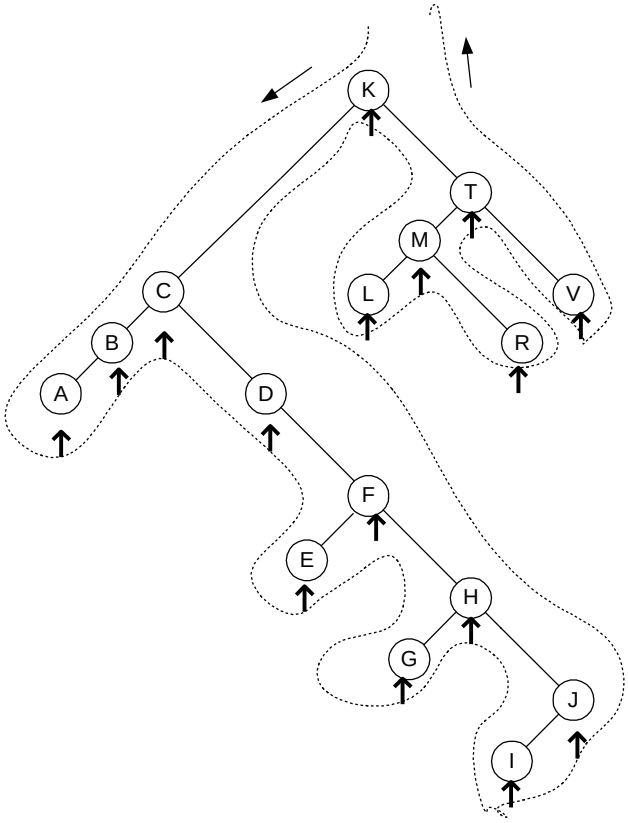


Рис. 8.22. Правильный обход в двоичном дереве поиска

9. Ориентированные графы

Мы впервые столкнулись с ориентированными графами в главе 4 при описании бинарных отношений. Ориентированные графы или, для краткости, орграфы используются для моделирования ситуаций, в которых есть отношение частичного порядка между объектами. Возникающие при этом схемы служат для изображения информационных потоков, сетевого планирования и планирования заданий.

Основная часть главы посвящена проблеме поиска путей в сетях, в том числе и компьютерных. Существование путей мы будем устанавливать с помощью матриц достижимости. Это матрицы замыкания относительно транзитивности отношения, задаваемого ребрами сети.

9.1. Ориентированные графы

Ориентированный граф, или *орграф*, представляет собой пару $G = (V, E)$, где V – конечное множество *вершин,* а E – множество ориентированных ребер, которые называются дугами. Множество E – это отношение на V. Графическое изображение графа состоит из множества помеченных вершин с дугами.

Дугу, соединяющую пару вершин u и v орграфа G, будем обозначать через uv. В простом орграфе отсутствуют петли и кратные дуги. Следовательно, для любой пары вершин u и v в орграфе найдется не более одной дуги uv из вершины u в v и не более одной дуги vu *из* v в u. Если uv – дуга орграфа, то вершину u называют *антецедентом* вершины v.

На рис. 9.1 приведен пример простого орграфа с множеством вершин $V = \{a, b, c, d\}$ и множеством дуг $E = \{ab, bd, cb, db, dc\}$. Вершины a, c и d здесь – антецеденты b.

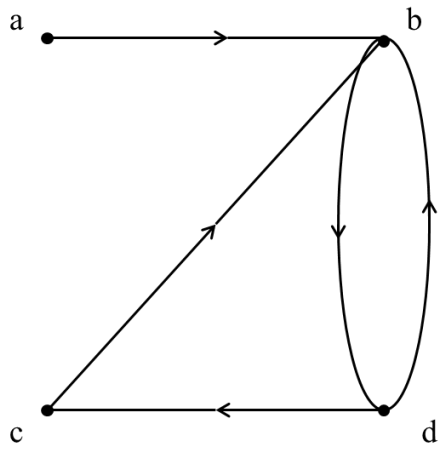


Рисунок 9.1. Пример ориентированного графа.

Матрицей смежности данного графа является несимметричная матрица

$$\begin{array}{c} \\ a \\ b \\ c \\ d \end{array}\begin{bmatrix} Л & И & Л & Л \\ Л & Л & Л & И \\ Л & И & Л & Л \\ Л & И & И & Л \end{bmatrix}.$$

Путем длины k в орграфе называют последовательность различных вершин $v_0, v_1, ..., v_k$, каждая пара $v_{i-1}v_i$ которой образует дугу ($i = 1,2,..., k$).

Контуром в орграфе G принято называть последовательность вершин $v_0, v_1, ..., v_k$, образующую путь, в которой первая вершина v_0 совпадает с последней v_k, а других повторяющихся вершин в ней нет. Орграф G называют *бесконтурным,* если в нем нет контуров.

Бесконтурные орграфы полезны в качестве моделей ситуаций, задачи в которых должны выполняться в определенном порядке (контур в такой интерпретации означает, что та или иная задача выполняется с некоторой периодичностью и предшествует сама себе). В задаче о планировании заданий соответствующий бесконтурный орграф называют «система ПЕРТ». ПЕРТ – это система планирования и руководства разработками. На английском языке аббревиатура PERT – это сокращение от «Program Evaluation and Review Technique». ПЕРТ была разработана для управления процессом конструирования подводной лодки для военно-морского флота США.

Пример 9.1. Для получения степени магистра биологии студенту

университета необходимо прослушать восемь курсов, которые некоторым образом зависят друг от друга. Эта зависимость представлена в табл. 9.1. Изобразите систему ПЕРТ, иллюстрирующую приоритетную структуру курсов.

Таблица 9.1. Данные для примера 9.1

	Название курсов	Предварительные
A	Биотехнология	B
B	Начальный курс биотехнологии	C
C	Цитология	H
D	Структура ДНК	C
E	Энзимология	D, G
F	Диетология	E
G	Генная инженерия	C
H	Биология человека	Нет

Решение. Система ПЕРТ (рис. 9.2) – это орграф, представляющий данную приоритетную структуру. Вершины орграфа – это восемь курсов. Для краткости обозначим курсы буквами латинского алфавита от A до H. Дуги орграфа отражают представленные в таблице требования, необходимые для усвоения данного курса.

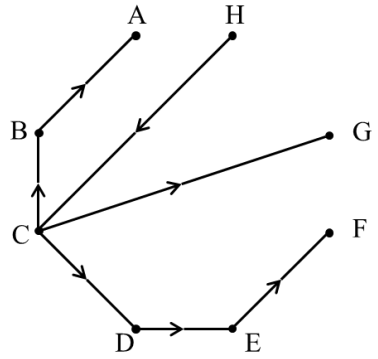


Рисунок 9.2. Система ПЕРТ: курсов приоритетная структура.

Предположим, что необходимо определить порядок, в котором следует изучать предметы, учитывая их зависимость друг от друга. Это можно сделать с помощью *алгоритма топологической сортировки.* Алгоритм создает *последовательность согласованных меток* для вершин бесконтурного орграфа таким образом, что если 1, 2, 3, ..., n – метки вершин и uv – дуга орграфа, идущая от вершины u с меткой i к вершине v с меткой j, то $i < j$.

Алгоритм топологической сортировки

Алгоритм генерирует последовательность согласованных меток для вершин бесконтурного орграфа $G = (V, E)$. В самом начале работы алгоритма антецеденты каждой вершины v записываются в множество $A(v)$. Такие множества можно легко получить из матрицы смежности орграфа, выбирая из столбцов, соответствующих вершинам, строки с ненулевыми значениями (здесь и далее для удобства мы будем в логических (или булевых) матрицах использовать символы 0 и 1 вместо Л и И):

	A	B	C	D	E	F	G	H
A	0	0	0	0	0	0	0	0
B	1	0	0	0	0	0	0	0
C	0	1	0	1	0	0	1	0
D	0	0	0	0	1	0	0	0
E	0	0	0	0	0	1	0	0
F	0	0	0	0	0	0	0	0
G	0	0	0	0	1	0	0	0
H	0	0	1	0	0	0	0	0

Алгоритм присваивает метки вершинам. Каждая вершина получает очередную метку в том случае, если у нее нет неотмеченных антецедентов.

```
begin
  for v∈V do
  вычислить A(v);
  label := 0;
  while остаются неотмеченные вершины, для которых A(v) = ∅ do
        begin
        label:= label+1;
        u:= вершина с A(u) = ∅;
        присвоить метку вершине u;
        for каждой неотмеченной вершины v∈V do;
            A(v):=A(v)\{u};
        end
end
```

Сейчас найдем последовательность меток для орграфа, моделирующего нашу задачу и изображенного на рис. 9.2.

Шаг 0. Множество антецедентов выглядит следующим образом: $A(A) = \{B\}$, $A(B) = \{C\}$, $A(C) = \{H\}$, $A(D) = \{C\}$, $A(E) = \{D,G\}$, $A(F) = \{E\}$, $A(G) = \{C\}$, $A(H) = \varnothing$.

Шаг 1. Первый проход цикла **while**. Назначить метку 1 вершине *H* и удалить вершину *H* из всех оставшихся множеств *A*(*v*).

A(*A*) = {*B*}, *A*(*B*) = {*C*}, *A*(*C*) = ∅, *A*(*D*) = {*C*}, *A*(*E*) = {*D*,*G*}, *A*(*F*) = {*E*}, *A*(*G*) = {*C*}.

Шаг 2. Второй проход цикла **while**. Назначить метку 2 вершине *C* и удалить вершину *C* из всех оставшихся множеств *A*(*v*).

A(*A*) = {*B*}, *A*(*B*) = ∅, *A*(*D*) = ∅, *A*(*E*) = {*D, G*}, *A* (*F*) = {*E*}, *A* (*G*) = ∅.

Шаг 3. Третий проход цикла **while**. Теперь у нас появился выбор: какой вершине присвоить очередную метку? В зависимости от нашего выбора получатся разные последовательности меток. Присвоим, например, метку 3 вершине *B* и удалим *B* из множеств *A*(*v*).

A(*A*) = ∅, *A* (*D*) = ∅, *A* (*E*) = {*D, G*}, *A* (*F*) = {*E*}, *A* (*G*) = ∅.

Шаг 4. Четвертый проход цикла **while**. Мы снова стоим перед выбором. Назначим метку 4 вершине *A* и удалим вершину *A* из *A*(*v*).

A(*D*) = ∅, *A* (*E*) = {*D, G*}, *A* (*F*) = {*E*}, *A* (*G*) = ∅.

Шаг 5. Пятый проход цикла **while**. Назначим метку 5 вершине *D* и удалим вершину *D* из *A* (*v*). *A* (*E*) = {*G*}, *A* (*F*) = {*E*}, *A* (*G*) = ∅.

Шаг 6. Шестой проход цикла **while**. Назначим метку 6 вершине *G* и удалим вершину *G* из *A* (*v*). *A* (*E*) = ∅, *A* (*F*) = ∅.

Шаг 7. Седьмой проход цикла **while**. Назначаем метку 7 вершине *E* и удаляем *E* из списка *A* (*v*). Останется только *A* (*F*) = ∅.

Шаг 8. Последний проход цикла **while.** Назначаем метку 8 вершине *F*.

Итак, один из возможных приоритетных списков: *H, C, B, A, D, G, E, F*. Он дает нам порядок, изображенный на рис. 9.3, в котором можно изучать курсы, соблюдая нужную последовательность.

Так как в вершине *C* мы сделали произвольный выбор метки при равных условиях, то возможны и другие последовательности, в том числе и параллельное выполнение процессов.

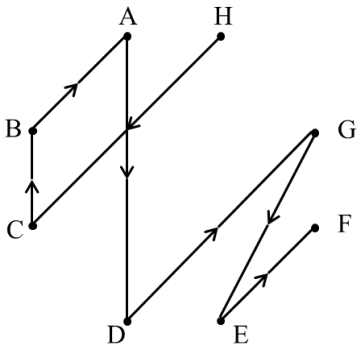


Рисунок 9.3. Возможная последовательность изучения курсов.

В математике алгоритм топологической сортировки применяется для преобразования матриц, например, для ускорения матричных вычислений. Перестроим нашу матрицу смежности с учетом результатов сортировки, не изменяя при этом значения матричных элементов.

	H	C	B	A	D	G	E	F
H	0	1	0	0	0	0	0	0
C	0	0	1	0	1	1	0	0
B	0	0	0	1	0	0	0	0
A	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	1	0
G	0	0	0	0	0	0	1	0
E	0	0	0	0	0	0	0	1
F	0	0	0	0	0	0	0	0

Мы получили матрицу смежности, с такими же связями, как и у исходной, но с ненулевыми элементами, лежащими только справа от главной диагонали.

9.2. Пути в орграфах

Ориентированные графы успешно применяются для схематичного изображения транспортных линий, коммуникационных сетей между компьютерами и многих других задач. В таких сетях важно знать последовательность выключений любого соединения (дуги или вершины) по всей сети. Например, если самолет не может приземлиться для дозаправки в некотором городе вследствие неблагоприятных погодных условий, то ошибка в его переадресации грозит катастрофой: ему может не хватить горючего для

достижения неправильно назначенного аэропорта. Аналогично, если одна или несколько цепей в компьютерной сети не работают, то для некоторых пользователей отдельные серверы могут оказаться недоступными.

Таким образом, мы приходим к задаче о поиске путей между произвольной парой вершин в ориентированном графе.

Пусть отношение R на множестве V задано орграфом $G = (V, E)$ с n вершинами, а M – его матрица смежности. Напомним, что цифрой 1 на пересечении i-той строки и j-го столбца мы обозначаем наличие дуги от вершины с номером i к вершине с номером j или в терминах отношения цифра 1 указывает на то, что упорядоченная пара (v_i, v_j) принадлежит отношению R. Дуга, по определению, является путем длины 1. Булево произведение матрицы смежности M с самой собой дает матрицу M^2, которая является матрицей смежности композиции отношения $R \circ R$. В этой матрице элемент равный 1 символизирует наличие пути длины 2. По матрице $M^3 = M \times M \times M$ можно определить все пути длины 3, матрица M^k дает сведения о путях длины k. Наконец, в *матрице достижимости* $M^* = M$ **или** M^2 **или … или** M^n, где n – максимальная длина пути, записаны пути любой длины между вершинами.

Если у нас есть две логические матрицы одного размера, то в результате логической операции **или** получится матрица, чьи элементы – результат применения этой операции к соответствующим элементам двух данных матриц. Более точно:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1(n-1)} & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2(n-1)} & a_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{m(n-1)} & a_{mn} \end{bmatrix} \text{ или } \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1(n-1)} & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2(n-1)} & b_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ b_{m1} & b_{m2} & \dots & b_{m(n-1)} & b_{mn} \end{bmatrix} = \begin{bmatrix} a_{11} \text{ или } b_{11} & \dots & a_{1n} \text{ или } b_{1n} \\ a_{21} \text{ или } b_{21} & \dots & a_{2n} \text{ или } b_{2n} \\ \dots & \dots & \dots \\ a_{m1} \text{ или } b_{m1} & \dots & a_{mn} \text{ или } b_{mn} \end{bmatrix}$$

Матрица достижимости орграфа $G = (V, E)$ фактически является матрицей замыкания по транзитивности E^* отношения E на вершинах орграфа G.

Пример 9.2. Вычислите матрицу достижимости орграфа, представленного на рис. 9.4.

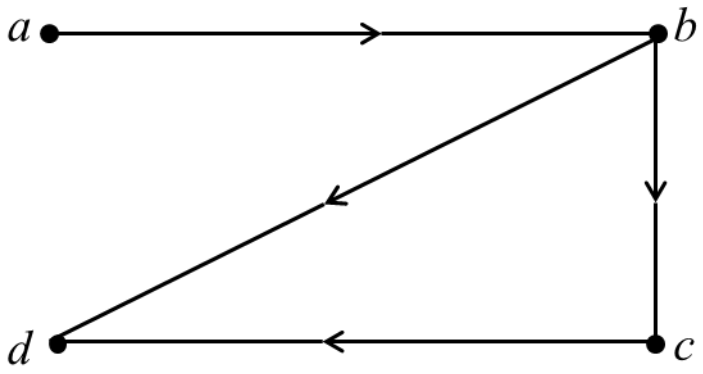


Рисунок 9.4. Орграф для примера 9.2.

Решение. Прежде всего, запишем матрицу смежности орграфа:

$$M = \begin{matrix} a \\ b \\ c \\ d \end{matrix}\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

Квадрат матрицы M равен

$$M^2 = M \times M = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Заметим, что матрица M^2 указывает на пути длиной 2 между вершинами орграфа G, а именно: abc, abd, и bdc.

Дальнейшие вычисления приводят к третьей и четвертой степеням матрицы M:

$$M^3 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, M^4 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Видим, что в орграфе отсутствуют пути длиной 4 и более.

Матрица достижимости орграфа G указывает на наличие путей между вершинами:

$$M^* = \text{M или } M^2 \text{ или } M^3 = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

Отметим, например, что цифра 1 в верхнем правом углу матрицы M^*

появляется из матрицы M^2 и соответствует пути *abd*. Для больших орграфов вычисление матрицы M^* с помощью возведения M в бо́льшую степень утомительно и неэффективно. Более удобный путь определения M^* дает алгоритм Уоршелла. Этот алгоритм вычисляет матрицу достижимости $W = M^*$ ориентированного графа $G = (V, E)$ с матрицей смежности M.

```
begin
  W:= M;
       for k = 1 to n do
          for i = 1 to n do
                   for j = 1 to n do
                            W(i, j) = W(i, j) или (W(i, k) и W(k, j))
end
```

Для лучшего понимания работы алгоритма Уоршелла желательно с его помощью вручную вычислить матрицу достижимости какого-нибудь орграфа.

9.3. Кратчайший путь

Рассмотрим задачу поиска кратчайшего пути, связывающего пару вершин в нагруженном орграфе. Термин «кратчайший путь» вполне уместен, поскольку довольно часто веса в орграфе – это расстояния между пунктами. Типичная ситуация, которая моделируется нагруженным орграфом, – это транспортная, или коммуникационная сеть.

Рассмотрим нагруженный граф, изображенный на рис. 9.5. Он может представлять, например, длины дорог в километрах между шестью деревнями. Поскольку количество вершин в этом графе невелико, то перебрать все возможные пути между любой парой заданных вершин нам вполне по силам. При этом, естественно, мы найдем наиболее короткий путь, соединяющий соответствующие деревни. В реальной задаче число вершин, как правило, настолько велико, что такой упрощенный подход к поиску кратчайшего пути слишком неэффективен.

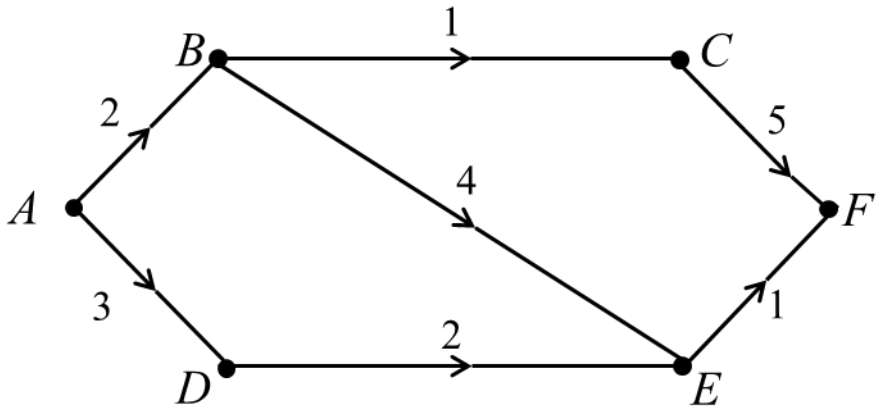


Рисунок 9.5. Нагруженный граф для примера о нахождении кратчайшего genb.

Существует множество алгоритмов поиска кратчайшего пути. Мы познакомимся с *алгоритмом Дейкстры.* Перед формальным изложением алгоритма мы опишем его действия и проиллюстрируем работу алгоритма на нашем примере. Допустим, что нужно найти кратчайший путь от вершины A к любой другой вершине орграфа (рис. 9.5). *Кратчайший путь* – это путь минимального общего веса, соединяющий выбранные вершины. Общий вес кратчайшего пути, ведущего из вершины u в вершину v, называют расстоянием от u до v. Определим весовую матрицу w, чьи элементы $w(u, v)$ задаются формулой:

$$w(u,v) = \begin{cases} 0, \text{ если } u = v, \\ \infty, \text{ если } u \text{ и } v \text{ не соединены дугой}, \\ d, \text{ если } uv - \text{дуга веса } d. \end{cases}$$

Для нашего графа весовая матрица имеет следующий вид:

$$\mathrm{W} = \begin{matrix} A \\ B \\ C \\ D \\ E \\ F \end{matrix} \begin{bmatrix} 0 & 2 & \infty & 3 & \infty & \infty \\ \infty & 0 & 1 & \infty & 4 & \infty \\ \infty & \infty & 0 & \infty & \infty & 5 \\ \infty & \infty & \infty & 0 & 2 & \infty \\ \infty & \infty & \infty & \infty & 0 & 1 \\ \infty & \infty & \infty & \infty & \infty & 0 \end{bmatrix}.$$

При работе алгоритма каждой вершине v орграфа присваивается число $d[v]$, равное расстоянию от вершины A до v. Перед началом работы $d[v]$ совпадает с весом дуги (A, v), если такая существует, или равно ∞ в противном случае. Мы

будем проходить вершины орграфа и уточнять значения $d[v]$.

На каждом шаге алгоритма отмечается одна вершина u, до которой уже найден кратчайший путь от A и расстояние $d[u]$ до нее. Далее полученное значение $d[u]$ отмеченной вершины не изменяется. Для оставшихся неотмеченных вершин v число $d[v]$ будет меняться с учетом того, что искомый кратчайший путь от них до A будет проходить через последнюю отмеченную вершину u. Алгоритм завершится тогда, когда все возможные вершины будут отмечены и получат свои окончательные значения $d[v]$.

Для каждого шага алгоритма, описанного ниже, в соответствующую строку таблицы заносится отмеченная вершина, текущие значения $d[v]$ и оставшиеся неотмеченные вершины (в закрашенных ячейках полужирным шрифтом выделено наименьшее из значений $d[v]$ среди неотмеченных вершин). Соответствующую вершину следует отметить на следующем шаге алгоритма.

Таблица 9.2. Действие алгоритма Дейкстры

Шаг	**Отмеченные вершины**	**Расстояния до вершины**						**Неотмеченные вершины**
		A	**B**	**C**	**D**	**E**	**F**	
0	A	0	**2**	∞	3	∞	∞	B,C,D,E,F
1	B	0	2	3	**3**	6	∞	C,D,E,F
2	D	0	2	**3**	3	5	∞	C,E,F
3	C	0	2	3	3	**5**	8	E,F
4	E	0	2	3	3	5	**6**	F
5	F	0	2	3	3	5	6	

Шаг 0. Поскольку нас интересует кратчайший путь от вершины A, мы ее отмечаем, используем первую строку весовой матрицы w для определения начальных значений $d[v]$. Таким образом получается первая строка таблицы. Наименьшее число из всех $d[v]$ для неотмеченных вершин – это $d[B] = 2$.

Шаг 1. Отмечаем вершину B, т. к. она является ближайшей к A. Вычисляем длины путей, ведущих от A к неотмеченным вершинам через вершину B. Если новые значения $d[v]$ оказываются меньше старых, то меняем последние на новые. При этом проходе цикла путь ABC имеет вес 3, а путь ABE – 6, в то время как старые расстояния до этих вершин от A были ∞. Следовательно, заполняя вторую строку таблицы, заменим $d[C]$ на 3 и $d[E]$ на 6.

Шаг 2. Из оставшихся неотмеченными вершины C и D находятся ближе всех

к *A*. Отметить можно любую из них. Возьмем вершину *D*. Так как длина пути *ADE* равна 5, текущее значение *d*[*E*] следует уменьшить до 5. Теперь можно заполнить третью строчку таблицы. Наименьшее значение *d*[*v*] среди неотмеченных к этому моменту вершин оказывается у вершины *C*.

Шаг 3. Отмечаем вершину *C* и подправляем значения *d*[*v*]. Теперь можно дойти и до вершины *F*, следуя путем *ABCF*. Его длина равна 8, значит и *d*[*F*] равен 8. К этому моменту остались неотмеченными две вершины: *E* и *F*.

Шаг 4. Мы отмечаем вершину *E*, что позволяет нам уменьшить величину *d*[*F*] с 8 до 6.

Шаг 5. Отмечаем вершину *F*.

Алгоритм Дейкстры

Пусть (*V*, *E*) – нагруженный граф, и *A* – его вершина. Алгоритм выбирает кратчайший путь от вершины *A* до любой вершины *v* и присваивает его длину переменной *d*[*v*]. Для вершин *u* и *v* через *w*(*u*, *v*) мы обозначаем вес дуги *uv*, а в списке PATHTO(*v*) перечисляются вершины кратчайшего пути от *A* до *v*.

```
begin
  for каждой v∈V do
        begin
           d[v]:= w(A,v);
           PATHTO(u):= A;
        end
  Отметить вершину A;
  while остаются неотмеченные вершины do
        begin
           u:= неотмеченную вершину с минимальным расстоянием от A;
           Отметить вершину и;
           for каждой неотмеченной вершины v с условием uv∈E do
                    begin
                             d':= d[u] + w(u,v)
                             if d' < d[v] then
                                      begin
                                               d[v]:= d';
                                               PATHTO(v):= PATHTO(u), v,
                                      end
                    end
        end
end
```

9.4. Коммуникационные сети

Успешной моделью компьютерной сети может служить ориентированный граф, чьи вершины (или *узлы)* представляют рабочие станции, а дуги – коммуникационные линии связи. Каждая дуга такого графа снабжена весом, обозначающим пропускную способность соответствующей линии. На рис. 9.6 показана, например, модель простой компьютерной сети из семи узлов, в которой веса дуг указывают на пропускную способность линий связи. Для коммуникаций между несмежными узлами необходимо определить маршруты передачи сообщений.

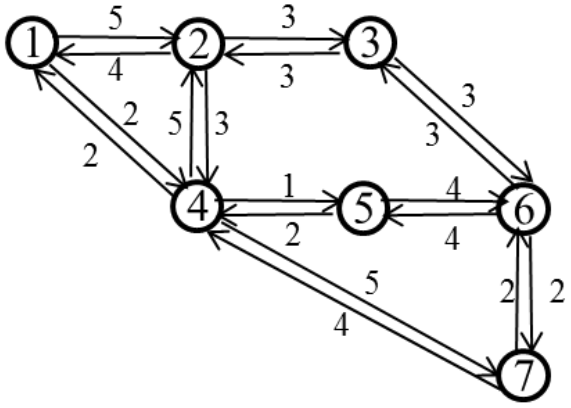


Рисунок 9.6. Модель компьютерной сети.

Процедура *статической маршрутизации* учитывает информацию о пропускной способности линий для определения фиксированного пути передач между узлами. В целях оптимизации таких путей в сети применяют алгоритм, подобный алгоритму Дейкстры. Однако при таком подходе могут возникать сбои в линиях или узлах сети. Задержки передач могут происходить в тех случаях, когда превышается пропускная способность линии.

Процедура *динамической маршрутизации* постоянно корректирует пропускную способность линий с учетом потребности. Отдельные узлы решают, когда и куда передавать новую информацию на основе используемых протоколов. Каждый узел поддерживает свою таблицу путей, так что задача оптимизации передачи сообщений рассредоточена по всей сети. В каждом узле сети, изображенной на рис. 9.6, выполняется алгоритм Дейкстры для

определения ближайших путей к другим узлам и эта информация распространяется по дереву, в корне которого сам узел и находится. Например, для узла 1 соответствующее дерево показано на рис. 9.7.

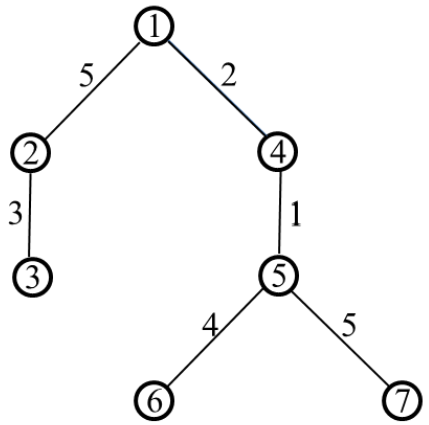


Рисунок 9.7. Дерево передачи информации для узла 1

Для передачи сообщений любому узлу требуется таблица, в которой указаны ближайшие соседи для передачи сообщения тому или иному адресату (таблицу ближайших соседей в этом контексте называют таблицей маршрутов). Такая таблица, относящаяся к узлу 1, приведена ниже (табл. 9.3). Выбор ближайшего соседа производится в каждом узле маршрута.

Таблица 9.3. Таблица маршрутов для узла 1 (рис. 9.7).

Адресат	**2**	**3**	**4**	**5**	**6**	**7**
Следующий узел	2	2	4	4	4	4

Пример 9.3. Используя алгоритм Дейкстры, найдите кратчайшие пути от узла 2 к любому другому по сети, изображенной на рис. 9.6. После этого изобразите дерево этих путей и заполните таблицу маршрутов для узла 2.

Решение. В результате действия алгоритма Дейкстры будет получена таблица 9.4. Кроме того, алгоритм Дейкстры определяет кратчайшие пути от вершины 2 к любой другой, например, PATHTO(6) = 2,3,6.

Таблица маршрутов для узла 2 представлена в таблице 9.5, а дерево кратчайших путей изображено на рисунке 9.8(а).

Таблица 9.4. Действие алгоритма Дейкстры для примера 9.3

	Отмеченные вершины	**Расстояния до вершины**							**Неотмеченные вершины**
		1	**2**	**3**	**4**	**5**	**6**	**7**	
0	2	4	0	3	3	∞	∞	∞	1,3,4,5,6,7
1	3	4	0	3	3	∞	6	∞	1,4,5,6,7
2	4	4	0	3	3	4	6	∞	1,5,6,7
3	5	4	0	3	3	4	6	9	1,6,7
4	1	4	0	3	3	4	6	9	6,7
5	6	4	0	3	3	4	6	8	7
6	7	4	0	3	3	4	6	8	

Таблица 9.5. Таблица маршрутов для примера 9.3.

Адресат	**1**	**3**	**4**	**5**	**6**	**7**
Следующий узел	1	3	4	4	3	3

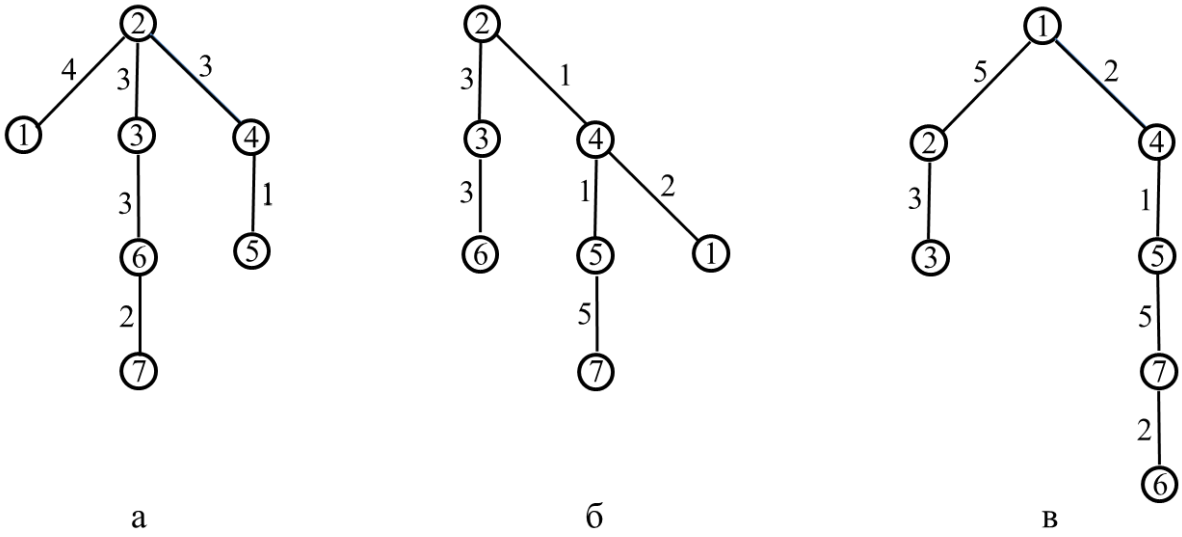


Рисунок 9.8. Дерево кратчайших путей для примеров 9.3 (а), 9.4 (б) и 9.5 (в).

Пример 9.4. Предположим, что временна́я задержка передачи от узла 2 к узлу 4 уменьшилась с 3 до 1. Как изменятся при этом дерево кратчайших путей и таблица маршрутов для узла 2?

Решение. Перезапустим алгоритм Дейкстры, установив временную задержку на линии 2–4, равную 1. Это изменение дает нам несколько новых деревьев кратчайших путей. Одно из них приведено на рис. 9.8(б).

Соответствующая таблица маршрутов – табл. 9.6.

Таблица 9.6. Таблица маршрутов для примера 8.4.

Адресат	**1**	**3**	**4**	**5**	**6**	**7**
Следующий узел	4	3	4	4	3	4

Пример 9.5. Какими будут дерево кратчайших путей и таблица маршрутов для узлов 1 и 2, если удалить линии между узлами 5 и 6?

Решение. Поскольку линия 5–6 не задействована при передаче информации от узла 2, то его дерево кратчайших путей и таблица маршрутов, найденные в примерах 9.3 и 9.4, останутся без изменений. Что касается узла 1, то мы можем ограничиться поиском кратчайшего пути от узла 1 к узлу 6. Алгоритм Дейкстры находит такой путь: PATHTO(6) = 1,4,5,7,6. Новое дерево кратчайших путей начерчено на рис. 9.8(в), а таблица 9.7 – соответствующая таблица маршрутов.

Таблица 9.7. Таблица маршрутов для примера 9.5.

Адресат	**2**	**3**	**4**	**5**	**6**	**7**
Следующий узел	2	2	4	4	4	4

9.5. Сети. Нахождение максимального потока в сетях

Сетью называется связный ориентированный граф $G = (V, E)$, каждой дуге которого e_{ij} ставится в соответствие неотрицательное действительное число $f(e_{ij})$, называемое ее *пропускной способностью*.

В дальнейшем будем считать, если не оговорено иное, что сеть содержит ровно один источник s и ровно один сток t и пропускная способность дуг не зависит от направления.

Введем для сети функцию $g(e)$, которая ставит в соответствие каждой дуге $e_{ij} \in E$ неотрицательное действительное число $g(e_{ij})$, называемое *потоком* через ребро e_{ij} таким образом, что поток через любую дугу не превосходит ее пропускной способности ($g(e_{ij}) \leq f(e_{ij})$ для любой дуги $e_{ij} \in E$) и суммарный поток, входящий в любую вершину, отличную от источника и стока, равен суммарному потоку, выходящему из нее (см. рис.9.9).

Дугу $e_{ij} \in E$ назовем *насыщенной*, если $g(e_{ij}) = f(e_{ij})$. Остальные дуги будем называть *ненасыщенными*. Если направление потока по дуге совпадает с ее ориентацией, то дуга называется *прямой*, если направление потока по дуге противоположно ее ориентации, то дуга называется *обратной*.

Сумма потоков через дуги, инцидентные источнику s, равна сумме потоков

через дуги, инцидентные стоку t. Эта сумма называется *величиной потока*. Величина максимального потока, который можно пропустить через заданную сеть, тесно связана с понятием *сечения*.

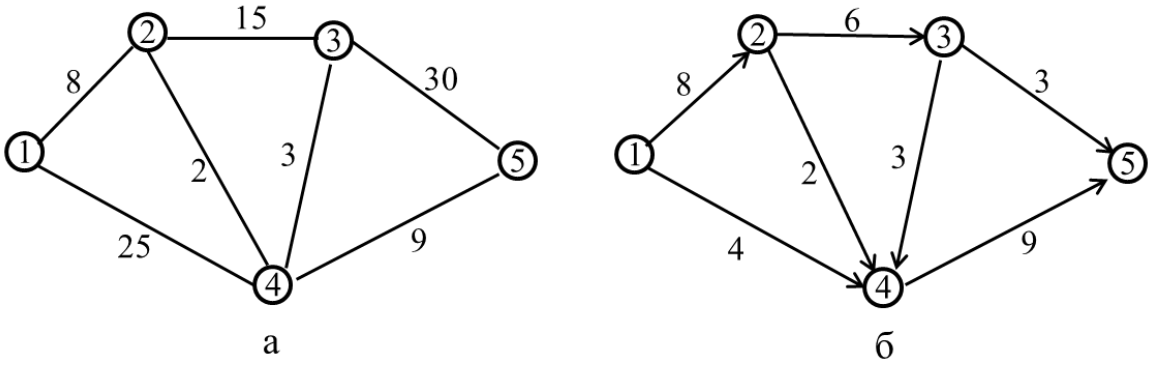


Рисунок 9.9. Графы, изображающие пропускную способность ребер $f(r_{ij})$ (а) и потоки через них $g(e_{ij})$ (б).

Сечение – множество X дуг ориентированного графа $G = (V, E)$, которое обладает тем свойством, что любой простой маршрут из s в t проходит через дугу, принадлежащую X. *Пропускной способностью сечения* называется сумма пропускных способностей принадлежащих ему дуг. *Минимальное сечение* – это сечение, обладающее минимальной пропускной способностью. Величина любого потока в сети не превышает пропускной способности любого сечения.

Теорема и алгоритм Форда–Фалкерсона

Во всякой сети величина максимального потока между вершинами s и t равна пропускной способности минимального сечения между этими вершинами.

Алгоритм Форда – Фалкерсона позволяет найти максимальный поток в сети. При его исполнении вершины графа помечаются следующим образом. Метка произвольной вершины v_j состоит из двух частей и имеет вид $(+v_i,\delta)$ или $(-v_i,\delta)$. Знак «+» означает, что поток допускает увеличение вдоль дуги (v_i,v_j) (дуга (v_i,v_j) прямая); знак «–» означает, что поток может быть уменьшен вдоль дуги (v_i,v_j) (дуга (v_i,v_j) обратная), δ задает максимальную величину дополнительного потока, который может протекать от s к t.

Будем помечать вершины сети от источника s Все помеченные вершины

образуют множество Y, все непомеченные вершины образуют множество Z. Источник $s \in Y$, его метка $(0,\infty)$. Знак ∞ означает, что ресурсы истока настолько велики, что не ограничивают поток.

Если вершина $v_i \in Y$ и $g(e_{ij}) \le f(e_{ij})$ и дуга (v_i,v_j) прямая ненасыщенная, то вершину v_j можно пометить и $v_j \in Y$. Метка вершины v_j – $(+v_i,\delta_j)$, где $\delta_j = \min\{\delta_j>0,\ f(e_{ij})-g(e_{ij})\}$.

Если вершина $v_i \in Y$ и дуга (v_i,v_j) обратная, то вершину v_j также можно пометить и $v_j \in Y$. В этом случае вершину v_j помечают как $(-v_i,\delta_j)$, где $\delta_j = \min\{\delta_j>0,\ f(e_{ij})+g(e_{ij})\}$.

Построение множества помеченных вершин Y индуктивное: новая вершина добавляется либо прямой ненасыщенной дугой, либо обратной дугой. После того, как к множеству Y нельзя добавить новых вершин, возможны два случая:

I. Если сток $t \in Y$ и $\delta_t>0$, то это означает, что поток от истока к стоку можно увеличить минимум на величину δ_t, увеличив потоки через дуги по маршруту помеченных вершин. Для прямых дуг величина потока станет $g(e_{ij})+\delta_t<f(e_{ij})$ и для обратных дуг величина потока будет $|g(e_{ij})-\delta_t|<f(e_{ij})$, причем если $g(e_{ij})-\delta_t<0$ то обратная дуга меняет направление и становится прямой. Для полученного нового потока снова проведем эту же процедуру.

II. Если сток $t \notin Y$, то множество непомеченных вершин непустое – $Z \ne \varnothing$. Так как граф связный, то из множества Y в множество Z идут прямые насыщенные дуги, которые образуют минимальное сечение между источником s и стоком t. Величина этого сечения равна максимальному потоку в сети $(G, f(e))$.

Пример 9.6. Требуется найти максимальный поток из вершины 1 в вершину 5 в сети, изображенной на рис. 9.9. Граф, который изображен слева, задает пропускные способности ребер. На графе справа задан поток, который необходимо либо улучшить, либо доказать, что он является максимальным.

Решение. По условию вершина 1 является источником, а вершина 5 –

стоком. Расставляем метки в графе, на котором указан поток.

1-я вершина (источник) всегда имеет метку (0, ∞) (второе число сколь угодно большое – это означает, что ресурсы источника не ограничивают поток). Далее, вершину 2 пометить пока нельзя (дуга (1, 2) насыщенная), но можно пометить вершину 4, т.к. дуга (1, 4) прямая ненасыщенная. Ее метка будет (+1, 21): +1 означает, что можно увеличить поток из вершины 1 в направлении имеющегося потока; 21 – это минимум из запасов δ_1 в вершине 1 (они не ограничены) и разности $f(e_{1,4})–g(e_{1,4})$. Вершину 5 пометить пока нельзя, но теперь можно пометить вершину 2: дуга (4, 2) обратная, и, учитывая возможный "разворот", величину потока можно увеличить на 4 единицы. Так как 4 меньше 21, получаем метку (–4, 4). Теперь можно пометить вершину 3 меткой (+2, 4) и вершину 5 меткой (+3, 4).

Таким образом, сток принадлежит множеству помеченных вершин Y и $\delta_t>0$. Значит, поток через сеть можно увеличить на 4 единицы. Прибавляем 4 единицы к прямым дугам и вычитаем из обратной с учетом возможного "разворота" обратной дуги. Расставленные метки и значения новых потоков через дуги изображены на рис. 9.10.

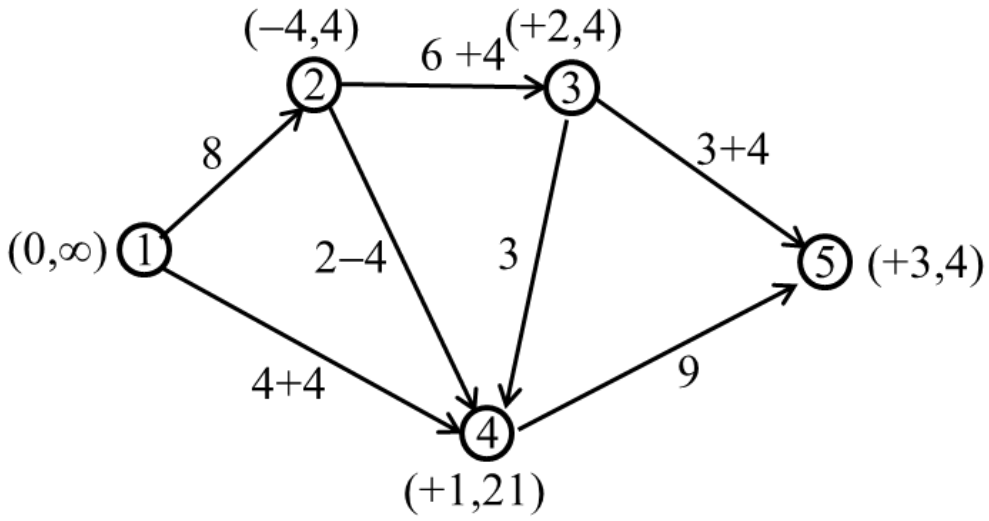


Рисунок 9.10. Увеличенные потоки через дуги после первой расстановки меток.

Повторно расставляем метки (сейчас имеем право пометить вершины

1,4,3,5) и видим, что новый поток также не максимальный – его можно увеличить минимум на 6 единиц по маршруту 1–4–3–5 (см. рис. 9.11).

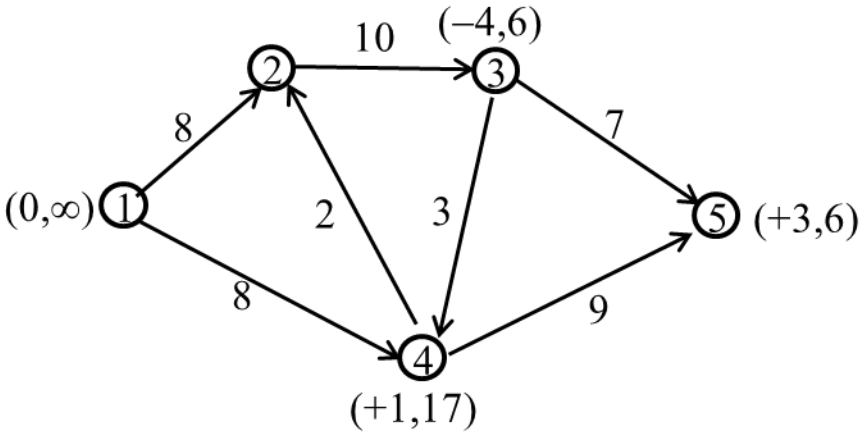


Рисунок 9.11. Результат повторной расстановки меток.

После увеличения потока на 6 единиц имеется возможность пометить только вершины 1 и 4 (рис. 9.12).

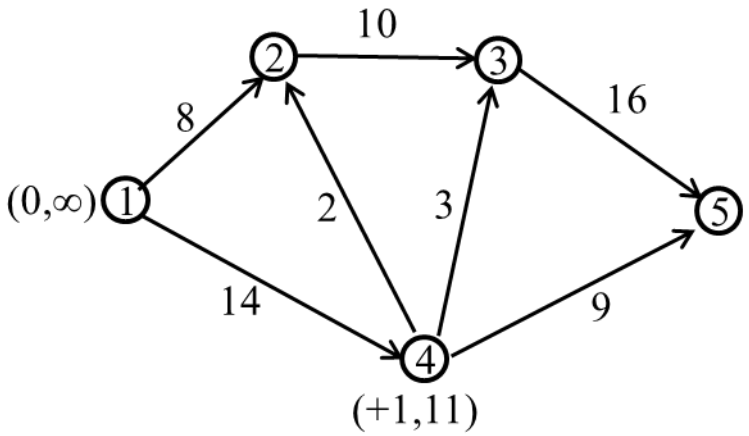


Рисунок 9.12. Сеть максимального потока.

Больше ничего пометить нельзя, так как из множества помеченных вершин $Y = \{1, 4\}$ в множество непомеченных вершин $Z = \{2, 3, 5\}$ идут только прямые, насыщенные дуги (1–2), (4–2), (4–3) и (4–5). Эти дуги образуют минимальное сечение, равное максимальной величине потока в 22 единицы.

Рекомендуемая литература

1. Хаггарти, Р. Дискретная математика для программистов / Р. Хаггарти. – М.: Техносфера, 2005. – 400 с.
2. Скуратович Е.А. Дискретная математика : учебное пособие / Е.А, Скуратович, В.А. Иванюкович. – Минск : Акад. Упр. При Президенте Респ. Беларусь, 2013. – 287 с.
3. Кузнецов, О.П. Дискретная математика для инженеров / О.П. Кузнецов. – М.: Лань, 2007. – 400 с.
4. Ерусалимский, Я.М. Дискретная математика: теория, задачи, приложения/ Я.М. Ерусалимский –: М.: Вузовская книга, 2002.– 268 с.
5. Нефедов, В. Н. Курс дискретной математики: учеб. пособие / В.Н. Нефедов, В.А. Осипов. – М.: Изд-во МАИ, 1992. – 264.
6. Оре, О. Теория графов / О. Оре. – М.: Наука, 1980. – 336 с.
7. Ерош, И.Л. Дискретная математика: учеб. пособие / И.Л. Ерош, М.Б. Сергеев, Н.В. Соловьев – СПб: ГУАП, 2005.– 144 с.
8. Судоплатов, С. В. Дискретная математика: учеб. / С.В. Судоплатов, Е.В. Овчиникова – М.: ИНФРА-М, 2007. – 256 с.
9. Виленкин, Н.Я. Популярная комбинаторика / Н.Я. Виленкин. – М.: Наука, 1975. – 208 с.
10. Риордан, Дж. Введение в комбинаторный анализ: [пер. с англ.] / Дж. Риордан. - М.: Изд-во иностр. лит., 1963. – 287 с.
11. Холл, М. Комбинаторика: [пер. с англ.] / М. Холл. – М.: Мир, 1970. – 424 с.
12. Капитонова, Ю. Лекции по дискретной математике / Ю. Капитонова – СПб.: BHV, 2004. – 624 с.
13. Новиков, Ф. А. Дискретная математика для программистов / Ф. А. Новиков – СПб: Питер, 2000. – 304 с.