МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Механико-математический факультет

Н. Т. Когабаев

ДИСКРЕТНАЯ МАТЕМАТИКА И ТЕОРИЯ АЛГОРИТМОВ

Учебное пособие

Новосибирск
2023

УДК 510.5+519.1 (075)
ББК В127я73-1
К 570

Рецензент
канд. физ.-мат. наук *С. С. Оспичев*

Когабаев, Н. Т.
К 570 Дискретная математика и теория алгоритмов : учеб. пособие / Н. Т. Когабаев ; Новосиб. гос. ун-т. — Новосибирск : ИПЦ НГУ, 2023. — 126 с.

ISBN 978-5-4437-1324-3

В настоящем учебном пособии изложены математические основы теории алгоритмов. Пособие отражает содержание лекций основного курса «Дискретная математика и теория алгоритмов» для студентов 1-го курса механико-математического факультета НГУ и охватывает материал из нескольких областей математики, так или иначе связанных с понятием алгоритма: алгоритмы на графах и их временна́я сложность, теория автоматов и регулярных языков, формальные грамматики, машины Тьюринга и частично рекурсивные функции, классическая теория вычислимости.

Предназначено для студентов 1-го курса механико-математического факультета НГУ, изучающих курс «Дискретная математика и теория алгоритмов», а также для всех желающих познакомиться с основами упомянутых в пособии математических теорий.

УДК 510.5+519.1 (075)
ББК В127я73-1

ISBN 978-5-4437-1324-3

Оглавление

Глава I

Предварительные сведения

§ 1. Некоторые аксиомы теории множеств

Все объекты, изучаемые в данном курсе, являются множествами. Множества — это символы, алфавиты и языки, это числа, кортежи и последовательности, а также предикаты, функции и операторы. Даже автоматы, машины и алгоритмы, изучению которых посвящён настоящий курс, являются множествами.

Для работы с множествами и формализации определённых понятий нам потребуются некоторые аксиомы *теории множеств Цермело — Френкеля* ZF. Теория ZF является формальной (синтаксической) теорией в языке с одним символом двухместного предиката $\in$ и символом равенства $\approx$. Однако мы будем формулировать понятия и аксиомы данной теории на естественном (общематематическом) языке. Подобная «нестрогость» не должна пугать читателя, поскольку при желании все формулировки можно «перевести» на формальный язык ZF, но в рамках данного курса в этом нет необходимости. Для более глубокого и подробного ознакомления с системой ZF можно порекомендовать книги [3, 4].

Понятия *множества* и *отношения принадлежности* $\in$ являются неопределяемыми через другие математические объекты. Неформально множество — это некоторая совокупность объектов A, отношение $x \in A$ означает, что объект x является элементом совокупности A. Мы также будем использовать термины *семейство* и *совокупность* для описания некоторых множеств. Часто множества задают перечислением всех его элементов, используя запись вида $A = \{x, y, \dots\}$.

Определение. Говорят, что множество A является *подмножеством* множества B, и пишут $A \subseteq B$, если $\forall x(x \in A \to x \in B)$. Другими словами, $A \subseteq B$, если каждый элемент множества A является элементом множества B.

Равенство двух множеств A и B определяется следующей аксиомой.

Аксиома экстенсиональности: *$A = B$ тогда и только тогда, когда справедливо $\forall x(x \in A \leftrightarrow x \in B)$. Таким образом, множества A и B равны, если $A \subseteq B$ и $B \subseteq A$.*

Следующая естественная аксиома постулирует существование наименьшего по включению множества.

Аксиома пустого множества: *существует пустое множество $\varnothing$, т. е. множество, не содержащее ни одного элемента.*

Следующие четыре аксиомы позволяют из одних множеств строить другие, более сложные по своей структуре.

Аксиома пары: *если A и B — множества, то существует неупорядоченная пара $\{A, B\}$, составленная из этих множеств.*

Аксиома суммы: *если A — множество, то существует множество $\cup A = \{x \mid x \in y$ для некоторого $y \in A\}$, которое называется объединением множества A.*

Аксиома степени: *если A — множество, то существует множество $P(A) = \{B \mid B \subseteq A\}$ всех подмножеств множества A.*

Аксиома выделения: *если A — множество, а $\Phi(x)$ — некоторое условие на множество x, то существует множество $\{x \in A \mid \Phi(x)\}$.*

Пример. Пусть множество $A = \{\{1,3,4\},\{3,6\},\{4,6\}\}$. Тогда его объединением будет множество $\cup A = \{1,3,4\} \cup \{3,6\} \cup \{4,6\} = \{1,3,4,6\}$.

Множество всех подмножеств множества $B = \{0,1,2\}$ состоит из восьми элементов, т. е. $P(B) = \{\varnothing, \{0\}, \{1\}, \{2\}, \{0,1\}, \{0,2\}, \{1,2\}, \{0,1,2\}\}$.

Из аксиомы пары и аксиомы суммы следует, что если x — множество, то $\{x\}$ — множество, а значит, и $x \cup \{x\}$ тоже является множеством, которое мы будем обозначать через $x+1$.

Из аксиомы пустого множества с помощью введённой операции $x+1$ получаем существование множеств вида $\varnothing$, $\{\varnothing\}$, $\{\varnothing, \{\varnothing\}\}$, $\{\varnothing, \{\varnothing\}, \{\varnothing, \{\varnothing\}\}\}$ и т. д. (каждое следующее состоит из всех предыдущих), эти множества мы будем называть *натуральными числами* и обозначать их соответственно через $0, 1, 2, 3$ и т. д.

Заметим, что, используя только те пять аксиом «существования», которые сформулированы выше, можно получить лишь конечные множества. В частности, из этих пяти аксиом невозможно вывести, что «совокупность» всех натуральных чисел образует множество. Для разрешения этого вопроса вводится следующая аксиома.

Аксиома бесконечности: *существует множество A такое, что $\varnothing \in A$ и если $x \in A$, то $x \cup \{x\} \in A$.*

Множество A, существование которого постулируется аксиомой бесконечности, обязано содержать все натуральные числа и, следовательно, бесконечно. Более того, из аксиомы бесконечности следует, что существует множество

$$\omega = \{0, 1, 2, 3, \ldots\},$$

состоящее из всех натуральных чисел. Для обозначения множества всех натуральных чисел также используют символ $\mathbb{N}$.

Теперь, располагая каноническим бесконечным множеством ω, можно строить другие бесконечные множества. В следующем определении вводятся стандартные теоретико-множественные операции объединения, пересечения, разности и дополнения (до некоторого множества).

Определение. Если A и B — множества, то их *объединением* называется множество $A \cup B = \{x \mid x \in A$ или $x \in B\}$, *пересечением* — множество $A \cap B = \{x \mid x \in A$ и $x \in B\}$, *разностью* — множество $A \setminus B = \{x \mid x \in A$ и $x \notin B\}$.

Если рассматриваемые множества являются подмножествами некоторого фиксированного множества U, то можно говорить о *дополнении* $\overline{A} = U \setminus A$ множества A (до множества U).

Объединением семейства множеств A называется множество $\cup A = \{x \mid \exists B \in A\ (x \in B)\}$, а *пересечением* непустого семейства множеств A — множество $\cap A = \{x \mid \forall B \in A (x \in B)\}$.

Из перечисленных выше аксиом следует, что, применяя эти операции к множествам, мы снова получаем множества. Например, если A, B — множества, то в силу аксиомы пары существует неупорядоченная пара $\{A, B\}$, а в силу аксиомы суммы — объединение $A \cup B = \cup\{A, B\}$. Затем, используя аксиому выделения, заключаем, что существует пересечение $A \cap B = \{x \in A \cup B \mid x \in A \text{ и } x \in B\}$.

Определение. *Разбиением* множества A называется такое множество $X \subseteq P(A)$, что $\cup X = A$ и для любых различных $B, C \in X$ справедливо $B \cap C = \varnothing$.

Аксиома пары постулирует существование множества $\{a, b\}$. Однако порядок расположения элементов в паре формально никак не задаётся, поскольку $\{a, b\} = \{b, a\}$. Более того, если $a = b$, то пара $\{a, b\}$ превращается в одноэлементное множество $\{a\}$. Чтобы всё-таки упорядочить элементы пары, вводится следующее определение.

Определение. *Упорядоченной парой* элементов a и b называется множество $\langle a, b\rangle = \{\{a\}, \{a, b\}\}$. В упорядоченной паре мы задаём строгий порядок расположения элементов: a — первый, b — второй. Следует различать $\langle a, b\rangle \neq \{a, b\}$!

Предложение 1. *Для любых элементов a, b, c, d имеет место $\langle a, b\rangle = \langle c, d\rangle$ тогда и только тогда, когда $a = c$ и $b = d$.*

Доказательство. Предлагается читателю в качестве упражнения. □

Определение. Пусть $n \in \omega, n \geqslant 1$. *Упорядоченная n-ка (кортеж длины n)* определяется по индукции: $\langle a_1\rangle = a_1$, $\langle a_1, \ldots, a_{n-1}, a_n\rangle = \langle\langle a_1, \ldots, a_{n-1}\rangle, a_n\rangle$.

Пустое множество $\varnothing$ по определению называем *кортежем длины* 0.

Следствие 2. *$\langle a_1, \ldots, a_n\rangle = \langle b_1, \ldots, b_n\rangle$ тогда и только тогда, когда имеет место $a_1 = b_1, \ldots, a_n = b_n$.*

Доказательство. Следует из предыдущего предложения по индукции. □

Определение. *Декартовым произведением* множеств $A_1, \ldots, A_n$ называется множество

$$A_1 \times \ldots \times A_n = \{\langle a_1, \ldots, a_n\rangle \mid a_1 \in A_1, \ldots, a_n \in A_n\}.$$

n-й декартовой степенью множества A называется множество $A^n = \underbrace{A \times \ldots \times A}_{n}$.

При $n = 0$ по определению полагаем $A^0 = \{\varnothing\}$.

Определение. Любое подмножество $R \subseteq A_1 \times \ldots \times A_n$ называется *отношением (предикатом)* на множествах $A_1, \ldots, A_n$. Если $\langle x_1, \ldots, x_n\rangle \in R$, то говорят, что *предикат R истинен на элементах $x_1, \ldots, x_n$*, и пишут $R(x_1, \ldots, x_n)$, иначе говорят, что *предикат R ложен на элементах $x_1, \ldots, x_n$*, и пишут $\neg R(x_1, \ldots, x_n)$.

Любое подмножество $R \subseteq A^n$ называется *n-местным отношением (предикатом)* на множестве A. В частности, *бинарное отношение (предикат)* на множестве A — это любое подмножество $R \subseteq A^2$.

Определение. Бинарное отношение $R \subseteq A \times A$ на множестве A называется:
(а) *рефлексивным*, если для всех $a \in A$ справедливо $\langle a, a\rangle \in R$;
(б) *симметричным*, если для всех $a, b \in A$ из $\langle a, b\rangle \in R$ следует $\langle b, a\rangle \in R$;
(в) *транзитивным*, если для всех $a, b, c \in A$ из $\langle a, b\rangle \in R$ и $\langle b, c\rangle \in R$ следует $\langle a, c\rangle \in R$;
(г) *отношением эквивалентности*, если оно рефлексивно, симметрично и транзитивно.

Определение. Пусть R — отношение эквивалентности на множестве A. *Классом эквивалентности* элемента $a \in A$ по отношению R называют множество $a/R = \{b \in A \mid \langle a, b\rangle \in R\}$, состоящее из всех элементов A, эквивалентных a. Множество $A/R = \{a/R \mid a \in A\}$ всех классов эквивалентности называют *фактор-множеством A по отношению R*.

Предложение 3. *Для любого отношения эквивалентности R на множестве A классы эквивалентности по отношению R образуют разбиение множества A, и обратно, любое разбиение множества A определяет отношение эквивалентности R на A, для которого множества в разбиении являются классами эквивалентности.*

Доказательство. Предлагается читателю в качестве упражнения. □

Определение. *Композицией* отношений $R_1 \subseteq A \times B$ и $R_2 \subseteq B \times C$ называется отношение $R_1 \circ R_2 = \{\langle a, c\rangle \mid \exists b(\langle a, b\rangle \in R_1 \text{ и } \langle b, c\rangle \in R_2)\}$.

Определение. *Обратным отношением* к отношению $R \subseteq A \times B$ называется отношение $R^{-1} = \{\langle b, a\rangle \mid \langle a, b\rangle \in R\}$.

Определение. Отношение $f \subseteq A \times B$ называется *функцией (отображением)*, если для любого $a \in A$ существует не более одного $b \in B$ такого, что $\langle a, b\rangle \in f$.

Для данного $a \in A$, если существует $b \in B$ со свойством $\langle a, b\rangle \in f$, то говорят, что *значение $f(a)$ определено и равно b*, и пишут $f(a)\downarrow$, в противном случае говорят, что *значение $f(a)$ не определено*, и пишут $f(a)\uparrow$.

Областью определения f называется множество $\mathrm{dom}(f) = \{a \in A \mid f(a)\downarrow\}$.

Областью значений f называется множество $\mathrm{range}(f) = \{f(a) \mid a \in A, f(a)\downarrow\}$.

Определение. Запись $f : A \to B$ означает, что для некоторых множеств X, Y отношение $f \subseteq X \times Y$ является функцией такой, что $\mathrm{dom}(f) = A$ и $\mathrm{range}(f) \subseteq B$. При этом говорят, что f является *функцией из A в B*.

Определение. Говорят, что функция $f : A \to B$ является *инъективной*, и пишут $f : A \xrightarrow{1-1} B$, если для любого $b \in B$ существует не более одного $a \in A$ такого, что $f(a) = b$.

Определение. Говорят, что функция $f : A \to B$ является *сюръективной*, и пишут $f : A \xrightarrow[\text{на}]{} B$, если $\mathrm{range}(f) = B$.

Определение. Говорят, что функция $f : A \to B$ является *биективной*, и пишут $f : A \xrightarrow[\text{на}]{1-1} B$, если f одновременно инъективна и сюръективна.

Определение. Если $f \subseteq A \times B$, $g \subseteq B \times C$ — функции, то их *композиция* $f \circ g \subseteq A \times C$ определяется как композиция отношений. Легко видеть, что $f \circ g$ тоже является функцией.

Определение. Если $f : A \xrightarrow[\text{на}]{1-1} B$ — биективная функция, то *обратная функция* f^{-1} определяется как обратное отношение. Легко видеть, что f^{-1} является биекцией вида $f^{-1} : B \xrightarrow[\text{на}]{1-1} A$.

Определение. Функция вида $f : X \to A$, где $X \subseteq A^n$ называется *n-местной частичной функцией на A*.

Замечание. Заметим, что существуют только два вида 0-местных частичных функций — это либо нигде не определенная функция $f{=}\varnothing$, либо функция вида $f{=}\{\langle\varnothing, a\rangle\}$ для некоторого $a \in A$. Во втором случае мы будем называть функцию константой и отождествлять её с элементом a, т. е. $f = a$.

Упражнения

1. Пусть A, B, C — произвольные множества. Верно ли, что если $A \in B$ и $B \in C$, то $A \in C$?

2. Пусть $X_0 \supseteq X_1 \supseteq X_2 \supseteq \ldots \supseteq X_n \supseteq \ldots$ — счётная невозрастающая последовательность множеств. Доказать, что пересечение любой бесконечной подпоследовательности этих множеств совпадает с пересечением всей последовательности.

3. Доказать, что теоретико-множественную операцию $\setminus$ нельзя определить через операции $\cap$ и $\cup$.

4. Доказать, что теоретико-множественную операцию $\cup$ нельзя определить через операции $\cap$ и $\setminus$.

5. Доказать предложение 1 и следствие 2.

6. Используя аксиомы пары, суммы, степени и выделения, доказать, что для любых множеств A и B существует множество $A \times B$.

7. Доказать предложение 3.

8. Найти R^{-1}, $R \circ R$, $R \circ R^{-1}$ и $R^{-1} \circ R$ для отношения $R = \{\langle x, y\rangle \in \mathbb{R}^2 \mid 2x \geqslant 3y\}$.

9. Доказать, что отношение $R \subseteq A \times B$ является биекцией из A на B тогда и только тогда, когда $R \circ R^{-1} = \{\langle a, a\rangle \mid a \in A\}$ и $R^{-1} \circ R = \{\langle b, b\rangle \mid b \in B\}$.

§ 2. Время работы алгоритмов и сортировки чисел

Для краткой записи алгоритмов в данном параграфе и следующей главе мы будем использовать *псевдокод*, который во многом похож на язык программирования. Представленные в виде псевдокода алгоритмы, как правило, будут интуитивно понятны, но при необходимости мы будем приводить пояснения к ним.

Время работы алгоритма измеряется в количестве элементарных операций, которые необходимо выполнить. Под элементарными операциями подразумеваются арифметические (сложение, вычитание, умножение, деление), операции перемещения данных (загрузка, занесение в память, копирование) и управляющие (условное и безусловное ветвление, вызов подпрограммы и возврат из неё). При этом мы считаем, что для исполнения каждой команды псевдокода требуется фиксированное количество элементарных операций.

Время работы алгоритма описывается как функция, зависящая от размера входных данных алгоритма. Важно уделять внимание определению времени работы в наихудшем случае, т.е. максимальному времени работы из всех входных данных размера n. Для этого мы используем O-символику, удобную для записи асимптотической верхней границы для функции времени работы.

Определение. Пусть даны две функции $f(n)$ и $g(n)$ от натурального аргумента n, значениями которых являются положительные действительные числа. Будем говорить, что $f(n)$ *растёт не быстрее* $g(n)$, и писать $f(n) = O(g(n))$, если существуют константа $c > 0$ и натуральное n_0 такие, что $f(n) \leqslant c \cdot g(n)$ для всех $n \geqslant n_0$.

Таким образом, фраза «*время работы алгоритма составляет* $O(g(n))$» означает, что на входных данных достаточно большого размера n время работы алгоритма не превосходит $c \cdot g(n)$, где c — положительная константа.

Разберём вычисление времени работы алгоритма на примере *задачи сортировки* конечных наборов чисел. Входными данными в этой задаче является упорядоченный набор $\langle a_1, a_2, \ldots, a_n\rangle$ действительных чисел, при этом длина набора n считается размером входных данных. Необходимо путём перестановки чисел преобразовать входной набор в набор $\langle a'_1, a'_2, \ldots, a'_n\rangle$ таким образом, чтобы выполнялось условие $a'_1 \leqslant a'_2 \leqslant \cdots \leqslant a'_n$.

Сортировка вставкой (insertion sort) — это простой и естественный алгоритм, решающий указанную задачу. Перейдём к описанию данного способа сортировки.

Описание алгоритма

На вход алгоритма подаётся массив $a[1], \ldots, a[n]$ длины $length[a] = n$, состоящий из действительных чисел. Алгоритм последовательно выбирает *ключевой* элемент $a[j]$ из массива и, предполагая, что элементы $a[1], \ldots, a[j-1]$ уже отсортированы в порядке неубывания, помещает $a[j]$ в нужное место среди элементов $a[1], \ldots, a[j-1]$. Чтобы определить, куда нужно поместить ключевой элемент, алгоритм сравнивает $a[j]$ с элементами массива $a[1], \ldots, a[j-1]$, двигаясь по нему справа налево. По окончании работы алгоритма входной массив содержит уже отсортированные числа.

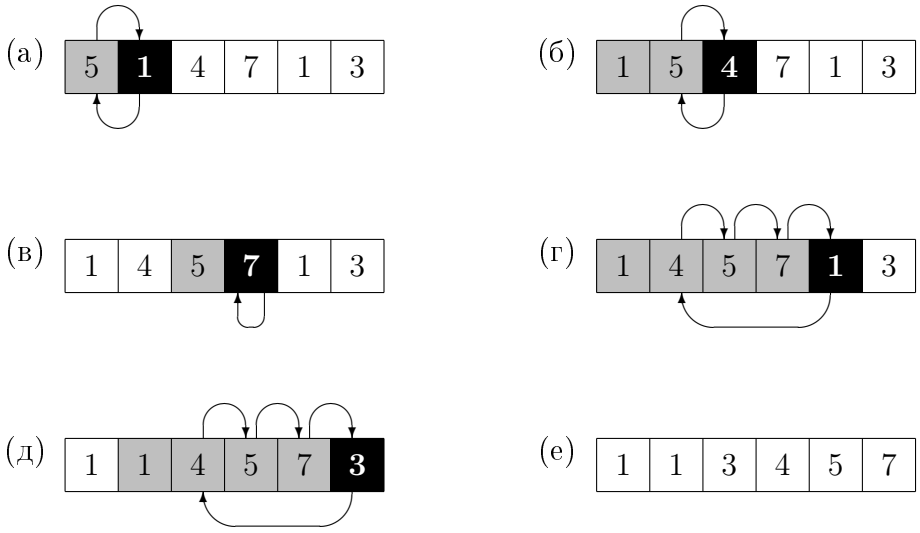
Ниже приведена процедура INSERTION_SORT, реализующая данный алгоритм.

```
INSERTION_SORT(a)
{Вход: массив a[1], ..., a[n].}
{Выход: отсортированный массив a[1], ..., a[n].}
01  for j ← 2 to length[a]
02      do key ← a[j]
03         i ← j − 1
04         while i > 0 и a[i] > key
05               do a[i + 1] ← a[i]
06                  i ← i − 1
07         a[i + 1] ← key
```

Процедура INSERTION_SORT работает следующим образом. В строках 01–07 организуется цикл **for**, на каждой итерации которого в строке 02 выбирается ключевой элемент $a[j]$. В строках 03–07 происходит вставка элемента $a[j]$ в отсортированный набор $a[1], \ldots, a[j-1]$. Для этого в строке 03 инициализируется убывающий счётчик i с начальным значением $i = j - 1$. Цикл **while** в строках 04–06 последовательно уменьшает i до тех пор, пока i не станет равным нулю либо пока не выполнится условие $a[i] \leqslant a[j]$. В строке 07 после выхода из цикла **while** ключевой элемент помещается на $(i+1)$-е место в массиве. Если $i > 0$ и $a[i] > a[j]$ в теле цикла **while**, то в строке 05 элемент $a[i]$ переставляется с i-го на $(i+1)$-е место в массиве, а счётчик i в строке 06 уменьшается на единицу.

Пример. Продемонстрируем, как работает процедура INSERTION_SORT на входном массиве $\langle 5, 1, 4, 7, 1, 3\rangle$. Элементы массива обозначены квадратами. Части (а)–(д) это-

го рисунка соответствуют итерациям цикла **for** в строках 01–07. На каждой итерации чёрный квадрат содержит значение ключа, которое сравнивается со значениями серых квадратов, расположенных слева от него (строка 04). Стрелками указаны перемещения элементов массива на одну позицию вправо (строка 05) и перемещения ключа (строка 07). Часть (е) показывает конечное состояние сортируемого массива.



Время работы алгоритма

Обозначим через $T(n)$ время работы процедуры INSERTION_SORT на входе размера n. Значение $T(n)$ зависит от времени исполнения каждой строки псевдокода и количества их повторений. Для $1 \leqslant i \leqslant 7$ обозначим время исполнения i-й строки через c_i, где c_i — константа. Для $2 \leqslant j \leqslant n$ обозначим через t_j количество проверок условия в заголовке цикла **while** в строке 04. Отметим, что при нормальном завершении циклов **for** или **while** (т.е. когда перестаёт выполняться условие, заданное в заголовке цикла) условие проверяется на один раз больше, чем выполняется тело цикла. Поэтому количество исполнений каждой из семи строк псевдокода будет следующим:

INSERTION_SORT(a)	время	количество раз
01 **for** $j \leftarrow 2$ **to** $length[a]$	c_1	n
02 **do** $key \leftarrow a[j]$	c_2	$n-1$
03 $i \leftarrow j-1$	c_3	$n-1$
04 **while** $i > 0$ и $a[i] > key$	c_4	$\sum_{j=2}^{n} t_j$
05 **do** $a[i+1] \leftarrow a[i]$	c_5	$\sum_{j=2}^{n}(t_j-1)$
06 $i \leftarrow i-1$	c_6	$\sum_{j=2}^{n}(t_j-1)$
07 $a[i+1] \leftarrow key$	c_7	$n-1$

Чтобы вычислить значение $T(n)$, необходимо просуммировать произведения значений, стоящих в столбцах *время* и *количество раз*:

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n}(t_j - 1) + c_6 \sum_{j=2}^{n}(t_j - 1) + c_7(n-1).$$

Даже если размер массива фиксирован, значения t_j, $2 \leqslant j \leqslant n$, зависят от степени упорядоченности входного массива. Самый благоприятный случай — это когда все

элементы $a[1], \ldots, a[n]$ уже находятся в порядке неубывания. Тогда $t_j = 1$ для всех j и время работы алгоритма вычисляется так:

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1) = \\ = (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7),$$

т.е. время работы в наилучшем случае является линейной функцией от n.

Наихудший случай — это когда элементы входного массива расположены в порядке убывания. Тогда каждый ключевой элемент $a[j]$ необходимо сравнивать со всеми элементами уже отсортированного набора $a[1], \ldots, a[j-1]$, и значит, $t_j = j$ для всех j. С учётом того что

$$\sum_{j=2}^{n} j = \frac{(n+2)(n-1)}{2}, \quad \sum_{j=2}^{n}(j-1) = \frac{n(n-1)}{2},$$

получаем, что для $n \geqslant 1$ время работы в наихудшем случае составляет

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4\frac{(n+2)(n-1)}{2} + c_5\frac{n(n-1)}{2} + c_6\frac{n(n-1)}{2} + c_7(n-1) = \\ = \Big(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\Big)n^2 + \Big(c_1 + c_2 + c_3 + \frac{c_4}{2} - \frac{c_5}{2} - \frac{c_6}{2} + c_7\Big)n + (-c_2 - c_3 - c_4 - c_7) \leqslant \\ \leqslant \Big(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\Big)n^2 + \Big(c_1 + c_2 + c_3 + \frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2} + c_7\Big)n + (c_2 + c_3 + c_4 + c_7) \leqslant \\ \leqslant \Big(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\Big)n^2 + \Big(c_1 + c_2 + c_3 + \frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2} + c_7\Big)n^2 + (c_2 + c_3 + c_4 + c_7)n^2 = \\ = (c_1 + 2c_2 + 2c_3 + 2c_4 + c_5 + c_6 + 2c_7)n^2 = dn^2,$$

где константа $d > 0$ зависит от c_i. Поскольку мы уделяем основное внимание времени работы алгоритма в наихудшем случае, можно окончательно заключить, что время работы сортировки вставкой составляет $O(n^2)$, т.е. время квадратично.

В дальнейшем мы не будем находить явную формулу для функции времени работы алгоритма настолько подробно, насколько это было сделано выше для сортировки вставкой. Нас будет интересовать только асимптотическая верхняя оценка, поэтому использование O-символики заметно облегчает анализ времени работы алгоритмов. Как видно на примере сортировки вставкой, достаточно принимать во внимание только главный член в формуле (в нашем случае n^2), поскольку при больших n членами меньшего порядка можно пренебречь. Кроме того, постоянные множители при главном члене также можно игнорировать, так как при росте размера входных данных они менее важны, чем порядок роста функции времени работы алгоритма.

Сортировка слиянием (merge sort) — это алгоритм, который также решает задачу сортировки, но в общем случае работает быстрее, чем алгоритм сортировки вставкой. Сортировку слиянием также часто называют *алгоритмом фон Неймана*.

Описание алгоритма

Пусть дан массив $a[1], \ldots, a[n]$ действительных чисел. Сортировка слиянием работает $\lceil \log_2 n \rceil$ итераций, где $\lceil x \rceil$ обозначает наименьшее целое число z с условием $x \leqslant z$. Перед началом k-й итерации ($k = 1, 2, \ldots, \lceil \log_2 n \rceil$) имеется последовательность $a[1], \ldots, a[n]$ тех же чисел, разбитая на группы по 2^{k-1} элементов (группы могут быть неполными), и внутри каждой группы элементы упорядочены по неубыванию. Итерация состоит в *слиянии* пар соседних групп в группы по 2^k элементов, при этом

элементы новых групп упорядочиваются по неубыванию. После исполнения последней итерации останется одна-единственная группа из n элементов, упорядоченных в нужном порядке.

Операцию слияния мы опишем с помощью отдельной процедуры. Пусть соседние группы состоят из элементов $a[p] \leqslant \cdots \leqslant a[q]$ и $a[q{+}1] \leqslant \cdots \leqslant a[r]$ соответственно, где $p \leqslant q < r$. Скопируем эти группы в два массива $b[1] \leqslant \cdots \leqslant b[n_1]$ и $c[1] \leqslant \cdots \leqslant c[n_2]$ соответственно, где $n_1 = q{-}p{+}1$, $n_2 = r{-}q$. Новая группа $a[p] \leqslant \cdots \leqslant a[r]$ строится по следующим правилам. Из элементов $b[1]$ и $c[1]$ выбираем минимальный и копируем его в $a[p]$ (в случае равенства выбирается элемент из массива b). Если этот минимум достигается на $b[1]$, то затем выбираем минимальный из $b[2]$ и $c[1]$ и копируем его в $a[p{+}1]$ и т. д. В общем случае мы сравниваем наименьшие элементы $b[i]$ и $c[j]$ массивов b и c, которые ещё не скопированы в a, выбираем из них минимум и переносим его в новую группу. Эти действия повторяются до тех пор, пока не закончатся элементы одного из массивов b или c, после чего оставшиеся элементы другого массива дописываются в конец новой группы. Для упрощения написания псевдокода поместим в конец массивов b и c *сигнальные* элементы $b[n_1{+}1] = \infty$ и $c[n_2{+}1] = \infty$. Поскольку значение сигнального элемента больше любого значения из исходного массива, можно не проверять, закончились ли элементы в b или c. Тогда слияние групп достаточно продолжать до тех пор, пока сравниваемые элементы массивов не окажутся сигнальными.

Ниже приведена процедура MERGE, осуществляющая слияние двух групп:

```
MERGE(a, p, q, r)
{Вход: отсортированные массивы a[p], ..., a[q] и a[q+1], ..., a[r], числа p ⩽ q < r.}
{Выход: отсортированный массив a[p], ..., a[r].}
01  n1 ← q − p + 1
02  n2 ← r − q
03  Создаём массивы b[1], ..., b[n1+1] и c[1], ..., c[n2+1]
04  for i ← 1 to n1
05      do b[i] ← a[p+i−1]
06  for j ← 1 to n2
07      do c[j] ← a[q+j]
08  b[n1+1] ← ∞
09  c[n2+1] ← ∞
10  i ← 1
11  j ← 1
12  for k ← p to r
13      do if b[i] ⩽ c[j]
14            then a[k] ← b[i]
15                 i ← i + 1
16            else a[k] ← c[j]
17                 j ← j + 1
```

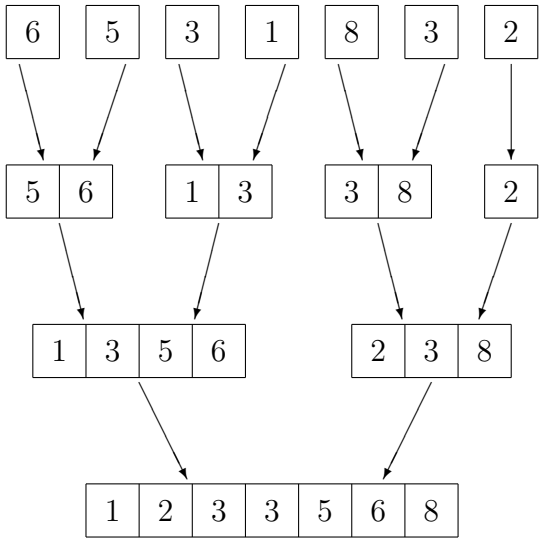
Теперь процедуру MERGE можно использовать в качестве подпрограммы в алгоритме сортировки слиянием. Основная процедура сортировки MERGE_SORT(a, p, r) применяет рекурсию, поэтому мы опишем её в общем случае для произвольного подмассива $a[p], \ldots, a[r]$ входного массива.

```
MERGE_SORT(a, p, r)
{Вход: массив a[p], ..., a[r], числа p, r.}
```

```
{Выход: отсортированный массив a[p], ..., a[r].}
01  if p < r
02    then q ← ⌊(p + r)/2⌋
03         MERGE_SORT(a, p, q)
04         MERGE_SORT(a, q + 1, r)
05         MERGE(a, p, q, r)
```

Процедура MERGE_SORT работает следующим образом. В строке 01 сравниваются индексы p и r. Если $p \geqslant r$, то в массиве $a[p], \ldots, a[r]$ содержится не более одного элемента, и значит, сортировка не требуется. Если же $p < r$, то в строке 02 процедура вычисляет разделяющий индекс $q = \lfloor (p + r)/2 \rfloor$, где $\lfloor x \rfloor$ обозначает наибольшее целое число z с условием $z \leqslant x$, и разбивает массив $a[p], \ldots, a[r]$ на два подмассива $a[p], \ldots, a[q]$ и $a[q{+}1], \ldots, a[r]$. В строках 03–04 полученные после разделения подмассивы сортируются по рекурсии, после чего в строке 05 происходит их слияние с помощью вызова подпрограммы MERGE.

Пример. На рисунке ниже проиллюстрирована работа процедуры MERGE_SORT на входном массиве $\langle 6, 5, 3, 1, 8, 3, 2 \rangle$ с индексами $p = 1$ и $r = 7$.



Время работы алгоритма

Чтобы отсортировать набор $a[1], \ldots, a[n]$ длины n, необходимо запустить процедуру MERGE_SORT$(a, 1, n)$. На k-й итерации ($k = 1, 2, \ldots, \lceil \log_2 n \rceil$) слияние двух групп из 2^{k-1} элементов требует время $O(2^k)$, и на k-й итерации происходит (в худшем случае) $n/2^k$ таких слияний. Следовательно, одна итерация занимает время $O(n)$, и общее время сортировки составляет $O(n \cdot \log_2 n)$.

Упражнения

1. Справедливы ли соотношения $2^{n+1} = O(2^n)$ и $2^{2n} = O(2^n)$?

2. *Сортировка выбором* (selection sort) производится следующим образом. Сначала среди $a[1], \ldots, a[n]$ определяется наименьший элемент, который переставляется на место $a[1]$. Затем среди $a[2], \ldots, a[n]$ производится поиск наименьшего элемента, который переставляется на место $a[2]$. Этот процесс продолжается $n - 1$ итерацию. Запишите псевдокод этого алгоритма. Определите время работы в наилучшем и наихудшем случаях и запишите его в O-символике.

3. Докажите, что число основных итераций алгоритма сортировки слиянием на массиве размера n равно $\lceil \log_2 n \rceil$. (Используйте индукцию по n.)

4. Докажите, что время работы процедуры MERGE(a, p, q, r) равно $O(r-p+1)$.

5. Перепишите процедуру MERGE так, чтобы в ней не использовались сигнальные элементы. Сигналом к остановке должен служить тот факт, что все элементы одного из массивов b или c уже скопированы обратно в массив a, после чего в массив a копируются элементы, оставшиеся в непустом массиве.

6. *Пузырьковая сортировка* основана на многократной перестановке соседних элементов массива, нарушающих порядок сортировки. Процедура пузырьковой сортировки имеет следующий псевдокод:

   BUBBLESORT(a)
   {Вход: массив $a[1], \ldots, a[n]$.}
   {Выход: отсортированный массив $a[1], \ldots, a[n]$.}
   01 **for** $i \leftarrow 1$ **to** $length[a]$
   02 **do for** $j \leftarrow length[a]$ **downto** $i + 1$
   03 **do if** $a[j] < a[j-1]$
   04 **then** Поменять местами $a[j] \leftrightarrow a[j-1]$

   Определите время работы процедуры BUBBLESORT(a) в наихудшем случае.

7. *Схема Горнера* позволяет вычислить значение полинома по заданным коэффициентам $a_0, a_1, \ldots, a_n$ и величине x с использованием следующего тождества:

$$\sum_{i=0}^{n} a_i x^i = a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + xa_n) \ldots)).$$

   Ниже приведена процедура вычисления значения полинома по схеме Горнера:

   HORNER(a, n, x)
   {Вход: массив коэффициентов $a[0], \ldots, a[n]$, число n и значение x.}
   {Выход: значение полинома $y = a[0] + a[1] \cdot x + \cdots + a[n] \cdot x^n$.}
   01 $y \leftarrow 0$
   02 $i \leftarrow n$
   03 **while** $i \geqslant 0$
   04 **do** $y \leftarrow a[i] + x \cdot y$
   05 $i \leftarrow i - 1$
   06 **return** y

   Считая число n размером входных данных, определите асимптотическое время работы процедуры HORNER(a, n, x).

Глава II

Алгоритмы для работы с графами

Графы представляют собой распространённые структуры в информатике, и алгоритмы для работы с графами очень важны. Имеются сотни вычислительных задач, сформулированных с использованием графов. В этой главе мы рассмотрим только некоторые из них и разберём алгоритмы для их решения. Все графы в данной главе считаются конечными. Время работы алгоритма над данным графом $G = \langle V, E \rangle$ будет описываться как функция, зависящая от двух входных параметров: количества вершин $|V|$ и количества рёбер $|E|$ графа.

§ 3. Базовые понятия теории графов

Мы будем рассматривать два типа графов: ориентированные и неориентированные.

Определение. *Ориентированным графом* будем называть упорядоченную пару $G = \langle V, E \rangle$, где V — конечное множество, а E — бинарное отношение на V, т.е. каждый элемент E представим в виде упорядоченной пары $\langle u, v \rangle$, где $u, v \in V$.

Определение. *Неориентированным графом* будем называть упорядоченную пару $G = \langle V, E \rangle$, где V — конечное множество, а E — некоторое множество двухэлементных подмножеств V, т.е. каждый элемент E представим в виде неупорядоченной пары $\{u, v\}$, где $u, v \in V$ и $u \neq v$.

Определение. Пусть $G = \langle V, E \rangle$ — ориентированный или неориентированный граф. Элементы V называют *вершинами*, а элементы E — *рёбрами (дугами)* графа G. Для рёбер графа будем использовать единую запись (u, v), причём в неориентированном случае (u, v) и (v, u) обозначают одно и то же ребро.

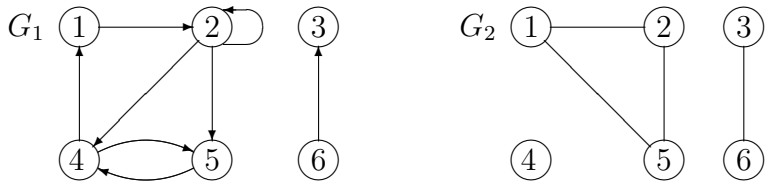
Ориентированные графы могут содержать *петли*, т.е. рёбра вида (v, v). В неориентированных графах петли запрещены.

Через $|V|$ будем обозначать количество вершин в графе G, а через $|E|$ — количество его рёбер.

Графическое изображение графов

Графы удобно представлять графически. Вершины графа рисуют в виде кружков, помеченных именами вершин. Рёбра в ориентированных графах изображают стрелками: если в графе есть ребро (u, v), то рисуют стрелку, выходящую из вершины u и входящую в вершину v. Рёбра в неориентированных графах изображают линиями: если в графе есть ребро (u, v), то рисуют линию, соединяющую вершины u и v.

Пример. На рисунке ниже изображены ориентированный граф G_1 с множеством вершин $\{1, 2, 3, 4, 5, 6\}$ и неориентированный граф G_2 с таким же множеством вершин.



Многие определения выглядят одинаково и для ориентированных, и для неориентированных графов, хотя некоторые отличия, конечно, имеются.

Определение. Если в ориентированном или неориентированном графе G имеется ребро (u, v), то говорят, что вершина v *смежна* с вершиной u.

Определение. *Путём длины* k из вершины u в вершину u' в ориентированном или неориентированном графе $G = \langle V, E\rangle$ называют упорядоченный набор $\langle v_0, v_1, \ldots, v_k\rangle$ вершин графа G такой, что $u = v_0$, $u' = v_k$ и $(v_i, v_{i+1}) \in E$ для всех $i < k$. Если имеется путь p из вершины u в вершину u', то говорят, что вершина u' *достижима* из u по пути p. Путь называется *простым*, если все вершины пути различны.

Пример (продолжение)**.** В графах G_1 и G_2 из предыдущего примера вершина 2 смежна с вершиной 1, поскольку ребро $(1, 2)$ присутствует в обоих графах. Вершина 1 смежна с вершиной 2 только в графе G_2.

В графе G_1 путь $\langle 1, 2, 5, 4\rangle$ является простым путём длины 3, а путь $\langle 2, 5, 4, 5\rangle$ простым не является.

Определение. В ориентированном графе путь $\langle v_0, v_1, \ldots, v_k\rangle$ образует *цикл*, если $v_0 = v_k$ и путь содержит хотя бы одно ребро, т.е. $k \geqslant 1$. Цикл называют *простым*, если, кроме этого, все вершины $v_1, v_2, \ldots, v_k$ различны.

В неориентированном графе путь $\langle v_0, v_1, \ldots, v_k\rangle$ образует *(простой) цикл*, если $k \geqslant 3$, $v_0 = v_k$ и все вершины $v_1, v_2, \ldots, v_k$ различны.

Ориентированный или неориентированный граф без циклов называется *ациклическим*.

Пример (продолжение)**.** В графе G_1 из предыдущего примера путь $\langle 1, 2, 4, 1\rangle$ образует тот же цикл, что и пути $\langle 2, 4, 1, 2\rangle$ и $\langle 4, 1, 2, 4\rangle$. Этот цикл простой. Цикл $\langle 1, 2, 4, 5, 4, 1\rangle$ не является простым. Петля $(2, 2)$ является циклом длины 1.

В графе G_2 единственным циклом является путь $\langle 1, 2, 5, 1\rangle$.

Определение. В ориентированном графе G вершины u и v *взаимно достижимы*, если в G существуют путь из u в v и путь из v в u. Отношение взаимной достижимости является отношением эквивалентности на множестве вершин графа G. Классы эквивалентности по данному отношению называют *сильно связными компонентами* графа G. Граф G является *сильно связным*, если он состоит из единственного сильно связного компонента, т.е. любые две его вершины взаимно достижимы.

Определение. В неориентированном графе G вершина v *достижима из* вершины u, если в G существует путь из u в v. Отношение достижимости одной вершины из другой является отношением эквивалентности на множестве вершин графа G. Классы эквивалентности по данному отношению называют *связными компонентами* графа G. Граф G является *связным*, если он состоит из единственного связного компонента, т.е. любая его вершина достижима из любой другой по некоторому пути.

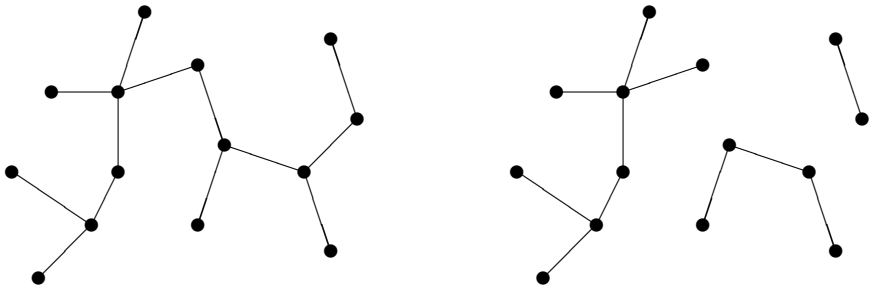
Замечание. Для любой вершины u набор $\langle u \rangle$ образует путь длины 0, т.е. любая вершина достижима из самой себя в любом графе. Таким образом, отношение достижимости одной вершины из другой, а значит, и отношение взаимной достижимости являются рефлексивными.

Пример (продолжение)**.** Граф G_1 из предыдущего примера разбивается на три сильно связных компонента: $\{1, 2, 4, 5\}$, $\{3\}$ и $\{6\}$. Все пары вершин в множестве $\{1, 2, 4, 5\}$ взаимно достижимы. Вершины $\{3, 6\}$ не образуют сильно связный компонент, т.к. вершина 6 не достижима из вершины 3.

Граф G_2 состоит из трёх связных компонентов: $\{1, 2, 5\}$, $\{3, 6\}$ и $\{4\}$.

Определение. Ациклический неориентированный граф называется *лесом*, а связный ациклический неориентированный граф — *деревом*.

Пример. Слева на рисунке представлен пример дерева, а справа — пример леса, не являющегося деревом.



В следующем утверждении указано несколько важных свойств деревьев.

Теорема 4. *Пусть $G = \langle V, E \rangle$ — неориентированный граф. Тогда следующие условия эквивалентны:*
(1) G — дерево.
(2) Любые две вершины G соединяются при помощи единственного простого пути.
(3) G — связный граф, но при удалении из E любого ребра перестаёт быть связным.
(4) G — связный граф и $|E| = |V| - 1$.

Доказательство. Предлагается читателю в качестве упражнения. □

Определение. *Деревом с корнем* называют упорядоченную пару $\langle T, r \rangle$, где T — дерево, а r — выделенная вершина T, именуемая *корнем* дерева.

Определение. Пусть x — вершина в дереве T с корнем r. Любая вершина y на единственном простом пути из r в x называется *предком* вершины x. Если y является предком вершины x, то x называется *потомком* вершины y.

В частности, любая вершина дерева является собственным предком и потомком.

Длина простого пути от корня r до вершины x называется *глубиной* вершины x в дереве T. *Высота* дерева T определяется как наибольшая глубина вершин из T.

Упражнения

1. Доказать, что если ориентированный или неориентированный граф содержит путь из вершины u в вершину v, то в нём есть простой путь из u в v.

2. Доказать, что если в ориентированном графе есть цикл, то в нём есть и простой цикл.

3. Доказать, что для любого связного неориентированного графа $G = \langle V, E\rangle$ выполняется соотношение $|E| \geqslant |V| - 1$.

4. Доказать теорему 4.

§ 4. Поиск в ширину

В этом параграфе рассматривается метод обхода графа, который принято называть *поиском в ширину* (breadth-first search). Под обходом графа понимается систематическое перемещение по рёбрам графа, при котором посещаются все его вершины, удовлетворяющие целям обхода. Алгоритм обхода графа может многое сказать о структуре графа, поэтому многие другие алгоритмы начинают свою работу с получения информации о свойствах графа путём его обхода.

Описание алгоритма

Пусть задан граф $G = \langle V, E\rangle$ и выделена *исходная* вершина $s \in V$. Алгоритм поиска в ширину обходит все рёбра G для *открытия* всех вершин, достижимых из s, вычисляя при этом *расстояние* (минимальное количество рёбер) от s до каждой достижимой из s вершины. Стратегия обхода в ширину заключается в движении вширь, т.е. перед тем как приступить к поиску вершин на расстоянии $k+1$, выполняется обход всех вершин на расстоянии k. Расстояние от s до вершины u хранится в переменной $d[u]$.

Алгоритм раскрашивает вершины графа в белый, серый и чёрный цвета. Изначально все вершины белые, и позже они могут стать серыми, а затем чёрными. Цвет вершины u хранится в переменной $color[u]$.

Алгоритм использует *очередь* Q, которая в каждый момент времени представляет из себя упорядоченный набор вершин графа. При добавлении в очередь вершина занимает место в её хвосте. При извлечении из очереди выводится вершина, находящаяся в её голове.

В процессе обхода также строится *дерево поиска* с корнем s, содержащее все достижимые из s вершины. Если в процессе сканирования вершин, смежных с уже открытой вершиной u, открывается новая вершина v, то вершина v и ребро (u, v) добавляются в дерево поиска, при этом вершина u объявляется *предшественником* v в дереве поиска. Предшественник вершины v хранится в переменной $\pi[v]$. Если у v нет предшественника, то $\pi[v] \uparrow$, т.е. значение $\pi[v]$ не определено.

Приведённая ниже процедура поиска в ширину BFS работает как для ориентированных, так и для неориентированных графов.

```
BFS(G, s)
{Вход: граф G = ⟨V, E⟩, вершина s ∈ V.}
{Выход: расстояние d[u] для всех u ∈ V, предшественник π[u] для некоторых u ∈ V.}
01  for каждой вершины u ∈ V \ {s}
02      do color[u] ← WHITE
03         d[u] ← ∞
04         π[u] ↑
05  color[s] ← GRAY
```

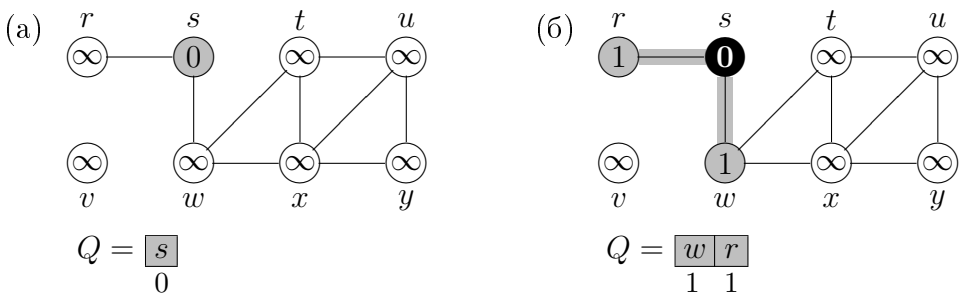
```
06  d[s] ← 0
07  π[s] ↑
08  Q ← ∅
09  INJECT(Q, s)
10  while Q ≠ ∅
11        do u ← EJECT(Q)
12           for каждого ребра (u, v) ∈ E
13                 do if color[v] = WHITE
14                       then color[v] ← GRAY
15                            d[v] ← d[u] + 1
16                            π[v] ← u
17                            INJECT(Q, v)
18           color[u] ← BLACK
```

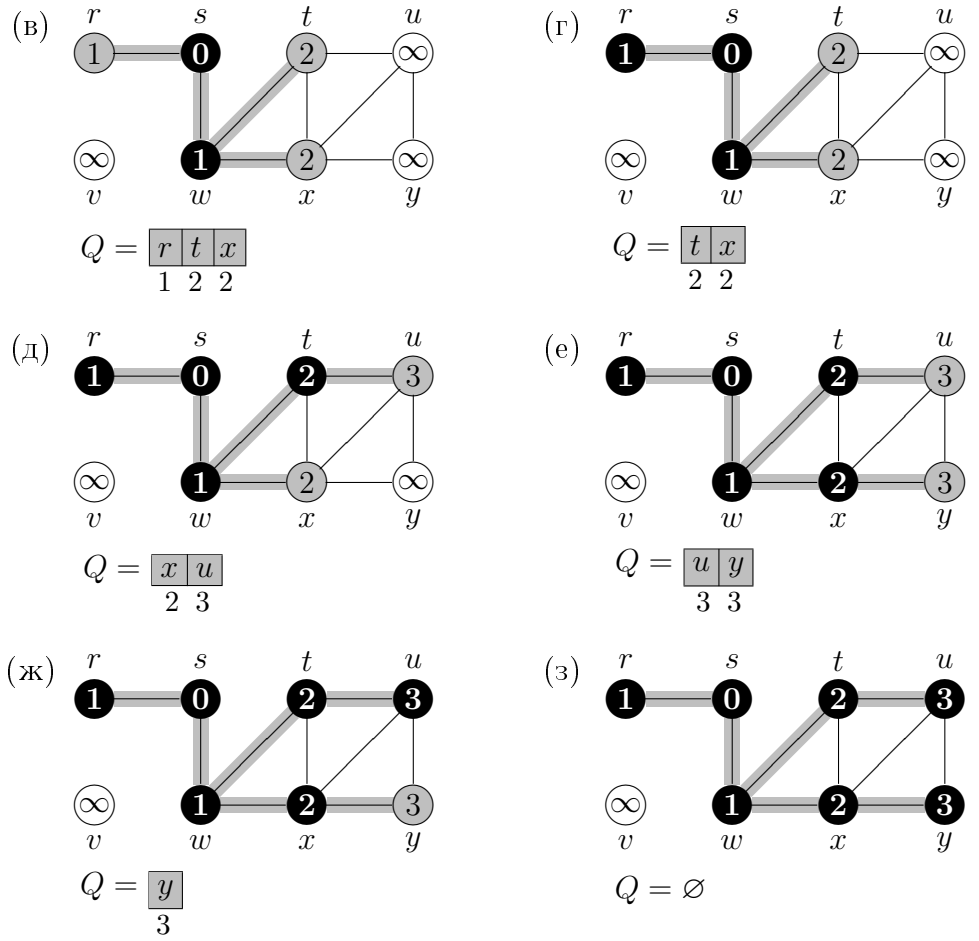
Процедура BFS работает следующим образом. В строках 01–04 все вершины $u \in V$, за исключением вершины s, окрашиваются в белый цвет, переменной $d[u]$ присваивается значение ∞, а предшественник $\pi[u]$ не определён. В строках 05–07 открывается исходная вершина s, она окрашивается в серый цвет, переменной $d[s]$ присваивается значение 0, предшественник $\pi[s]$ не определён. В строках 08–09 создаётся пустая очередь Q, в которую затем помещается s.

Цикл **while** в строках 10–18 выполняется до тех пор, пока в очереди Q есть вершины. Заметим, что на каждой итерации при проверке в строке 10 очередь Q состоит в точности из всех серых вершин. В строке 11 из головы очереди извлекается вершина u. Цикл **for** в строках 12–17 просматривает все вершины v, смежные с u. Если вершина v белая, значит, она ещё не открыта, и алгоритм открывает её, выполняя строки 14–17, в которых v окрашивается в серый цвет, расстояние $d[v]$ устанавливается равным $d[u] + 1$, а её предшественником объявляется вершина u. Затем v помещается в хвост очереди. После того как все вершины, смежные с u, просмотрены, вершине u в строке 18 присваивается чёрный цвет.

Результаты поиска в ширину могут зависеть от порядка просмотра вершин, смежных с данной вершиной, в строке 12. Дерево поиска и предшественники вершин могут варьироваться, но расстояния d, вычисленные алгоритмом, не зависят от порядка просмотра.

Пример. Ниже проиллюстрирована работа процедуры BFS на примере восьмиэлементного неориентированного графа. Внутри каждой вершины u приведено текущее значение $d[u]$. Состояние очереди Q показано на момент проверки в строке 10. Под элементами очереди изображены расстояния от исходной вершины, вычисленные алгоритмом. Рёбра, которые добавляются в дерево поиска, выделяются.





Свойства алгоритма

Для изучения свойств алгоритма поиска в ширину и обоснования его корректности введём функцию длины кратчайшего пути в графе.

Определение. Для произвольного ориентированного или неориентированного графа $G = \langle V, E\rangle$ и вершин $s, v \in V$ определим *длину* $\delta(s, v)$ *кратчайшего пути от* s *до* v в графе G следующим образом:

(а) если вершина v достижима из s в графе G, то определим $\delta(s, v)$ как минимальное количество рёбер на каком-либо пути от s до v (такой путь называется *кратчайшим от* s *до* v);

(б) если же вершина v не достижима из s в графе G, положим $\delta(s, v) = \infty$.

Лемма 5. *Пусть* $G = \langle V, E\rangle$ *— ориентированный или неориентированный граф,* $s \in V$, $(u, v) \in E$. *Тогда* $\delta(s, v) \leqslant \delta(s, u) + 1$.

Доказательство. Если вершина u достижима из s, то достижима и вершина v. В этом случае кратчайший путь из s в v не может быть длиннее пути, составленного из кратчайшего пути из s в u и ребра (u, v). Следовательно, $\delta(s, v) \leqslant \delta(s, u) + 1$.

Если же вершина u не достижима из s, то $\delta(s, u) = \infty$. Следовательно, $\delta(s, u)+1 = \infty \geqslant \delta(s, v)$. □

Лемма 6. *Пусть* $G = \langle V, E\rangle$ *— ориентированный или неориентированный граф и* $s \in V$. *Тогда для каждой вершины* $v \in V$ *значение* $d[v]$, *вычисленное процедурой* BFS(G, s), *удовлетворяет неравенству* $d[v] \geqslant \delta(s, v)$.

Доказательство. Непосредственно после инициализации все вершины $v \in V \setminus \{s\}$ имеют белый цвет, и поэтому $d[v] = \infty \geqslant \delta(s, v)$. Кроме этого, для вершины s справедливо $d[s] = 0 = \delta(s, s)$.

Из кода процедуры BFS видно, что значение $d[v]$ может измениться только у белой вершины v, причём после такого изменения вершина v помещается операцией INJECT в очередь, окрашивается в серый цвет и никогда уже не станет белой, и значит, после этого значение $d[v]$ больше не изменяется.

Таким образом, достаточно доказать утверждение леммы для вершин v, у которых значение $d[v]$ меняется с бесконечного на конечное. Доказательство проведём индукцией по числу применений операции INJECT.

Впервые операция INJECT используется в строке 09 для исходной вершины s. После её применения, как было установлено выше, неравенство $d[v] \geqslant \delta(s, v)$ выполняется для всех $v \in V$.

Пусть после k-го применения операции INJECT справедливо $d[v] \geqslant \delta(s, v)$ для всех $v \in V$. Рассмотрим $(k{+}1)$-е применение операции INJECT (строка 17), которое помещает некоторую вершину v в очередь. В этом случае вершина v смежна вершине u, цвет которой отличен от белого. Следовательно, значение $d[u]$ уже стабилизировалось, и, в силу индукционного предположения, справедливо $d[u] \geqslant \delta(s, u)$. Тогда в силу присвоения в строке 15 и леммы 5 получаем

$$d[v] = d[u] + 1 \geqslant \delta(s, u) + 1 \geqslant \delta(s, v).$$

□

Следующая лемма показывает, что в любой момент времени в очереди находится не более двух различных значений расстояния d.

Лемма 7. *Предположим, что в процессе выполнения процедуры* $\mathrm{BFS}(G, s)$ *очередь находится в состоянии* $Q = \langle v_1, v_2, \ldots, v_r \rangle$. *Тогда* $d[v_r] \leqslant d[v_1] + 1$ *и* $d[v_i] \leqslant d[v_{i+1}]$ *для всех* $1 \leqslant i \leqslant r - 1$.

Доказательство. Докажем утверждение леммы индукцией по числу операций с очередью. Когда очередь пуста или состоит из одной исходной вершины s, утверждение очевидно.

Если из очереди извлекается её голова v_1, новой головой очереди становится вершина v_2. В силу индукционного предположения, $d[v_1] \leqslant d[v_2]$. Тогда $d[v_r] \leqslant d[v_1] + 1 \leqslant$ $\leqslant d[v_2] + 1$, а все остальные неравенства не изменяются.

Если в очередь помещается вершина v после исполнения строки 17, то v становится вершиной v_{r+1} в хвосте очереди. Следовательно, вершина v смежна некоторой вершине u, которая в текущей итерации цикла **while** была удалена из очереди. Пусть непосредственно перед удалением u из очереди её состояние имело вид $\langle u, v_1, v_2, \ldots, v_p \rangle$ для некоторого $p \leqslant r$. Следовательно, вершины $v_{p+1}, \ldots, v_r$ тоже были добавлены в очередь в текущей итерации цикла **while**, и значит, $v_{p+1}, \ldots, v_r$ тоже смежны вершине u. Если $p = 0$, то $d[v_1] = \cdots = d[v_{r+1}] = d[u] + 1$ и утверждение леммы очевидно. Если же $p \geqslant 1$, то, в силу индукционного предположения для состояния очереди $\langle u, v_1, v_2, \ldots, v_p \rangle$, справедливы неравенства $d[u] \leqslant d[v_1]$ и $d[v_p] \leqslant d[u] + 1$. Тогда $d[v_{r+1}] = d[v] = d[u] + 1 \leqslant d[v_1] + 1$. Кроме этого, $d[v_p] \leqslant d[u] + 1 = d[v_{p+1}] =$ $= \cdots = d[v_{r+1}]$, а все остальные неравенства не изменяются. □

Приведённое ниже следствие показывает, что значения d помещаемых в очередь вершин монотонно возрастают.

Следствие 8. *Предположим, что в процессе выполнения процедуры* $\mathrm{BFS}(G, s)$ *вершины* v_i *и* v_j *помещаются в очередь* Q*, причём* v_i *попадает в очередь до* v_j*. Тогда в момент помещения* v_j *в очередь выполняется неравенство* $d[v_i] \leqslant d[v_j]$.

Доказательство. Вытекает из леммы 7 и того факта, что в процессе выполнения процедуры BFS каждая вершина v получает конечное значение $d[v]$ не более одного раза. □

Дерево поиска в ширину однозначно определяется функцией предшественников.

Определение. Пусть π — функция предшественников, вычисленная в результате применения процедуры $\mathrm{BFS}(G, s)$ к графу $G = \langle V, E\rangle$ с исходной вершиной $s \in V$. Определим неориентированный граф $G_\pi = \langle V_\pi, E_\pi\rangle$, где

$$V_\pi = \{s\} \cup \{v \in V \mid \pi[v] \downarrow\},$$
$$E_\pi = \{(\pi[v], v) \mid v \in V_\pi \setminus \{s\}\}.$$

Граф G_π является деревом (доказательство этого факта приведено ниже в теореме 9) и называется *деревом поиска в ширину*.

Теорема 9 (о корректности поиска в ширину)**.** *Пусть* $G = \langle V, E\rangle$ *— ориентированный или неориентированный граф и пусть процедура* BFS *выполняется над графом* G *с исходной вершиной* $s \in V$*. Тогда справедливы следующие утверждения:*

(1) По окончании работы $\mathrm{BFS}(G, s)$ *для всех* $v \in V$ *выполняется* $d[v] = \delta(s, v)$.

(2) В процессе работы $\mathrm{BFS}(G, s)$ *открываются все вершины, достижимые из* s.

(3) Граф G_π *является деревом, состоящим из всех вершин, достижимых из* s *в графе* G.

(4) Для всех $v \in V_\pi$ *в дереве* G_π *имеется единственный простой путь из* s *в* v*, который одновременно является кратчайшим путём из* s *в* v *в графе* G.

Доказательство. (1) Допустим, напротив, существуют вершины $v \in V$ такие, что $d[v] \neq \delta(s, v)$. Среди всех таких вершин выберем вершину v с минимальной длиной $\delta(s, v)$. Очевидно, что $v \neq s$. В силу леммы 6, имеет место $d[v] \geqslant \delta(s, v)$. Следовательно, в силу нашего предположения, $d[v] > \delta(s, v)$. Тогда вершина v должна быть достижима из s, так как иначе $\delta(s, v) = \infty \geqslant d[v]$. Выберем в графе G какой-нибудь кратчайший путь из s в v. Так как $v \neq s$, на этом пути есть вершина u, непосредственно предшествующая вершине v. Следовательно, $\delta(s, v) = \delta(s, u) + 1$. Таким образом, $\delta(s, u) < \delta(s, v)$ и, значит, в силу выбора вершины v, заключаем, что $d[u] = \delta(s, u)$. Объединяя найденные соотношения, получаем

$$d[v] > \delta(s, v) = \delta(s, u) + 1 = d[u] + 1. \qquad (*)$$

Ясно, что вершина u тоже достижима из s. Следовательно, значение $\delta(s, u)$ конечно, и значит, $d[u]$ тоже конечно. Из кода процедуры BFS видно, что если $d[u]$ присваивается конечное значение, то u попадает в очередь Q. Следовательно, наступит момент, когда u будет удалена из очереди (строка 11). В этот момент вершина v может быть белой, серой или чёрной. Покажем, что в любом из этих трёх случаев получается противоречие с неравенством $(*)$. Если v белая, то в строке 15 выполняется присваивание $d[v] = d[u] + 1$, противоречащее $(*)$. Если v чёрная, то она уже

удалена из очереди и, в силу следствия 8, $d[v] \leqslant d[u]$, что опять противоречит $(*)$. Если v серая, то она была покрашена в этот цвет при удалении из очереди некоторой вершины w, которая была удалена раньше вершины u и для которой выполняется $d[v] = d[w]+1$. Однако из следствия 8 вытекает, что $d[w] \leqslant d[u]$, поэтому $d[v] \leqslant d[u]+1$, что тоже противоречит $(*)$.

(2) Допустим, существует достижимая из s вершина v, которая не открывается процедурой $\mathrm{BFS}(G, s)$. Тогда $d[v] = \infty$, но при этом значение $\delta(s, v)$ конечно, что противоречит доказанному выше свойству $d[v] = \delta(s, v)$.

(3) Заметим, что для вершины $v \neq s$ условие $v \in V_\pi$ равносильно тому, что значение $\pi[v]$ становится определённым в строке 16 процедуры BFS, что, в свою очередь, происходит тогда и только тогда, когда значение $d[v]$ становится конечным в строке 15. Поскольку $d[v] = \delta(s, v)$, заключаем, что $v \in V_\pi$ тогда и только тогда, когда $\delta(s, v) < \infty$, т.е. v достижима из s в графе G. Также отметим, что если $\pi[v] = u$, то $d[v] = d[u] + 1$, и следовательно, из условия $d[v] = \delta(s, v) < \infty$ вытекает соотношение $d[u] = \delta(s, u) < \infty$, т.е. $u \in V_\pi$, а ребро $(\pi[v], v) \in E_\pi$ действительно соединяет вершины из V_π.

Таким образом, граф G_π определён корректно и состоит из всех вершин, достижимых из s в G. Тогда G_π — связный граф, поскольку любые две его вершины связаны путями с s. Кроме этого, из определения графа G_π следует, что $|E_\pi| = |V_\pi| - 1$. Следовательно, в силу теоремы 4, граф G_π является деревом.

(4) Пусть $v \in V_\pi$. Поскольку G_π является деревом, в силу теоремы 4, заключаем, что в G_π существует единственный простой путь $p_v = \langle v_0, \ldots, v_k \rangle$ от корня $s = v_0$ к вершине $v = v_k$. Докажем, что для всех $1 \leqslant i \leqslant k$ справедливо свойство $v_{i-1} = \pi[v_i]$ (т.е. случай $\pi[v_{i-1}] = v_i$ невозможен). Действительно, для $i = 1$ это свойство справедливо, поскольку $\pi[s]$ не определено. Далее, рассуждая по индукции, допустим, что для фиксированного $i \geqslant 1$ справедливо свойство $v_{i-1} = \pi[v_i]$. Если тождество $v_i = \pi[v_{i+1}]$ неверно, то $\pi[v_i] = v_{i+1}$, откуда следует, что $v_{i-1} = v_{i+1}$, а это противоречит простоте пути p_v. Следовательно, $v_i = \pi[v_{i+1}]$, что и требовалось.

Теперь индукцией по длине k пути p_v докажем, что этот путь одновременно является кратчайшим путём из s в v в графе G. Если $k = 0$, то путь, состоящий из одной вершины s, очевидно, является кратчайшим в G. Если же $k > 0$, то $v \neq s$, и значит, определено значение $\pi[v] = u$. Отсюда, в силу условий $v = v_k$ и $v_{k-1} = \pi[v_k]$, вытекает, что $u = v_{k-1}$. Ясно, что путь $p_u = \langle v_0, \ldots, v_{k-1} \rangle$ является простым путём от s к u в дереве G_π. Следовательно, в силу индукционного предположения, p_u является кратчайшим путём из s в u в графе G, и значит, длина пути p_u совпадает со значением $\delta(s, u)$. Тогда длина пути p_v равна $\delta(s, u) + 1$. Заметим, что по построению $d[v] = d[u] + 1$, откуда следует, что $\delta(s, v) = \delta(s, u) + 1$. Тогда длина пути p_v равна $\delta(s, v)$, т.е. p_v — кратчайший путь из s в v в графе G. □

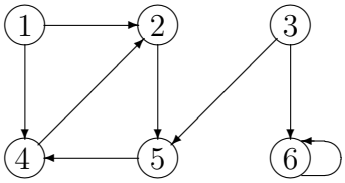
Время работы алгоритма

Инициализация в строках 01–07 расходует одно и то же постоянное количество операций на каждую из вершин. Поэтому общее время выполнения инициализации равно $O(|V|)$. После инициализации ни одна вершина не окрашивается в белый цвет, поэтому проверка в строке 13 гарантирует, что каждая вершина вносится в очередь не более одного раза, а следовательно, и удаляется из очереди не более одного раза. Операции внесения в очередь и удаления из неё требуют $O(1)$ времени, так что общее количество операций с очередью составляет $O(|V|)$. Все оставшиеся операции производятся в цикле **while**, внутри которого для каждой вершины u, извлекаемой

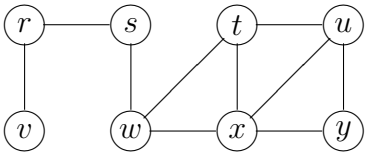
из очереди, сканируется список всех выходящих из u рёбер, для каждого из которых осуществляется $O(1)$ операций. Если граф ориентированный, то ребро в цикле просматривается один раз; если неориентированный — то два раза. Поскольку суммарное количество рёбер равно $|E|$, общее время выполнения всех итераций цикла **while** равно $O(|E|)$. Таким образом, общее время работы процедуры BFS составляет $O(|V|+|E|)$, т.е. время поиска в ширину линейно зависит от размера входного графа.

Упражнения

1. Применить алгоритм поиска в ширину к следующему ориентированному графу, взяв в качестве исходной вершину 3.



2. Применить алгоритм поиска в ширину к следующему неориентированному графу, взяв в качестве исходной вершину u.



   Применить алгоритм поиска в ширину к этому же графу, изменив порядок просмотра вершин, смежных с данной.

3. Привести пример ориентированного графа $G = \langle V, E\rangle$, исходной вершины $s \in V$ и множества рёбер $E' \subseteq E$ таких, что граф $G' = \langle V, E'\rangle$ является деревом, для каждой вершины $v \in V$ единственный путь в дереве G' от s к v будет кратчайшим путём из s к v в графе G, но дерево G' невозможно получить как дерево поиска процедуры $\mathrm{BFS}(G, s)$ ни при каком порядке просмотра вершин. (Считаем, что в графе G' рёбра из E' подвергаются операции дезориентации, т.е. превращаются из упорядоченных пар в неупорядоченные.)

4. Используя процедуру BFS, разработать алгоритм нахождения всех связных компонентов неориентированного графа.

§ 5. Поиск в глубину

Обход графа методом *поиска в глубину* (depth-first search) часто используется в качестве подпрограммы в других алгоритмах. Так же как и обход в ширину, алгоритм поиска в глубину даёт важную информацию о структуре графа. Основное отличие поиска в глубину от поиска в ширину заключается в том, что множество открытых к данному моменту вершин организовано не в виде очереди, как это было сделано в

процедуре BFS, а в виде *стека*. В процедуре поиска в глубину, изложенной в данном параграфе, вместо явного стека используется рекурсия.

Описание алгоритма

Стратегия поиска в глубину состоит в том, чтобы идти вглубь графа, насколько это возможно. Поиск в глубину исследует все рёбра, выходящие из вершины u, открытой последней, при этом открываются новые вершины, что означает запуск рекурсии. Когда не останется неисследованных рёбер, выходящих из u, поиск покидает вершину u и возвращается в ту, из которой была открыта u. Этот процесс продолжается до тех пор, пока не будут открыты все вершины, достижимые из исходной. Если при этом остаются неоткрытые вершины, то одна из них выбирается в качестве новой исходной вершины и поиск повторяется уже из неё. Поиск в глубину продолжается до тех пор, пока не будут открыты все вершины графа.

Алгоритм раскрашивает каждую вершину графа последовательно в белый, потом в серый и наконец в чёрный цвета. Изначально каждая вершина белая, затем при *открытии* она окрашивается в серый цвет, а по *завершении*, т.е. когда исследованы все выходящие из неё рёбра, становится чёрной. Цвет вершины u хранится в переменной $color[u]$.

Алгоритм также проставляет в вершинах *метки времени*. Каждая вершина u имеет две такие метки. В первой метке $d[u]$ указывается *время открытия* вершины u. Вторая метка $f[u]$ фиксирует *время завершения* исследования выходящих из u рёбер. Метки времени представляют собой целые числа от 1 до $2{\cdot}|V|$. Для проставления меток времени используется глобальная переменная $time$.

Как и в случае поиска в ширину, когда вершина v открывается в процессе сканирования вершин, смежных с уже открытой вершиной u, алгоритм поиска в глубину объявляет u *предшественником* вершины v. Предшественник вершины v хранится в переменной $\pi[v]$, а рёбра вида $(\pi[v], v)$ образуют *лес поиска*, который может состоять из нескольких *деревьев поиска*.

Представленная ниже процедура поиска в глубину DFS работает как для ориентированных, так и для неориентированных графов.

```
DFS(G)
{Вход: граф G = ⟨V, E⟩.}
{Выход: время открытия d[u] и время завершения f[u] для всех u ∈ V, предшественник π[u] для некоторых u ∈ V.}
01  for каждой вершины u ∈ V
02      do color[u] ← WHITE
03         π[u] ↑
04  time ← 0
05  for каждой вершины u ∈ V
06      do if color[u] = WHITE
07            then DFS_Visit(u)

DFS_Visit(u)
01  color[u] ← GRAY
02  time ← time + 1
03  d[u] ← time
04  for каждого ребра (u, v) ∈ E
05      do if color[v] = WHITE
06            then π[v] ← u
```

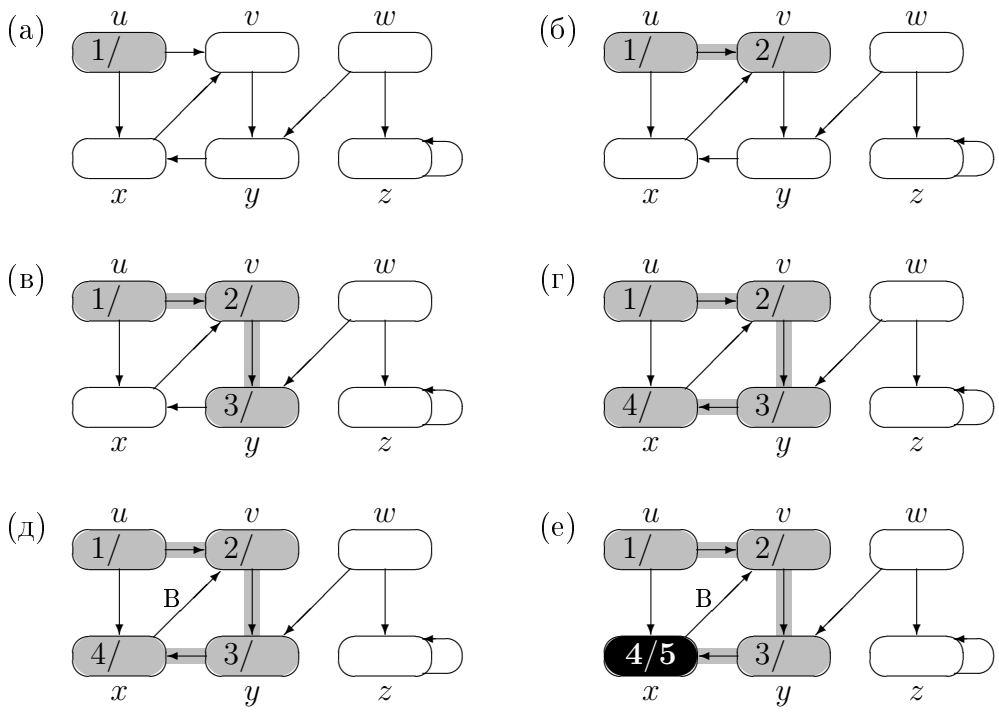
07 DFS_Visit(v)
08 $color[u] \leftarrow$ BLACK
09 $time \leftarrow time + 1$
10 $f[u] \leftarrow time$

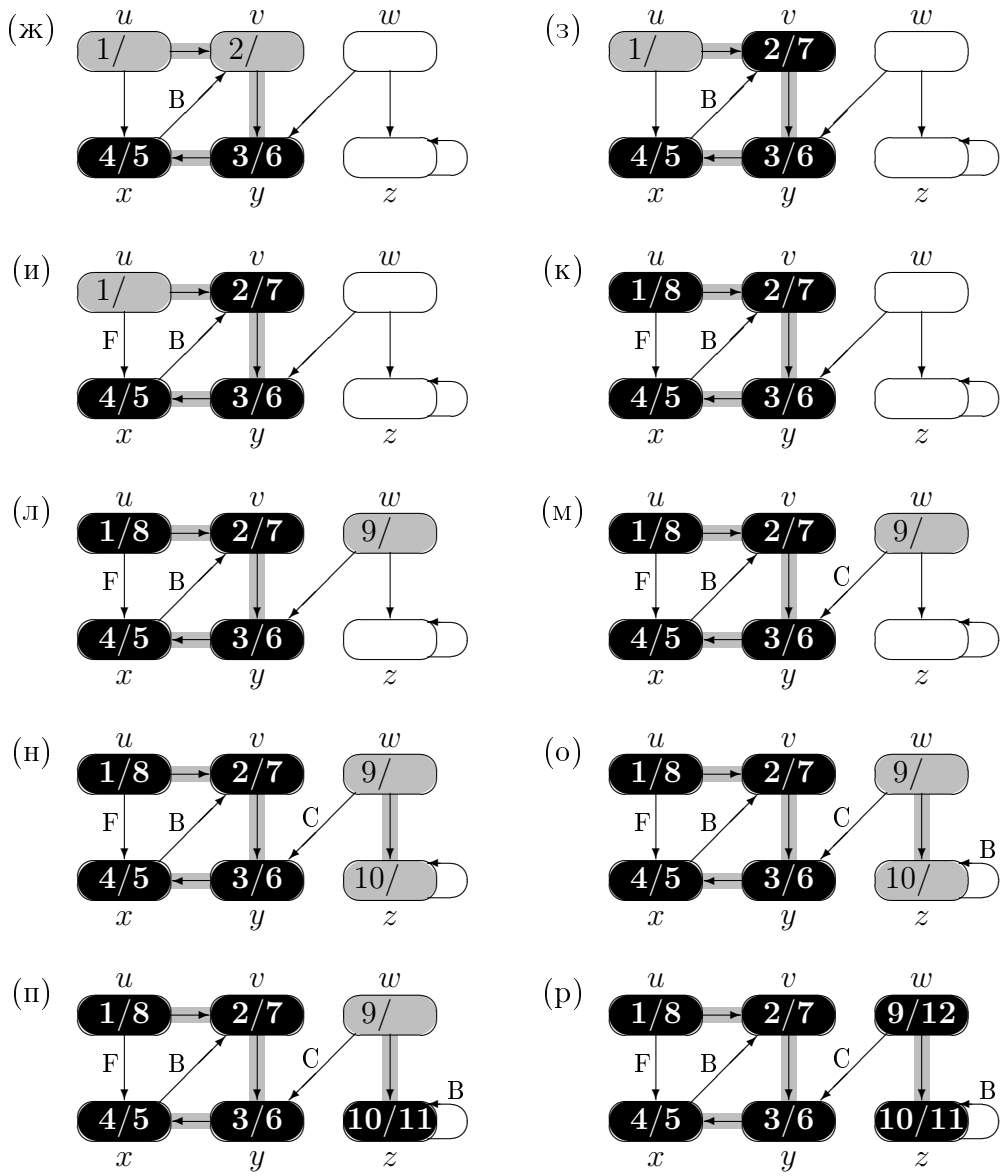
Процедура DFS работает следующим образом. В строках 01–03 все вершины окрашиваются в белый цвет, а их предшественники объявляются неопределёнными. В строке 04 выполняется сброс глобального счётчика времени. В строках 05–07 поочерёдно проверяются все вершины u из V, и, когда обнаруживается белая вершина, она обрабатывается с помощью процедуры DFS_Visit, которая вызывается в строке 07, при этом вершина u становится корнем нового дерева из леса поиска.

В момент вызова процедуры DFS_Visit(u) вершина u имеет белый цвет. В строках 01–03 она окрашивается в серый цвет, глобальная переменная $time$ увеличивается на единицу, и новое значение $time$ фиксируется как время открытия $d[u]$ вершины u. В строках 04–07 исследуются все вершины v, смежные с u, и, если вершина v белая, её предшественником объявляется вершина u, и выполняется рекурсивное посещение вершины v. Наконец, после того как исследованы все вершины, смежные с u, в строках 08–10 вершина u окрашивается в чёрный цвет, переменная $time$ увеличивается на единицу и записывается время завершения $f[u]$ работы с вершиной u.

Результаты поиска в глубину могут зависеть от порядка просмотра вершин в строке 05 процедуры DFS, а также от порядка посещения смежных вершин в строке 04 процедуры DFS_Visit.

Пример. Ниже проиллюстрирована работа процедуры DFS на примере шестиэлементного ориентированного графа. В вершинах указаны метки времени в формате открытие/завершение. Рёбра, исследованные процедурой DFS_Visit в строке 04, либо выделяются серым цветом (если это рёбра деревьев), либо помечены буквами F (прямые рёбра), B (обратные рёбра) или C (перекрёстные рёбра). (Разметка рёбер не входит в функции процедуры DFS. Определение типов рёбер будет приведено позже.)





Свойства алгоритма

Наиболее фундаментальное свойство поиска в глубину заключается в том, что определённый с помощью функции предшественников π лес поиска в глубину в точности отражает структуру рекурсивных вызовов процедуры DFS_Visit.

Определение. Пусть π — функция предшественников, вычисленная в результате применения процедуры DFS(G) к графу $G = \langle V, E\rangle$. Определим неориентированный граф $G_\pi = \langle V, E_\pi\rangle$, где

$$E_\pi = \{(\pi[v], v) \mid v \in V \text{ и } \pi[v] \downarrow\}.$$

Граф G_π является лесом (доказательство этого факта приведено ниже в теореме 11), называется *лесом поиска в глубину* и состоит из нескольких *деревьев поиска в глубину*. Вершины $v \in V$, для которых не определено значение $\pi[v]$, будем называть *корнями деревьев поиска в глубину*.

Лемма 10. *Пусть $G = \langle V, E\rangle$ — ориентированный или неориентированный граф, $u, v \in V$ и $d[u] \leqslant d[v] < f[u]$. Тогда в графе G_π существует путь из вершины u в вершину v.*

Доказательство. Докажем утверждение леммы индукцией по значению $d[v]$. Если $d[v] = d[u]$, то $v = u$ и утверждение очевидно.

Пусть $d[v] > d[u]$. Поскольку $d[u] < d[v] < f[u]$, заключаем, что вершина v открывается в процессе исполнения процедуры DFS_Visit(u). Следовательно, в процессе исполнения DFS_Visit(u) для некоторой вершины u' происходит рекурсивный вызов процедуры DFS_Visit(u'), при исполнении которой вершина v открывается как смежная с u'. Таким образом, по построению $\pi[v] = u'$ и ребро (u', v) добавляется в граф G_π. Ясно, что $d[u] \leqslant d[u'] < d[v]$. Тогда, в силу индукционного предположения, в графе G_π существует путь p из u в u'. Дополнив путь p ребром $(u', v) \in E_\pi$, получаем искомый путь из u в v. □

Теорема 11 (о корректности поиска в глубину)**.** *Пусть $G = \langle V, E\rangle$ — ориентированный или неориентированный граф и пусть процедура* DFS *выполняется над графом G. Тогда справедливы следующие утверждения:*

(1) В процессе работы DFS(G) *открываются все вершины графа G.*

(2) Граф G_π является лесом, состоящим из нескольких деревьев.

Доказательство. (1) Цикл **for** в строках 05–07 процедуры DFS просматривает все вершины $u \in V$, и если вершина u белая, то она открывается вызовом процедуры DFS_Visit(u), а если не белая, то u уже была открыта ранее. Таким образом, к моменту окончания работы DFS(G) все вершины графа будут открыты.

(2) Пусть $r_1, \ldots, r_n$ — набор всех вершин из V, которые при проверке в строке 06 процедуры DFS оказались белыми, при этом будем считать, что при $i < j$ вершина r_i просматривается процедурой раньше, чем r_j. Процедура DFS_Visit(u) не изменяет значение $\pi[u]$, поэтому к моменту окончания поиска в глубину значение $\pi[r_i]$ не определено для всех $1 \leqslant i \leqslant n$. Все остальные вершины, т.е. вершины $v \in V \setminus \{r_1, \ldots, r_n\}$, открываются рекурсивным вызовом DFS_Visit(v) в строке 07 процедуры DFS_Visit, а значит, в строке 06 для них определяется предшественник $\pi[v]$.

Для каждого $1 \leqslant i \leqslant n$ обозначим через V_i множество вершин v, которые открываются в процессе работы процедуры DFS_Visit(r_i), т.е. $d[r_i] \leqslant d[v] < f[r_i]$. Заметим, что для $i < j$ в момент запуска процедуры DFS_Visit(r_j) все вершины из V_i уже окрашены в чёрный цвет, а в процессе работы DFS_Visit(r_j) открываются только белые вершины. Следовательно, $V_i \cap V_j = \varnothing$ для всех $i \neq j$. Таким образом, $\{V_1, \ldots, V_n\}$ является разбиением V на непересекающиеся подмножества, причём $r_i \in V_i$, $1 \leqslant i \leqslant n$.

Для каждого $1 \leqslant i \leqslant n$ положим $E_i = \{(\pi[v], v) \mid v \in V_i \setminus \{r_i\}\}$. Заметим, что для любой вершины $v \in V_i \setminus \{r_i\}$ в момент определения её предшественника $\pi[v]$ вершина $u = \pi[v]$ должна быть серой. Поскольку в этот момент все вершины из $V_1 \cup \cdots \cup V_{i-1}$ уже окрашены в чёрный цвет, а все вершины из $V_{i+1} \cup \cdots \cup V_n$ пока остаются белыми, заключаем, что $u \in V_i$. Таким образом, V_i замкнуто относительно предшественников, и значит, $E_i \subseteq V_i \times V_i$.

Обозначим $G_i = \langle V_i, E_i\rangle$, $1 \leqslant i \leqslant n$. Мы установили, что граф G_π разбивается на попарно несвязанные графы $G_1, \ldots, G_n$. Докажем, что для всех $1 \leqslant i \leqslant n$ граф G_i — связный. Для этого достаточно показать, что для любой вершины $v \in V_i$ в графе G_i существует путь из r_i в v. Из неравенства $d[r_i] \leqslant d[v] < f[r_i]$ по лемме 10 следует, что в графе G_π существует путь из r_i в v. Поскольку $v \in V_i$ и V_i замкнуто относительно предшественников, заключаем, что данный путь полностью содержится в графе G_i.

Итак, для всех $1 \leqslant i \leqslant n$ граф G_i связный и при этом $|E_i| = |V_i| - 1$. Следовательно, по теореме 4 графы $G_1, \ldots, G_n$ являются деревьями с корнями $r_1, \ldots, r_n$ соответственно, а их объединение, т.е. граф G_π, — лесом. □

Тот факт, что граф G_π является лесом, позволяет нам говорить о потомках и предках в деревьях данного леса.

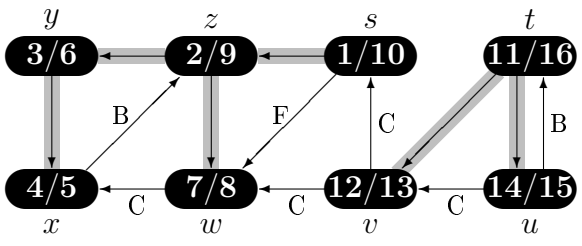
Лемма 12. *Пусть G_π — лес поиска, полученный в результате применения процедуры* DFS *к ориентированному или неориентированному графу $G = \langle V, E\rangle$, и пусть $u, v \in V$, $u \neq v$. Вершина v является потомком вершины u в лесу G_π тогда и только тогда, когда вершина u была серой в момент открытия вершины v.*

Доказательство. ($\Longrightarrow$) Поскольку $v \in V$, заключаем, что в одном из деревьев поиска с корнем r существует путь $\langle v_0, \ldots, v_k\rangle$ такой, что $v_0 = r$, $v_k = v$ и $v_i = \pi[v_{i+1}]$ для всех $i < k$. Достаточно показать, что в момент открытия вершины v все вершины $v_0, \ldots, v_{k-1}$ окрашены в серый цвет. Докажем это утверждение индукцией по значению i, изменяющемуся от $k-1$ до 1. Очевидно, в момент открытия вершины v её предшественник v_{k-1} должен иметь серый цвет. Допустим теперь, что в момент открытия v вершина v_{i+1} имеет серый цвет. Из условия $v_i = \pi[v_{i+1}]$ следует, что v_{i+1} была открыта при исполнении строки 07 процедуры DFS_Visit(v_i), при этом произошёл рекурсивный вызов процедуры DFS_Visit(v_{i+1}). Поскольку в рассматриваемый момент вершина v_{i+1} всё ещё серая, процедура DFS_Visit(v_{i+1}) ещё не завершила свою работу, и значит, процедура DFS_Visit(v_i) ещё не перешла к исполнению строки 08. Таким образом, вершина v_i в рассматриваемый момент тоже серая.

($\Longleftarrow$) Пусть в момент открытия вершины v вершина u имеет серый цвет. Следовательно, ранее была вызвана процедура DFS_Visit(u), которая в момент открытия вершины v ещё не закончила свою работу. Отсюда следует, что $d[u] < d[v] < f[u]$. В силу леммы 10 в графе G_π найдётся путь p из u в v. Тогда путь p полностью содержится в одном из деревьев поиска с некоторым корнем r. В этом дереве существует единственный путь из r в u, дополнив который путём p, мы получим путь из r в v, показывающий, что v является потомком u в данном дереве. □

Ещё одно важное свойство поиска в глубину заключается в том, что метки открытия и завершения образуют скобочную структуру. Если представить открытие вершины u в виде открывающейся скобки "$(u$", а завершение — в виде закрывающейся скобки "$u)$", то хронологический список открытий и завершений образует корректное выражение в смысле сбалансированности скобок.

Пример. Пусть результат поиска в глубину имеет вид, изображённый на рисунке.



Тогда соответствующее ему скобочное выражение имеет вид

$$\Big(s\big(z(y(xx)y)(ww)z\big)s\Big)\Big(t(vv)(uu)t\Big).$$

Теорема 13 (о скобках). *Пусть процедура* DFS *применяется к ориентированному или неориентированному графу* $G = \langle V, E\rangle$. *Тогда для любых различных вершин* $u, v \in V$ *выполняется ровно одно из следующих трёх утверждений:*

(1) Отрезки $[d[u], f[u]]$ *и* $[d[v], f[v]]$ *не пересекаются, и ни и не является потомком* v *в лесу поиска* G_π, *ни* v *не является потомком* u.

(2) Отрезок $[d[u], f[u]]$ *полностью содержится в отрезке* $[d[v], f[v]]$, *и* u *является потомком* v *в лесу поиска* G_π.

(3) Отрезок $[d[v], f[v]]$ *полностью содержится в отрезке* $[d[u], f[u]]$, *и* v *является потомком* u *в лесу поиска* G_π.

Доказательство. Из условия $u \neq v$ следует, что $d[u] < d[v]$ или $d[u] > d[v]$. Рассмотрим случай $d[u] < d[v]$, в котором имеется два подслучая, в зависимости от того, справедливо неравенство $d[v] < f[u]$ или нет. Если $d[v] < f[u]$, то вершина v была открыта, когда вершина u была окрашена в серый цвет. Отсюда по лемме 12 заключаем, что v является потомком вершины u в лесу G_π. Кроме этого, поскольку вершина v открыта позже, чем u, перед тем как вернуться для завершения поиска к вершине u, процедурой DFS будут исследованы все выходящие из v рёбра. В таком случае отрезок $[d[v], f[v]]$ полностью содержится в отрезке $[d[u], f[u]]$, т.е. выполняется утверждение (3). Если же $f[u] < d[v]$, то отрезки $[d[u], f[u]]$ и $[d[v], f[v]]$ не пересекаются, а значит, когда открывается одна их вершин u или v, другая имеет белый или чёрный цвет. Следовательно, в силу леммы 12, ни одна из данных вершин не является потомком другой, т.е. выполняется утверждение (1).

Случай $d[u] > d[v]$ рассматривается аналогично. Здесь будет справедливо либо утверждение (1), либо утверждение (2). □

Поиск в глубину можно использовать для классификации рёбер входного графа.

Определение. Пусть G_π — лес поиска, полученный в результате применения процедуры DFS к ориентированному или неориентированному графу $G = \langle V, E\rangle$. Определим следующие четыре типа рёбер графа G:

(а) *Рёбра деревьев* — это рёбра графа G_π.

(б) *Обратные рёбра* — это рёбра $(u, v) \in E$, соединяющие вершину u с её предком v в дереве поиска. Рёбра-петли, которые могут встречаться в ориентированных графах, считаются обратными рёбрами.

(в) *Прямые рёбра* — это рёбра $(u, v) \in E$, не являющиеся рёбрами деревьев и соединяющие вершину u с её потомком v в дереве поиска.

(г) *Перекрёстные рёбра* — все остальные рёбра. Они могут соединять вершины из разных деревьев поиска или вершины одного и того же дерева поиска, когда ни одна из вершин не является предком другой.

Замечание. В неориентированных графах при классификации рёбер может возникнуть неоднозначность, связанная с тем, что алгоритм просматривает каждое ребро дважды: в виде (u, v) и (v, u). В таком случае, когда ребро удовлетворяет нескольким категориям из списка (а)–(г), оно классифицируется в соответствии с *первой* категорией в списке, применимой для данного ребра. Приняв данное соглашение, можно

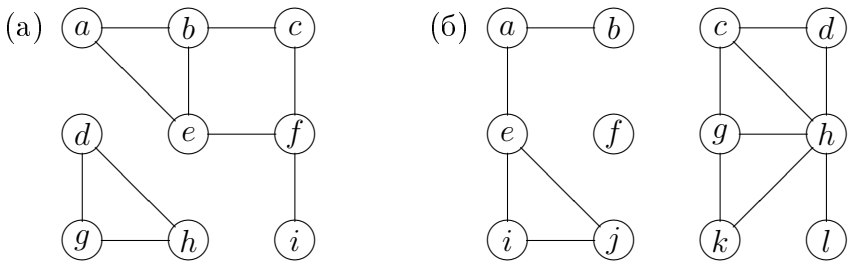
доказать, что при любом поиске в глубину в неориентированном графе любое ребро является либо ребром дерева, либо обратным ребром. Тот же результат получается, если выполнять классификацию в соответствии с тем, в каком именно виде — (u, v) или (v, u) — ребро встречается в первый раз в процессе поиска.

Время работы алгоритма

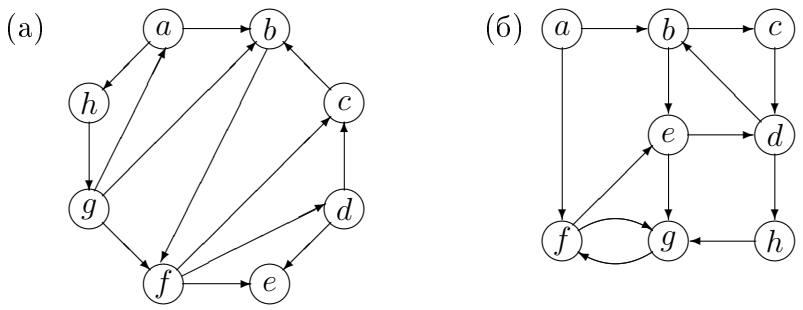
Циклы в строках 01–03 и 05–07 процедуры DFS выполняются за время $O(|V|)$, исключая время, необходимое для вызова процедуры DFS_Visit. Заметим, что процедура DFS_Visit вызывается ровно по одному разу для каждой вершины $u \in V$, поскольку она вызывается только для белых вершин, и первое, что она делает, — это окрашивает переданную в качестве параметра вершину в серый цвет. Количество итераций цикла **for** в строках 04–07 процедуры DFS_Visit(u) совпадает с количеством рёбер графа, выходящих из вершины u. Суммарно каждое ребро графа просматривается один (для ориентированных графов) или два раза (для неориентированных графов). Поэтому общее время выполнения строк 04–07 процедуры DFS_Visit(u) для всех вершин u составляет $O(|E|)$. Таким образом, время работы процедуры DFS равно $O(|V| + |E|)$, т.е. оно линейно.

Упражнения

1. Применить алгоритм поиска в глубину к следующим неориентированным графам, перебирая вершины в алфавитном порядке. Пометить каждое ребро как ребро дерева или обратное ребро. Написать скобочное выражение, соответствующее данному поиску в глубину.



2. Применить алгоритм поиска в глубину к следующим ориентированным графам, перебирая вершины в алфавитном порядке. Пометить каждое ребро как ребро дерева, прямое, обратное или перекрёстное. Написать скобочное выражение, соответствующее данному поиску в глубину.



3. Используя процедуру DFS, разработать алгоритм, формализующий метод прохождения лабиринта с помощью клубка ниток и мела. (Указание: представить лабиринт в виде неориентированного графа, вершины которого соответствуют помещениям лабиринта, а рёбра — проходам между соседними помещениями.)

4. Доказать, что при поиске в глубину в неориентированном графе G любое ребро G является либо ребром дерева, либо обратным ребром.

5. Доказать, что при поиске в глубину ребро (u, v) является:
(а) ребром дерева или прямым ребром, если и только если $d[u]<d[v]<f[v]<f[u]$;
(б) обратным ребром, если и только если $d[v] \leqslant d[u] < f[u] \leqslant f[v]$;
(в) перекрёстным ребром, если и только если $d[v] < f[v] < d[u] < f[u]$.

6. Привести пример ориентированного графа G и двух его вершин u и v таких, что граф G содержит путь из u в v, и $d[u] < d[v]$ при некотором поиске в глубину в G, но вершина v не является потомком вершины u в лесу поиска.

7. Модифицировать псевдокод процедур DFS и DFS_Visit так, чтобы поиск в глубину расставлял метки рёбер графа в соответствии с их типами.

§ 6. Алгоритм Дейкстры

Данный параграф посвящён алгоритму решения *задачи о кратчайшем пути*, которая имеет естественную постановку на практике. Допустим, имеется карта дорог, на которой указаны различные населённые пункты, некоторые из них связаны дорогами. При этом на карте приведены расстояния между парами соседних пунктов, связанных дорогой. Необходимо найти кратчайший маршрут из одного пункта в другой. Карту дорог можно смоделировать в виде графа, вершины которого представляют населённые пункты, а рёбра — отрезки дорог между соседними пунктами, причём каждому ребру приписан его вес, равный расстоянию между соответствующими пунктами. Вес рёбер можно интерпретировать не как расстояние, а как временны́е интервалы, стоимость, штрафы или как любую другую величину, которая линейно накапливается по мере продвижения вдоль рёбер графа и которую необходимо оптимизировать.

Для математической формулировки задачи введём следующие определения.

Определение. Граф $G = \langle V, E\rangle$ называют *взвешенным*, если на множестве его рёбер задана *весовая функция* $w : E \to \mathbb{R}$, сопоставляющая каждому ребру $(u, v) \in E$ некоторое действительное число $w(u, v) \in \mathbb{R}$.

Определение. Пусть $G = \langle V, E\rangle$ — взвешенный ориентированный граф с неотрицательной весовой функцией $w : E \to \mathbb{R}$, т.е. $w(u, v) \geqslant 0$ для каждого ребра $(u, v) \in E$.

Весом $w(p)$ *пути* $p = \langle v_0, v_1, \ldots, v_k\rangle$ в графе G называется суммарный вес входящих в него рёбер, т.е.

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i).$$

(При $k = 0$ считаем, что путь $p = \langle v_0\rangle$ имеет вес $w(p) = 0$.)

Для произвольных $u, v \in V$ определим *вес* $\delta(u, v)$ *кратчайшего пути из вершины* u *к вершине* v в графе G следующим образом:

(а) Если вершина v достижима из вершины u в графе G, то полагаем

$$\delta(u, v) = \min\{w(p) \mid p \text{ — путь из } u \text{ к } v \text{ в графе } G\}.$$

При этом путь p из u к v в графе G назовём *кратчайшим*, если $w(p) = \delta(u, v)$.

(б) Если же вершина v не достижима из u в графе G, положим $\delta(u, v) = \infty$.

В данном параграфе рассматривается *задача о кратчайшем пути из одной вершины*. В этой задаче для заданного взвешенного графа $G = \langle V, E\rangle$ требуется найти кратчайший путь из определённой *исходной вершины* $s \in V$ в любую другую из вершин $v \in V$.

Отметим, что алгоритм поиска в ширину, описанный ранее, представляет собой алгоритм поиска кратчайшего пути из исходной вершины по невзвешенному графу, т.е. графу, каждому ребру которого приписывается единичный вес. Для взвешенных графах, вообще говоря, необходим другой алгоритм.

Описание алгоритма

Алгоритм Дейкстры решает задачу о кратчайшем пути из одной вершины s во взвешенном графе $G = \langle V, E\rangle$ в том случае, когда веса рёбер не отрицательны. Алгоритм не только находит кратчайший путь до каждой достижимой из s вершины, но и вычисляет вес кратчайшего пути.

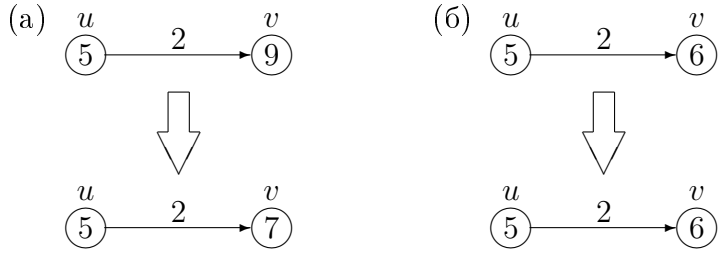
Как и в поиске в ширину, для представления кратчайших путей используется переменная $\pi[v]$, которая называется *предшественником* вершины $v \in V$. Функция предшественников π позволяет проследить кратчайшие пути из исходной вершины s в остальные вершины графа, достижимые из s. Однако до тех пор, пока алгоритм не закончил свою работу, значения π не обязательно указывают кратчайшие пути. В начальный момент значение $\pi[v]$ не определено для всех $v \in V$.

Для каждой вершины $v \in V$ также поддерживается переменная $d[v]$ такая, что $\delta(s, v) \leqslant d[v]$. Переменную $d[v]$ назовём *оценкой кратчайшего пути*. Считается, что в момент запуска алгоритма имеет место $d[s] = 0$ и $d[v] = \infty$ для всех $v \in V \setminus \{s\}$.

В алгоритме используется процесс *ослабления (релаксации)* рёбер, который мы опишем в виде отдельной процедуры. Ослабление ребра $(u, v) \in E$ заключается в проверке, нельзя ли улучшить найденный до сих пор кратчайший путь к вершине v, проведя его через вершину u, а также в обновлении значений $d[v]$ и $\pi[v]$ при наличии такой возможности улучшения. Ослабление может уменьшить оценку кратчайшего пути $d[v]$ и обновить предшественника $\pi[v]$ для вершины v. Приведённая ниже процедура выполняет ослабление ребра (u, v).

```
RELAX(u, v, w)
01  if d[v] > d[u] + w(u, v)
02      then d[v] ← d[u] + w(u, v)
03           π[v] ← u
```

Пример. На рисунке приведены два примера ослабления ребра. В вершинах указаны оценки кратчайших путей, а рёбра помечены их весами. На рисунке слева оценка кратчайшего пути понижается, на рисунке справа не происходит никаких изменений.



Ослабление — единственная операция в алгоритме, которая изменяет оценки кратчайших путей и предшественников, причём каждое ребро ослабляется не более одного раза.

В алгоритме присутствует множество вершин S, для которых уже вычислены окончательные веса кратчайших путей к ним из истока s. В вычислениях поочерёдно выбирается вершина $u \in V \setminus S$, которая на данный момент имеет минимальную оценку кратчайшего пути. После добавления этой вершины u в множество S производится ослабление всех рёбер, исходящих из u.

Множество $Q = V \setminus S$ представляется в виде *очереди с приоритетами*, в которой вершины v с меньшим значением $d[v]$ имеют более высокий приоритет, т.е. множеству Q сопоставляется упорядоченный набор $\langle v_1, \ldots, v_r \rangle$, состоящий из всех элементов Q, в котором $d[v_i] \leqslant d[v_{i+1}]$ для всех $i < r$. Алгоритм работает до тех пор, пока в Q не останется вершин.

Алгоритм Дейкстры имеет следующую реализацию в виде псевдокода.

DIJKSTRA(G, w, s)
{Вход: ориентированный граф $G = \langle V, E \rangle$, неотрицательная весовая функция $w : E \to \mathbb{R}$, вершина $s \in V$.}
{Выход: оценка кратчайшего пути $d[u]$ для всех $u \in V$, предшественник $\pi[u]$ для некоторых $u \in V$.}

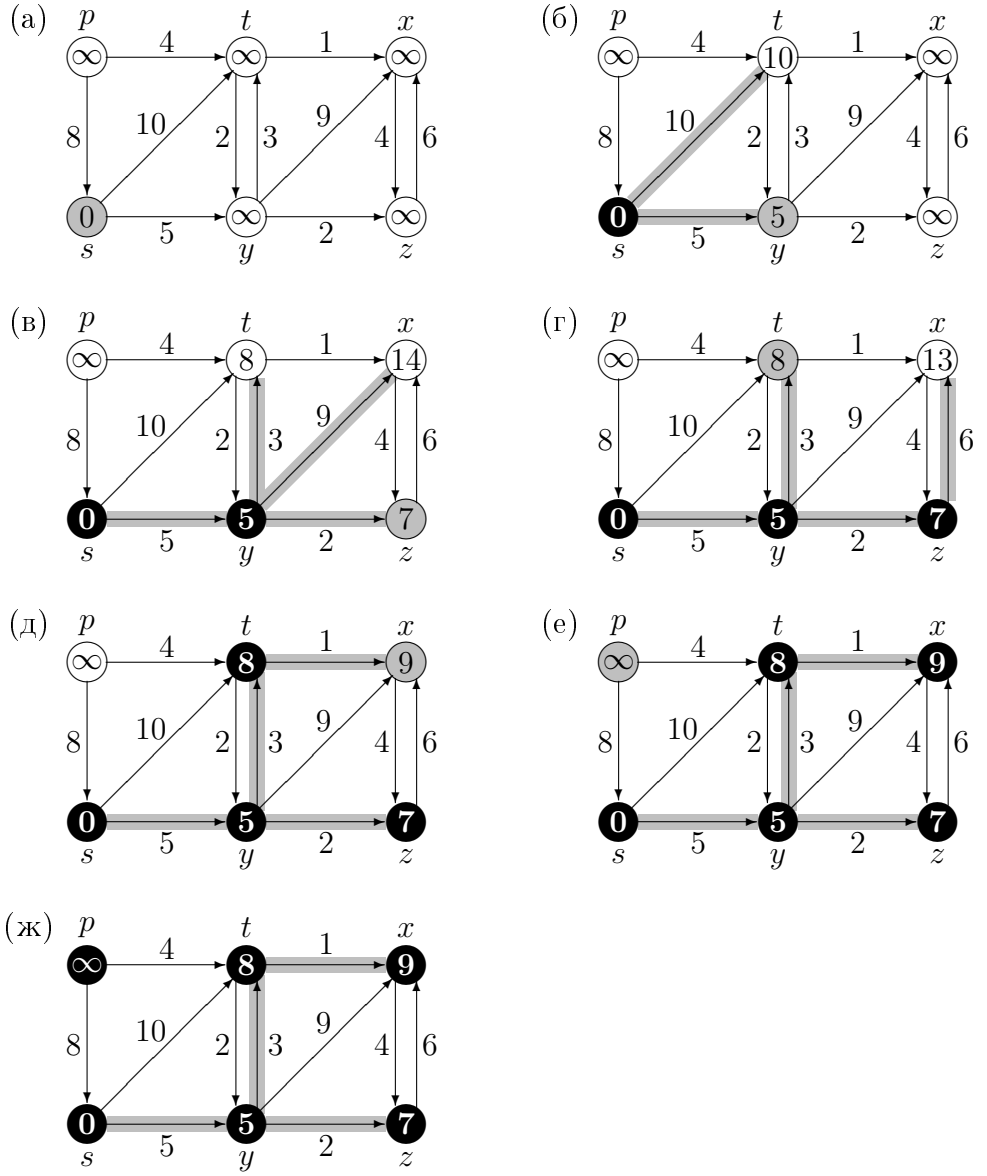
```
01  for каждой вершины v ∈ V
02      do d[v] ← ∞
03         π[v] ↑
04  d[s] ← 0
05  S ← ∅
06  Q ← V
07  while Q ≠ ∅
08        do u ← EXTRACT_MIN(Q)
09           S ← S ∪ {u}
10           for каждого ребра (u, v) ∈ E
11               do RELAX(u, v, w)
```

Процедура DIJKSTRA работает следующим образом. В строках 01–04 происходит инициализация величин d и π: значения $\pi[v]$ не определены для всех вершин $v \in V$, $d[s] = 0$ и $d[v] = \infty$ для всех $v \in V \setminus \{s\}$. В строках 05–06 инициализируются пустое множество вершин S и очередь с приоритетами Q, которая в начальный момент содержит все вершины из V. В алгоритме Дейкстры перед каждой итерацией цикла **while** (строки 07–11) выполняется тождество $Q = V \setminus S$. В строках 08–09 цикла **while** из множества Q извлекается вершина u с минимальным текущим значением оценки $d[u]$, и затем u добавляется в множество S. В строках 10–11 ослабляются все рёбра (u, v), исходящие из u. Отметим, что количество итераций цикла **while** в строках 07–11 равно $|V|$.

Результаты работы процедуры DIJKSTRA могут зависеть от порядка просмотра рёбер в строке 10, а также от порядка извлечения вершин из очереди в строке 08 в случае, когда в Q имеется более одной вершины с минимальным текущим значением оценки кратчайшего пути.

Пример. Ниже проиллюстрирован пример работы процедуры DIJKSTRA с исходной вершиной s. В каждой вершине приведена текущая оценка кратчайшего пути к ней. Выделенные рёбра указывают предшественников. Чёрным цветом выделены верши-

ны, добавленные в множество S, серым — вершина, которая выбирается в качестве вершины u в строке 08, а белым — остальные содержащиеся в множестве $Q = V \setminus S$ вершины. На рисунках (а)–(е) проиллюстрированы ситуации, сложившиеся перед очередной итерацией цикла **while**. На рисунке (ж) приведены окончательные значения величин d и π.



Свойства алгоритма

Для доказательства корректности работы алгоритма Дейкстры необходимы некоторые свойства оценок кратчайших путей и ослабления рёбер.

Лемма 14. *В процессе выполнения процедуры* DIJKSTRA(G, w, s) *для всех вершин* $v \in V$ *всегда справедливо неравенство* $\delta(s, v) \leqslant d[v]$, *а после того как величина* $d[v]$ *становится равной* $\delta(s, v)$, *она больше не изменяется.*

Доказательство. Докажем справедливость неравенства $\delta(s, v) \leqslant d[v]$ индукцией по числу применений процедуры RELAX.

Непосредственно после инициализации справедливо $\delta(s, s) = d[s] = 0$ и $\delta(s, v) \leqslant$ $\leqslant d[v] = \infty$ для всех $v \in V \setminus \{s\}$.

Пусть перед ослаблением некоторого ребра $(u, v) \in E$ неравенство $\delta(s, x) \leqslant d[x]$ справедливо для всех $x \in V$. Единственное значение d, которое может измениться после ослабления ребра (u, v), — это значение $d[v]$. Если оно изменяется, то, в силу индукционного предположения, получаем

$$\delta(s, u) + w(u, v) \leqslant d[u] + w(u, v) = d[v].$$

Перед ослаблением ребра (u, v) выполняется неравенство $d[v] > d[u] + w(u, v)$. Следовательно, значение $d[u]$ конечно, а значит, и значение $\delta(s, u)$ тоже конечно. Поэтому в графе G существует кратчайший путь p из s в u. Дополнив путь p ребром (u, v), получим путь p' из s в v с весом $\delta(s, u) + w(u, v)$. Вес пути p' не может быть меньше веса кратчайшего пути из s в v, откуда заключаем, что

$$\delta(s, v) \leqslant \delta(s, u) + w(u, v) \leqslant d[v].$$

Допустим, в некоторый момент становится справедливым равенство $d[v] = \delta(s, v)$. После этого момента значение $d[v]$ не может уменьшиться, т.к. иначе нарушится неравенство $d[v] \geqslant \delta(s, v)$. С другой стороны, значение $d[v]$ не может увеличиться, т.к. ослабление ребёр не увеличивает значений d. Таким образом, после достижения равенства $d[v] = \delta(s, v)$ значение $d[v]$ не изменяется. □

Следствие 15. *Если вершина v не достижима из s в графе G, то в процессе выполнения процедуры* DIJKSTRA(G, w, s) *всегда верно тождество $d[v] = \delta(s, v) = \infty$.*

Доказательство. Следует из леммы 14, поскольку для недостижимой из s вершины v всегда справедливо $\infty = \delta(s, v) \leqslant d[v]$. □

Лемма 16. *Пусть $p = \langle s, \ldots, u, v\rangle$ — кратчайший в графе G путь из s к v, последним ребром которого является ребро $(u, v) \in E$, и пусть в процессе выполнения процедуры* DIJKSTRA(G, w, s) *происходит ослабление ребра (u, v). Если в некоторый момент времени до ослабления ребра (u, v) выполняется $d[u] = \delta(s, u)$, то в любой момент времени после ослабления ребра (u, v) выполняется $d[v] = \delta(s, v)$.*

Доказательство. Поскольку $p = \langle s, \ldots, u, v\rangle$ — кратчайший путь из s к v, то, удалив из него ребро (u, v), мы получим путь $p' = \langle s, \ldots, u\rangle$, который является кратчайшим из s к u. Отсюда следует тождество

$$\delta(s, u) + w(u, v) = \delta(s, v).$$

Пусть выполняется $d[u] = \delta(s, u)$ в некоторый момент времени до ослабления ребра (u, v). По лемме 14 равенство $d[u] = \delta(s, u)$ остаётся справедливым и впоследствии. Если непосредственно перед данным ослаблением выполняется неравенство $d[v] > d[u] + w(u, v)$, то после этой операции оно преобразуется в равенство $d[v] = d[u] + w(u, v)$. Если же непосредственно перед ослаблением выполняется неравенство $d[v] \leqslant d[u] + w(u, v)$, то в процессе ослабления ничего не изменится и неравенство сохранится. В любом случае сразу после ослабления ребра (u, v) имеет место соотношение

$$d[v] \leqslant d[u] + w(u, v) = \delta(s, u) + w(u, v) = \delta(s, v).$$

С другой стороны, по лемме 14 справедливо $d[v] \geqslant \delta(s, v)$. Таким образом, сразу после указанного ослабления выполняется равенство $d[v] = \delta(s, v)$, которое впоследствии сохраняется. □

Как и при поиске в ширину, окончательные значения функции предшественников однозначно определяют дерево кратчайших путей.

Определение. Пусть π — функция предшественников, вычисленная в результате выполнения процедуры Dijkstra(G, w, s). Определим неориентированный граф $G_\pi = \langle V_\pi, E_\pi \rangle$, где

$$V_\pi = \{s\} \cup \{v \in V \mid \pi[v] \downarrow\},$$
$$E_\pi = \{(\pi[v], v) \mid v \in V_\pi \setminus \{s\}\}.$$

Граф G_π является деревом (доказательство этого факта приведено ниже в теореме 17) и называется *деревом кратчайших путей.*

Теорема 17 (о корректности алгоритма Дейкстры)**.** *Пусть $G = \langle V, E \rangle$ — взвешенный ориентированный граф с неотрицательной весовой функцией $w : E \to \mathbb{R}$ и пусть процедура* Dijkstra *выполняется над графом G с исходной вершиной $s \in V$. Тогда справедливы следующие утверждения:*

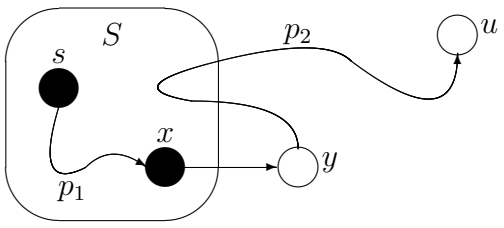
(1) По окончании работы процедуры Dijkstra(G, w, s) *для всех $u \in V$ выполняется $d[u] = \delta(s, u)$.*

(2) Граф G_π является деревом, состоящим из всех вершин, достижимых из s в графе G.

(3) Для всех $v \in V_\pi$ единственный простой путь из s в v в дереве G_π совпадает с кратчайшим путём из s в v в графе G.

Доказательство. (1) Докажем, что для каждой вершины $u \in V$ в момент её добавления в множество S выполняется равенство $d[u] = \delta(s, u)$. Допустим, напротив, для некоторой вершины u это утверждение неверно. Пусть u — первая добавленная в множество S вершина, для которой в момент добавления $d[u] \neq \delta(s, u)$. Ясно, что $u \neq s$, поскольку в момент добавления вершины s в множество S выполняется $d[s] = \delta(s, s) = 0$. Из условия $u \neq s$ следует, что непосредственно перед добавлением u в множество S оно не является пустым.

В графе G вершина u должна быть достижимой из вершины s, т.к. в противном случае, в силу следствия 15, всегда выполняется равенство $d[u] = \delta(s, u) = \infty$, которое противоречит выбору вершины u. Поскольку u достижима из s, заключаем, что в графе G существует кратчайший путь p из s в u. Непосредственно перед добавлением u в множество S путь p соединяет вершину из S (вершину s) с вершиной из $V \setminus S$ (вершиной u). Следовательно, можно выбрать первую вершину y на пути p, принадлежащую $V \setminus S$, и значит, ей предшествует некоторая вершина $x \in S$ (см. рисунок). Тогда путь p можно разложить на три составляющие: путь p_1, ведущий из s в x, ребро (x, y) и путь p_2, ведущий из y в u. (Любой из путей p_1 и p_2 может не содержать ни одного ребра. Путь p_2 может возвращаться в множество S.)



Поскольку вершина x попала в множество S раньше, чем u, а вершина u выбрана как первая вершина, после добавления которой в S выполняется соотношение $d[u] \neq \delta(s,u)$, заключаем, что после добавления x в множество S выполняется равенство $d[x] = \delta(s,x)$. В этот момент (т.е. сразу после добавления x в S) происходит ослабление ребра (x,y). Отсюда по лемме 16 следует, что в любой момент времени после ослабления ребра (x,y) справедливо $d[y] = \delta(s,y)$.

Поскольку на кратчайшем пути p вершина y находится перед вершиной u и вес каждого из рёбер неотрицателен, выполняется неравенство $\delta(s,y) \leqslant \delta(s,u)$. Следовательно, используя лемму 14, получаем, что в момент добавления вершины u в множество S выполняются соотношения

$$d[y] = \delta(s,y) \leqslant \delta(s,u) \leqslant d[u]. \qquad (*)$$

Однако, поскольку и вершина u, и вершина y в момент выбора вершины u в строке 08 находились в множестве $V \setminus S$, заключаем, что в этот момент выполняется неравенство $d[u] \leqslant d[y]$. Таким образом, оба неравенства в $(*)$ являются равенствами, т.е. в момент добавления вершины u в множество S выполняется

$$d[y] = \delta(s,y) = \delta(s,u) = d[u].$$

Следовательно, $d[u] = \delta(s,u)$, что противоречит выбору вершины u.

Итак, для каждой вершины $u \in V$ равенство $d[u] = \delta(s,u)$ выполняется в момент её добавления в множество S, а значит, в силу леммы 14, оно выполняется и в дальнейшем. Поскольку любая вершина $u \in V$ попадает в множество S, заключаем, что по окончании работы процедуры $\textsc{Dijkstra}(G,w,s)$ равенство $d[u] = \delta(s,u)$ выполняется для всех $u \in V$.

(2) Пусть $v \neq s$. Условие $v \in V_\pi$ выполняется тогда и только тогда, когда значение $\pi[v]$ становится определённым в процессе работы процедуры $\textsc{Dijkstra}(G,w,s)$, что, в свою очередь, происходит тогда и только тогда, когда значение $d[v]$ становится конечным после ослабления некоторого ребра $(u,v) \in E$. Поскольку после окончания работы процедуры $\textsc{Dijkstra}(G,w,s)$ выполняется $d[v] = \delta(s,v)$, заключаем, что $v \in V_\pi$ тогда и только тогда, когда $\delta(s,v) < \infty$, т.е. v достижима из s в графе G. Заметим также, что при окончательном определении значения $\pi[v] = u$ происходит окончательное присваивание $d[v] = d[u] + w(u,v)$, и значит, в силу соотношения $d[v] = \delta(s,v) < \infty$, отсюда следует, что $d[u] = \delta(s,u) < \infty$, т.е. $u \in V_\pi$, а ребро $(\pi[v],v) \in E_\pi$ действительно соединяет вершины из V_π.

Таким образом, граф G_π определён корректно и состоит из всех вершин, достижимых из s в G. Тогда G_π — связный граф, поскольку любые две его вершины связаны путями с s. Кроме этого, из определения графа G_π видно, что $|E_\pi| = |V_\pi| - 1$. Следовательно, в силу теоремы 4, граф G_π является деревом.

(3) Пусть $v \in V_\pi$. По теореме 4 в дереве G_π существует единственный простой путь $p_v = \langle v_0, \ldots, v_k \rangle$ от корня $s = v_0$ к вершине $v = v_k$. Как и в случае алгоритма поиска в ширину, можно доказать, что для всех $1 \leqslant i \leqslant k$ справедливо свойство $v_{i-1} = \pi[v_i]$. Индукцией по длине k пути p_v докажем, что этот путь одновременно является кратчайшим путём из s в v в графе G. Если $k = 0$, то путь, состоящий из одной вершины s, очевидно, является кратчайшим в G. Если же $k > 0$, то $v \neq s$, и значит, определён предшественник $\pi[v] = u$. Отсюда, в силу условий $v = v_k$ и $v_{k-1} = \pi[v_k]$, вытекает, что $u = v_{k-1}$. Ясно, что путь $p_u = \langle v_0, \ldots, v_{k-1} \rangle$ является простым путём от s к u в дереве G_π. Следовательно, в силу индукционного предположения, p_u является кратчайшим путём из s в u в графе G, и значит, для веса пути p_u выполняется тождество

$w(p_u) = \delta(s, u)$. Тогда вес пути p_v вычисляется как $w(p_v) = \delta(s, u) + w(u, v)$. Заметим, что в момент определения окончательного значения $\pi[v]$ одновременно определяется окончательное значение $d[v] = d[u] + w(u, v)$, причём в этот же момент значение $d[u]$ тоже уже стабилизировалось. Отсюда следует, что $\delta(s, v) = \delta(s, u) + w(u, v)$. Тогда $w(p_v) = \delta(s, v)$, откуда заключаем, что p_v — кратчайший путь из s в v в графе G. □

Время работы алгоритма

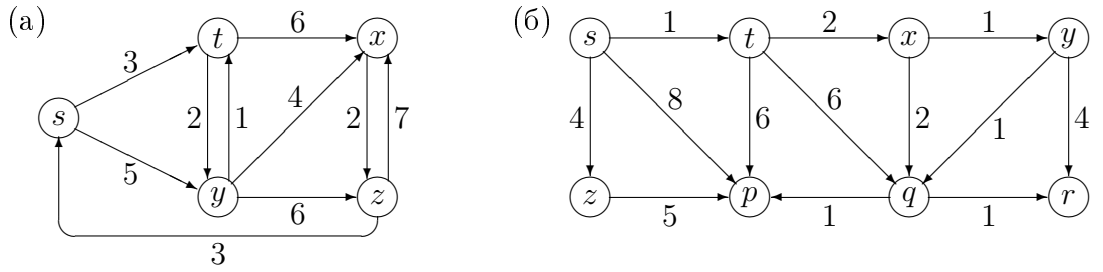
Инициализация в строках 01–05 требует время, равное $O(|V|)$. Дальнейшее время работы алгоритма зависит от способа реализации очереди с приоритетами. Самое простое — хранить значения $d[v]$ в неупорядоченном *массиве* с индексами от 1 до $|V|$, как-то отмечая удалённые значения d. Тогда инициализация очереди в строке 06 расходует время, равное $O(|V|)$. Количество итераций цикла **while** в строках 07–11 равно $|V|$. Каждая итерация этого цикла начинается с извлечения из очереди вершины u, которое осуществляется с помощью процедуры EXTRACT_MIN. Один вызов процедуры EXTRACT_MIN требует время, равное $O(|V|)$, поскольку в ней происходит поиск минимума по всему массиву значений d. Далее каждое ребро $(u, v) \in E$ обрабатывается в цикле **for** в строках 10–11 ровно по одному разу на протяжении всей работы алгоритма. Поэтому суммарно выполняется $|E|$ итераций цикла **for**, и на каждой итерации вызывается процедура RELAX, исполняемая за время $O(1)$. Таким образом, полное время работы алгоритма Дейкстры составляет $O(|V| + |V|^2 + |E|) = O(|V|^2)$, т.е. время квадратично.

Замечание. Реализация очереди с приоритетами в виде массива предпочтительнее в случае *плотного* графа, т.е. когда число рёбер достаточно велико. Если граф *разреженный*, т.е. число его рёбер достаточно мало, быстрее работают реализации очереди с приоритетами в виде *бинарной неубывающей пирамиды*, или *пирамиды Фибоначчи* [1, 7].

Замечание. В некоторых задачах поиска кратчайшего пути во взвешенном графе веса рёбер могут принимать отрицательные значения. Для решения подобных задач также существуют алгоритмы, с которыми можно ознакомиться в [1, 7].

Упражнения

1. Применить алгоритм Дейкстры к следующим ориентированным графам, используя в качестве истока вершину s. Указать значения d и π и пометить вершины, входящие в множество S, после каждой итерации цикла **while**.



2. Привести пример ориентированного графа, содержащего рёбра с отрицательным весом, для которого алгоритм Дейкстры даёт неправильные ответы при поиске кратчайших путей.

3. Пусть во взвешенном ориентированном графе G веса рёбер, выходящих из некоторого истока s, могут быть отрицательными, веса всех других рёбер неотрицательные, и G не содержит циклов отрицательного веса. Доказать, что в таком графе алгоритм Дейкстры корректно находит кратчайшие пути из истока s.

§ 7. Алгоритм Крускала

При разработке электронных схем часто возникает необходимость электрически связать несколько компонентов в одну схему, соединяя контакты компонентов попарно проводом. Обычно желательно получить компоновку, использующую минимальное количество провода.

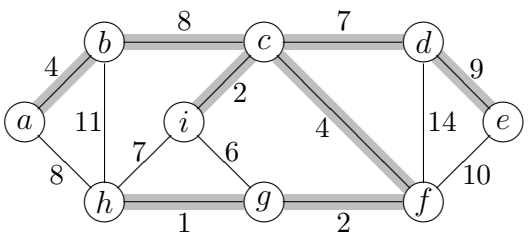
Эта задача моделируется на связном неориентированном графе $G = \langle V, E\rangle$ с весовой функцией $w : E \to \mathbb{R}$, где V — множество контактов, E — множество возможных соединений между парами контактов, а вес $w(u, v)$ ребра $(u, v) \in E$ интерпретируется как количество необходимого провода для соединения u и v.

Ясно, что оптимальная компоновка не может содержать циклов, поскольку удаление ребра из цикла снижает расход провода без потери связности схемы. Таким образом, математическая задача состоит в поиске ациклического подмножества $T \subseteq E$, которое соединяет все вершины V и чей общий вес

$$w(T) = \sum_{(u,v)\in T} w(u, v)$$

минимален. Поскольку множество T ациклическое и связывает все вершины V, граф $\langle V, T\rangle$ должен быть деревом, которое называют *остовным деревом* графа G, а задачу поиска T с минимальным весом именуют *задачей поиска минимального остовного дерева* (minimum-spanning-tree problem).

Пример. На рисунке показан пример взвешенного связного неориентированного графа и его минимального остовного дерева. На рёбрах указан их вес, а рёбра минимального остовного дерева выделены цветом. Общий вес данного дерева равен 37. Приведённое дерево не единственное: удалив ребро (b, c) и заменив его ребром (a, h), мы получим другое остовное дерево с тем же весом 37.



Убедимся в том, что задача нахождения минимального остовного дерева в связном неориентированном графе разрешима.

Предложение 18. *Для любого взвешенного связного неориентированного графа $G = \langle V, E\rangle$ существует минимальное остовное дерево.*

Доказательство. Выберем произвольную вершину $s \in V$ и применим к графу G с исходной вершиной s алгоритм поиска в ширину — в результате получим дерево поиска $G_\pi = \langle V_\pi, E_\pi \rangle$. По теореме 9 множество V_π состоит из всех вершин, достижимых из s в G. Поскольку граф G связный, отсюда следует, что $V_\pi = V$. Следовательно, дерево G_π связывает все вершины исходного графа G, т.е. G_π является остовным деревом для G. Таким образом, существует хотя бы одно остовное дерево. Граф G может иметь только конечное число различных остовных деревьев, а значит, среди них всегда можно выбрать дерево с минимальным общим весом. □

Описание алгоритма

Алгоритм Крускала решает задачу поиска минимального остовного дерева во взвешенном связном неориентированном графе $G = \langle V, E \rangle$. Алгоритм работает с множеством рёбер $A \subseteq E$, которое растёт путём добавления к нему рёбер по одному. Перед каждой очередной итерацией A является подмножеством некоторого множества T, образующего минимальное остовное дерево. Алгоритм определяет такое ребро (u, v), что после его добавления множество $A \cup \{(u, v)\}$ по-прежнему остаётся подмножеством (вообще говоря, другого) множества T', тоже образующего минимальное остовное дерево. Ребро с таким свойством называется *безопасным* для A.

В любой момент выполнения алгоритма граф $G_A = \langle V, A \rangle$ представляет собой лес, каждый связный компонент которого является деревом. Изначально множество A пусто, а лес G_A состоит из $|V|$ одноэлементных деревьев. Алгоритм Крускала находит безопасное ребро для добавления в растущий лес в результате поиска в G ребра (u, v) с минимальным весом среди всех рёбер, соединяющих два дерева в текущем лесу G_A. Каждая итерация уменьшает количество деревьев в лесу G_A на единицу. Когда в лесу G_A останется только одно дерево, алгоритм завершает работу.

В алгоритме используется разбиение множества V на непересекающиеся подмножества, каждое из которых состоит из вершин дерева в текущем лесу. Операция Find_Set(u) однозначно определяет, в каком из подмножеств содержится вершина u. Таким образом, чтобы установить, принадлежат ли две вершины u и v одному и тому же дереву, достаточно проверить равенство Find_Set(u) и Find_Set(v). Объединение деревьев ребром (u, v) выполняется при помощи процедуры Union(u, v).

Алгоритм Крускала имеет следующую реализацию в виде псевдокода.

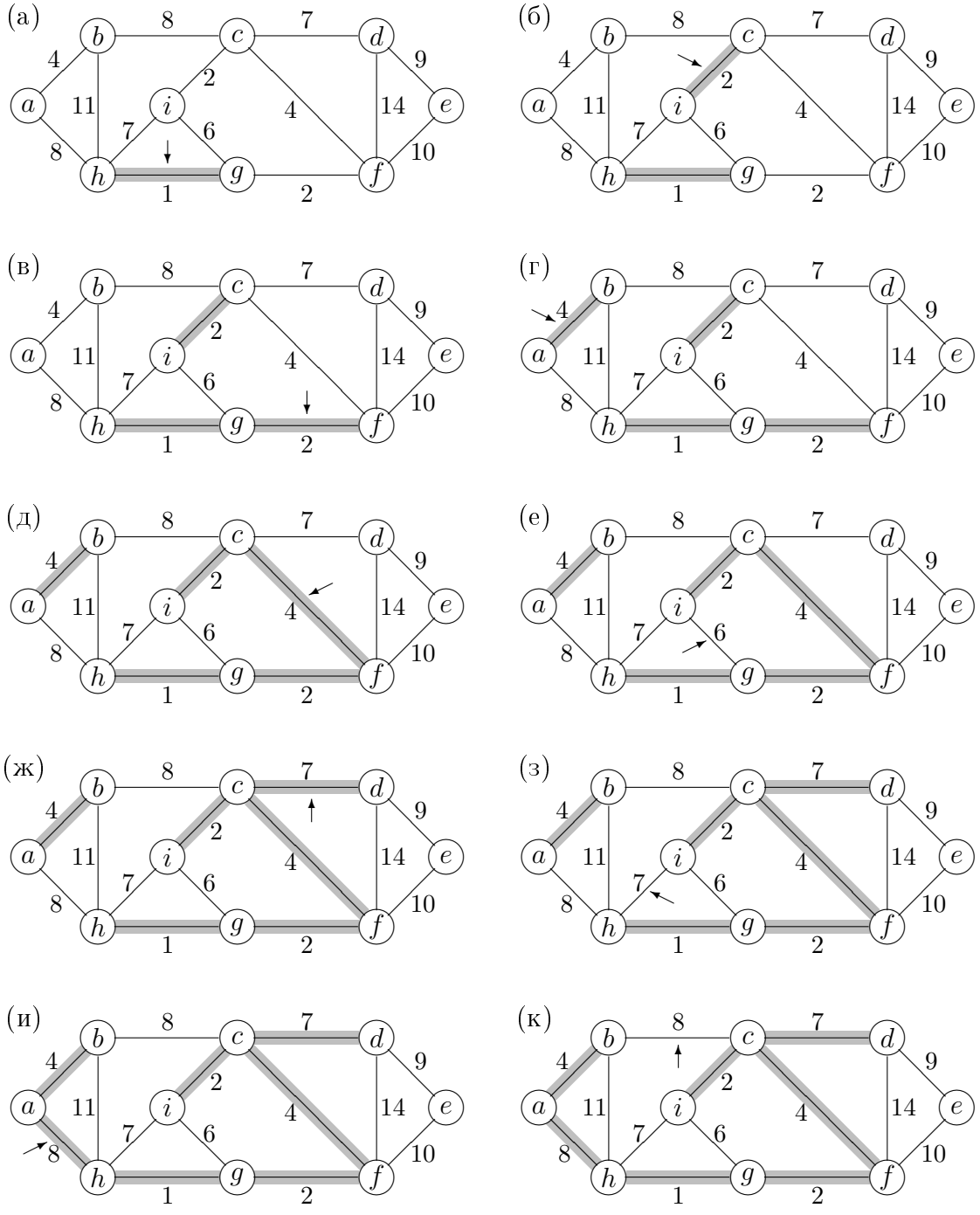
```
Kruskal(G, w)
{Вход: связный неориентированный граф G = ⟨V, E⟩, весовая функция w : E → ℝ.}
{Выход: множество A ⊆ E такое, что ⟨V, A⟩ — минимальное остовное дерево.}
01  A ← ∅
02  for каждой вершины v ∈ V
03      do Make_Set(v)
04  Сортируем рёбра из E в неубывающем порядке их весов w
05  for каждого ребра (u, v) ∈ E (в порядке возрастания веса)
06      do if Find_Set(u) ≠ Find_Set(v)
07            then A ← A ∪ {(u, v)}
08                 Union(u, v)
09  return A
```

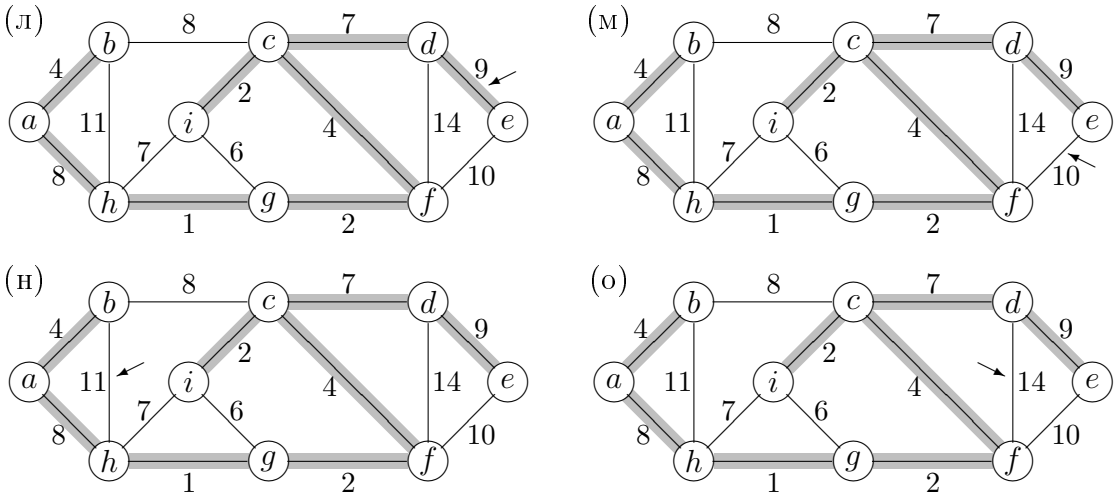
Процедура Kruskal работает следующим образом. В строках 01–03 происходит инициализация множества A пустым множеством и создаётся $|V|$ деревьев, каждое из которых содержит по одной вершине. В строке 04 все рёбра графа G сортируются согласно их весу в неубывающем порядке. Цикл **for** в строках 05–08 проверяет для

каждого ребра (u, v), принадлежат ли его концы разным деревьям. Если это так, в строке 07 ребро (u, v) добавляется в множество A, и вершины двух деревьев объединяются в строке 08. В противном случае данное ребро не может быть добавлено к лесу без того, чтобы создать при этом цикл, поэтому такое ребро отбрасывается. Множество рёбер A, возвращаемое в строке 09, является искомым.

Результат работы процедуры KRUSKAL может зависеть от порядка просмотра рёбер с одинаковым весом в строке 05 — этот порядок задаётся сортировкой рёбер в строке 04.

Пример. Ниже проиллюстрирован пример работы процедуры KRUSKAL. Рёбра растущего леса $G_A = \langle V, A \rangle$ выделены цветом. Стрелкой отмечено ребро, которое исследуется в текущей итерации цикла **for**.





Свойства алгоритма

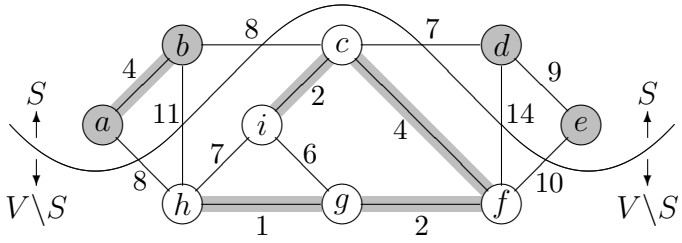
Корректность работы алгоритма Крускала следует из *свойства разреза*. Данное свойство позволяет убедиться в том, что на каждой итерации в лес G_A действительно добавляется безопасное ребро.

Определение. *Разрезом* неориентированного графа $G = \langle V, E\rangle$ называется разбиение $\langle S, V \setminus S\rangle$ множества V на два непересекающихся подмножества S и $V \setminus S$.

Говорят, что ребро $(u, v) \in E$ *пересекает* разрез $\langle S, V \setminus S\rangle$, если одна из вершин u или v принадлежит множеству S, а другая — множеству $V \setminus S$.

Говорят, что разрез графа G *согласован* с множеством рёбер $A \subseteq E$, если ни одно ребро из A не пересекает данный разрез.

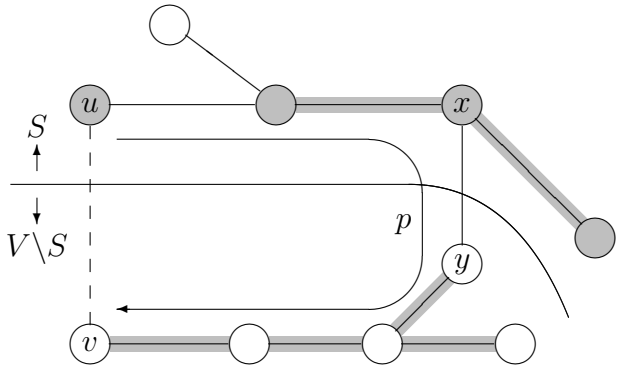
Пример. На рисунке проиллюстрирован пример разреза взвешенного связного неориентированного графа. Вершины множества S окрашены в серый цвет, вершины $V \setminus S$ — в белый. Выделенные цветом рёбра образуют множество A. Приведённый на рисунке разрез согласован с множеством A.



Лемма 19 (свойство разреза)**.** *Пусть $G = \langle V, E\rangle$ — взвешенный связный неориентированный граф с весовой функцией $w : E \to \mathbb{R}$. Пусть A — подмножество E, которое входит в некоторое минимальное остовное дерево графа G; $\langle S, V \setminus S\rangle$ — разрез графа G, согласованный с A; а $(u, v) \in E$ — ребро, пересекающее разрез $\langle S, V \setminus S\rangle$ и имеющее минимальный вес среди всех рёбер, пересекающих данный разрез. Тогда ребро (u, v) является безопасным для A.*

Доказательство. По условию существует $T \subseteq E$ такое, что $A \subseteq T$ и $\langle V, T\rangle$ — минимальное остовное дерево. Если $(u, v) \in T$, то лемма доказана. Поэтому предположим, что $(u, v) \notin T$, и построим другое множество $T' \subseteq E$, которое образует минимальное остовное дерево и включает $A \cup \{(u, v)\}$. Множество T' получается из T путём вырезания и вставки рёбер.

Поскольку $\langle V, T\rangle$ — остовное дерево для G, в нём найдётся путь p из вершины u к вершине v. Ребро (u, v) образует цикл с рёбрами на пути p (см. рисунок).



Поскольку u и v находятся на разных сторонах разреза $\langle S, V \setminus S\rangle$, на пути p найдётся ребро $(x, y) \in T$, которое пересекает данный разрез. Ребро (x, y) не принадлежит A, т.к. разрез согласован с A. По теореме 4 удаление ребра (x, y) из T разбивает дерево $\langle V, T\rangle$ на два связных компонента, а добавления ребра (u, v) восстанавливает связность. Следовательно, множество рёбер

$$T' = (T \setminus \{(x, y)\}) \cup \{(u, v)\}$$

задаёт на вершинах из V связный граф $\langle V, T'\rangle$. Граф $\langle V, T'\rangle$ также является ациклическим. Для этого достаточно заметить, что если бы граф $\langle V, T'\rangle$ содержал цикл c, включающий ребро (u, v), то, заменив (u, v) в цикле c на путь p, мы получили бы цикл в дереве $\langle V, T\rangle$, что невозможно. Таким образом, $\langle V, T'\rangle$ — остовное дерево для графа G.

Докажем, что $\langle V, T'\rangle$ — минимальное остовное дерево. Поскольку ребро (u, v) имеет минимальный вес среди всех рёбер, пересекающих разрез $\langle S, V \setminus S\rangle$, и ребро (x, y) тоже пересекает данный разрез, заключаем, что $w(u, v) \leqslant w(x, y)$. Тогда

$$w(T') = w(T) - w(x, y) + w(u, v) \leqslant w(T).$$

С другой стороны, в силу минимальности общего веса T, имеем $w(T) \leqslant w(T')$. Таким образом, $w(T') = w(T)$ и, значит, множество T' тоже задаёт минимальное остовное дерево для графа G.

Остаётся показать, что (u, v) — безопасное ребро для A. Действительно, из условий $A \subseteq T$ и $(x, y) \notin A$ следует, что $A \subseteq T'$. Тогда $A \cup \{(u, v)\} \subseteq T'$ и при этом $\langle V, T'\rangle$ — минимальное остовное дерево, т.е. ребро (u, v) безопасно для A. □

Теорема 20 (о корректности алгоритма Крускала)**.** *Пусть $G = \langle V, E\rangle$ — взвешенный связный неориентированный граф с весовой функцией $w : E \to \mathbb{R}$ и пусть A — множество рёбер, полученное в результате применения процедуры* Kruskal *к графу G и функции w. Тогда граф $\langle V, A\rangle$ является минимальным остовным деревом для G.*

Доказательство. Докажем, что в любой момент времени выполнения процедуры Kruskal текущее множество рёбер A содержится в некотором минимальном остовном дереве, при этом граф $G_A = \langle V, A\rangle$ является лесом. После инициализации в строке 01 множество A пусто. Ясно, что пустое множество рёбер содержится в минимальном остовном дереве, существующем в силу предложения 18. Кроме этого, граф с пустым множеством рёбер, очевидно, является лесом, состоящим из одноэлементных деревьев.

Пусть перед очередной итерацией цикла **for** (строки 05–08) множество A включается в некоторое $T \subseteq E$, образующее минимальное остовное дерево, и пусть на данной итерации в A добавляется ребро (u, v). По построению ребро (u, v) соединяет связный компонент S в лесу G_A с некоторым другим связным компонентом. Тогда разрез $\langle S, V \setminus S\rangle$ согласован с A, а ребро (u, v) пересекает этот разрез.

Покажем, что ребро (u, v) имеет минимальный вес среди всех рёбер, пересекающих разрез $\langle S, V \setminus S\rangle$. Любое ребро $(x, y) \in E$ с весом $w(x, y) < w(u, v)$ было рассмотрено на одной из предыдущих итераций. Следовательно, либо ребро (x, y) уже попало в A и тогда не пересекает разрез $\langle S, V \setminus S\rangle$, либо концы ребра (x, y) в момент его просмотра лежали в одном связном компоненте. Заметим, что в процессе исполнения процедуры KRUSKAL связные компоненты могут только объединяться. Отсюда следует, что если в момент просмотра ребра (x, y) его концы попали в один связный компонент, то они будут находиться в одном связном компоненте и в момент просмотра ребра (u, v), и значит, ребро (x, y) не может пересекать разрез $\langle S, V \setminus S\rangle$. Таким образом, если какое-то ребро пересекает разрез $\langle S, V \setminus S\rangle$, то вес данного ребра не меньше чем $w(u, v)$. Следовательно, (u, v) имеет минимальный вес среди всех рёбер, пересекающих указанный разрез. Отсюда, в силу леммы 19, заключаем, что ребро (u, v) является безопасным для текущего множества A. Следовательно, множество $A \cup \{(u, v)\}$ тоже содержится в некотором минимальном остовном дереве, а граф $\langle V, A \cup \{(u, v)\}\rangle$ является лесом, поскольку получается путём объединения двух деревьев из леса G_A.

Пусть теперь A — итоговое множество рёбер, возвращаемое агоритмом в строке 09. В силу доказанного выше, существует $T \subseteq E$ такое, что $A \subseteq T$ и граф $\langle V, T\rangle$ является минимальным остовным деревом. Остаётся доказать включение $T \subseteq A$. Допустим, напротив, существует ребро $(u, v) \in T$ такое, что $(u, v) \notin A$. Следовательно, в момент просмотра ребра (u, v) вершины u и v лежат в одном связном компоненте растущего леса. Тогда и в итоговом лесу G_A вершины u и v тоже лежат в одном связном компоненте. Поэтому существует путь p по рёбрам из A, начинающийся в вершине u и заканчивающийся в вершине v, причём p не совпадает с ребром (u, v), поскольку $(u, v) \notin A$. Из включения $A \subseteq T$ следует, что путь p полностью содержится в дереве $\langle V, T\rangle$. Добавив к пути p ребро $(u, v) \in T$, мы получим в дереве $\langle V, T\rangle$ цикл, что невозможно. Таким образом, $A = T$ и $\langle V, A\rangle$ является минимальным остовным деревом. □

Время работы алгоритма

Инициализация множества A в строке 01 занимает время $O(1)$. Дальнейшее время работы алгоритма Крускала зависит от выбранного способа сортировки весов рёбер и от реализации структуры разбиения множества вершин V на непересекающиеся подмножества.

Будем использовать быструю сортировку весов рёбер в неубывающем порядке с помощью *сортировки слиянием*, т.е. применяя процедуру MERGE_SORT к массиву $w_1, \ldots, w_n$, состоящему из весов рёбер графа G, где $n = |E|$. Тогда общее время сортировки составляет $O(|E| \cdot \log_2 |E|)$.

Каждое множество из разбиения V на непересекающиеся подмножества мы будем хранить в виде ориентированного дерева. Вершины этого дерева — элементы множества. Для каждой вершины v, кроме корня, определён предшественник $\pi[v]$ в дереве. Для корня r считаем, что $\pi[r] = r$. Таким образом, ориентированные рёбра имеют вид $(v, \pi[v])$, включая петлю на корне r. Помимо указателя на предшественника для

каждой вершины v определён *ранг*, который обозначается через $rank[v]$ и совпадает с высотой поддерева, растущего из вершины v.

Не каждое такое ориентированное дерево может возникнуть в ходе исполнения процедуры Kruskal. Нас интересуют только те ориентированные деревья, которые могу возникнуть в результате применения операций Make_Set и Union.

Процедура, создающая дерево из одной вершины, занимает время $O(1)$ и имеет следующий псевдокод:

```
Make_Set(v)
01  π[v] ← v
02  rank[v] ← 0
```

Корень дерева можно рассматривать как *представитель* соответствующего множества. Тогда процедура, определяющая, в каком из подмножеств разбиения содержится вершина v, находит корень соответствующего дерева за время, пропорциональное высоте дерева, и имеет следующий псевдокод:

```
Find_Set(v)
01  while v ≠ π[v]
02        do v ← π[v]
03  return v
```

Чтобы объединить два множества, представленных в виде деревьев, достаточно найти их корни и один из них сделать предком другого. Чтобы высота деревьев росла медленно, будем подвешивать более низкое дерево к более высокому. В таком случае высота увеличивается, только если объединяемые деревья имеют одинаковую высоту. Такой подход называется *объединением по рангу* и реализуется следующим псевдокодом:

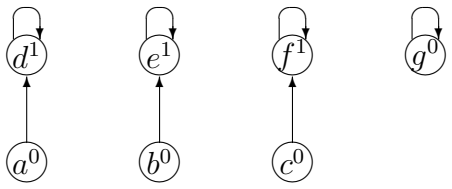
```
Union(u, v)
01  r_u ← Find_Set(u)
02  r_v ← Find_Set(v)
03  if rank[r_u] > rank[r_v]
04    then π[r_v] ← r_u
05    else π[r_u] ← r_v
06         if rank[r_u] = rank[r_v]
07           then rank[r_v] ← rank[r_v] + 1
```

Время работы процедуры Union(u, v) пропорционально сумме высот объединяемых деревьев.

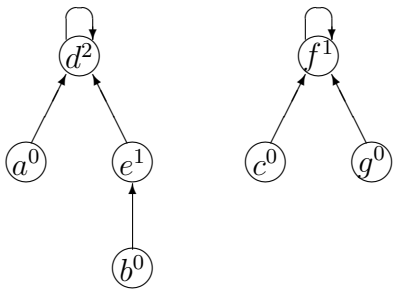
Пример. На рисунке приведён пример последовательного использования процедур Make_Set и Union. В верхних индексах вершин указаны их ранги.

После применения Make_Set(a), Make_Set(b), …, Make_Set(g):

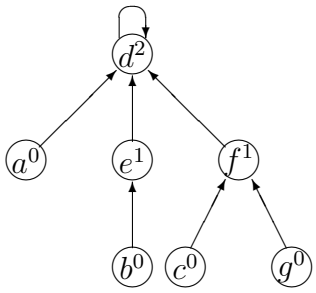
После применения Union(a, d), Union(b, e) и Union(c, f):



После применения UNION(e, a) и UNION(c, g):



После применения UNION(b, g):



Итак, все рассматриваемые множества из разбиения V на подмножества получаются только путём применения процедур MAKE_SET и UNION. Отсюда следует, что дерево высоты k получается только при объединении двух деревьев высоты $k-1$. Тогда, используя индукцию по k, можно показать, что вершина ранга k имеет не менее 2^k потомков, включая её саму.

Докажем, что имеется не более $|V|/2^k$ элементов ранга k. Допустим, напротив, количество вершин ранга k равно m и при этом $m > |V|/2^k$. Каждая из этих m вершин имеет не менее 2^k потомков. Следовательно, множество V содержит не менее $m \cdot 2^k$ элементов. В силу нашего предположения, $m \cdot 2^k > (|V|/2^k) \cdot 2^k = |V|$, что, очевидно, невозможно. Теперь покажем, что ранг любого элемента не превосходит $\log_2 |V|$. Если это не так, то найдётся хотя бы одна вершина ранга $r > \log_2 |V|$. Общее количество вершин ранга r не превосходит $|V|/2^r$. Тогда из справедливости соотношений $|V|/2^r < |V|/2^{\log_2 |V|} = 1$ заключаем, что в множестве V вершины ранга r отсутствуют.

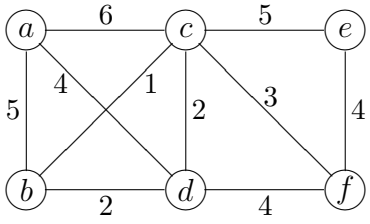
Итак, ранг любого элемента не превосходит $\log_2 |V|$. Значит, все ориентированные деревья имеют высоту не более чем $\log_2 |V|$, откуда следует, что время работы каждой из процедур FIND_SET и UNION составляет $O(\log_2 |V|)$.

Всего в процедуре KRUSKAL используется $|V|$ операций MAKE_SET, $2{\cdot}|E|$ операций FIND_SET и $|V|-1$ операций UNION. Следовательно, общее время работы над непересекающимися множествами составляет $O(|V| + (|E| + |V|){\cdot}\log_2 |V|) = = O((|E| + |V|){\cdot}\log_2 |V|)$. Поскольку граф G связный, имеет место соотношение $|E| \geqslant |V| - 1$, так что операции над непересекающимися множествами требуют время $O(|E|{\cdot}\log_2 |V|)$. Добавив к этому времени время, затрачиваемое на сортировку рёбер, получаем $O(|E|{\cdot}\log_2 |E| + |E|{\cdot}\log_2 |V|)$. Из неравенства $|E| \leqslant |V|^2$ следует, что $\log_2 |E| \leqslant \log_2 |V|^2 = 2\log_2 |V|$. Поэтому общее время работы алгоритма Крускала составляет $O(|E|{\cdot}\log_2 |V|)$.

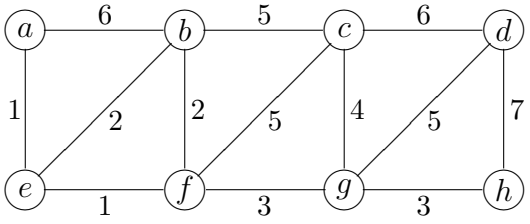
Упражнения

1. Применить алгоритм Крускала к следующим неориентированным графам. Для каждой итерации работы алгоритма привести текущее состояние разбиения множества вершин на непересекающиеся множества, представляя их в виде ориентированных деревьев.

(а)



(б)



2. Доказать, что для любого множества рёбер T, определяющего минимальное остовное дерево $\langle V, T\rangle$ для графа $G = \langle V, E\rangle$, можно указать способ сортировки рёбер из E, для которого алгоритм Крускала даст множество рёбер T.

3. Доказать, что если веса всех рёбер в связном неориентированном графе различны, то граф имеет единственное минимальное остовное дерево.

4. Пусть (u, v) — ребро минимального веса во взвешенном связном неориентированном графе G. Доказать, что (u, v) принадлежит некоторому минимальному остовному дереву графа G.

5. Доказать, что если ребро содержится в некотором минимальном остовном дереве графа G, то оно является ребром с минимальным весом, пересекающим некоторый разрез графа G.

6. Привести пример взвешенного связного неориентированного графа $G = \langle V, E\rangle$ с весовой функцией $w : E \to \mathbb{R}$, подмножества $A \subseteq E$, входящего в некоторое минимальное остовное дерево графа G, разреза $\langle S, V \setminus S\rangle$ графа G, согласованного с A, и ребра $(u, v) \in E$, пересекающего разрез $\langle S, V \setminus S\rangle$, таких, что ребро (u, v) является безопасным для A, но вес ребра (u, v) не является минимальным среди всех рёбер, пересекающих данный разрез.

7. Доказать, что в представлении множеств из разбиения V в виде ориентированных деревьев каждая вершина ранга k имеет не менее 2^k потомков, включая её саму.

Глава III

Конечные автоматы и формальные грамматики

§ 8. Алфавиты и формальные языки

В большинстве случаев данные, с которыми работают алгоритмы, представляются в виде слов некоторого языка. Алгоритмы подобного вида можно назвать *словарными*. Например, алгоритмы, реализуемые с помощью конечных автоматов или формальных грамматик, безусловно, являются словарными. Более того, описание самих алгоритмов, как правило, осуществляется с помощью специального текста, который обычно называют *программой*.

Понятия слова, языка и текста имеют очень широкий спектр значений, выходящий за рамки математической науки. Мы будем работать только с *формальными языками*, т. е. с языками, которые поддаются формальному описанию в рамках теории множеств и которые можно изучать, используя строгие математические методы.

Определение. *Алфавит* — это любое непустое конечное множество $A=\{a_0, \ldots, a_n\}$. Элементы алфавита принято называть *буквами*, или *символами*.

Определение. *Словом длины* n в алфавите A мы будем называть любой кортеж $\langle a_1, a_2, \ldots, a_n\rangle$ длины n, где $a_1, \ldots, a_n$ — буквы алфавита A. Слова обычно записывают в виде $a_1a_2\ldots a_n$. *Пустое слово* $\varnothing$ не содержит ни одной буквы, имеет длину 0 и обозначается через Λ.

Замечание. В дальнейшем запись слова в виде $a_1a_2\ldots a_n$ подразумевает, в частности, что при $n = 0$ это слово пусто. Это же соглашение распространяется на конечные кортежи и конечные множества, т. е. $\langle a_1, \ldots, a_n\rangle = \{a_1, \ldots, a_n\} = \varnothing$ при $n = 0$.

Определение. Множество всех слов в алфавите A, включая пустое слово, обозначается через A^*, а множество всех непустых слов в алфавите A — через A^+, т. е. $A^+ = A^* \setminus \{\Lambda\}$.

Определение. Длина слова $w \in A^*$ обозначается через $|w|$.

Определение. Слово u называется *подсловом* слова v, если существуют слова w_1 и w_2 такие, что $v = w_1uw_2$, при этом *вхождением* подслова u в слово v назовём упорядоченную тройку $\langle w_1, u, w_2\rangle$. Отметим, что одно и то же слово u может иметь несколько различных вхождений в слово v.

Определение. Слово u называется *префиксом* (*суффиксом*) слова v, если $v = uw$ ($v = wu$) для некоторого слова w.

Определение. Введём следующие операции над словами в алфавите A:
(а) *Конкатенацией слов* u и v называется слово uv, полученное приписыванием к слову u слова v справа.
(б) *Степень слова* u определяется индукцией по $n \in \omega$ следующим образом: $u^0 = \Lambda$, $u^{n+1} = u^n u$.
(в) *Обращением слова* $u = a_1 a_2 \dots a_n$, где $a_1, a_2, \dots, a_n$ — буквы алфавита A, называется слово $u^R = a_n \dots a_2 a_1$.

Пример. Конкатенацией слов *aaba* и *abb* является слово *aabaabb*. Если теперь взять конкатенацию тех же слов, но в другом порядке, то получится слово *abbaaba*. Слово *aab* является подсловом слова *aabaabb*, причем оно имеет два вхождения: первое вхождение начинается со 1-го символа слова, а второе — с 4-го. Обращением слова *abb* будет слово *bba*. Степенями слова *aaba* являются слова Λ, *aaba*, *aabaaaba*, *aabaaabaaaaba* и т. д. При этом каждое слово из данной последовательности будет префиксом всех последующих слов.

Определение. Любое подмножество $L \subseteq A^*$ называется *формальным языком над алфавитом* A.

Определение. Введём следующие операции над языками в алфавите A:
(а) *Объединение* $L_1 \cup L_2$ и *пересечение* $L_1 \cap L_2$ языков L_1 и L_2 определяются как обычные теоретико-множественные объединение и пересечение.
(б) *Дополнением* языка L называется язык $\overline{L} = A^* \setminus L$.
(в) *Конкатенацией языков* L_1 и L_2 называется язык $L_1 L_2 = \{uv \mid u \in L_1, v \in L_2\}$.
(г) *Степень языка* L определяется индукцией по $n \in \omega$ следующим образом: $L^0 = \{\Lambda\}$, $L^{n+1} = L^n L$.
(д) *Звёздочкой Клини* от языка L называется язык $L^* = \bigcup\limits_{n \in \omega} L^n$. Другими словами,

$$L^* = \{u \mid u = u_1 u_2 \dots u_n \text{ для некоторых } n \in \omega \text{ и } u_1, u_2, \dots, u_n \in L\}.$$

В частности, всегда имеет место $\Lambda \in L^*$ (при $n = 0$).

Замечание. Заметим, что определение звёздочки Клини согласуется с введёным выше обозначением A^*. Звёздочка Клини от алфавита — это и есть множество всех слов в данном алфавите.

Пример. С помощью введённых операций можно определять формально те языки, которые изначально имеют неформальное описание.

Например, язык всех слов в алфавите $A = \{a, b\}$, имеющих хотя бы одно вхождение буквы a, совпадает с языком $\{b\}^*\{a\}\{a, b\}^*$. Действительно, в любом таком слове можно выделить первое вхождение буквы a, т. е. представить в виде $b^n a w$, где $n \in \omega$, w — некоторое слово в алфавите A. С другой стороны, любое слово вида $b^n a w$ лежит в языке $\{b\}^*\{a\}\{a, b\}^*$.

Другой пример — это язык всех слов чётной длины в том же алфавите, который можно представить как $\{aa, ab, ba, bb\}^*$. Действительно, слово w имеет чётную длину тогда и только тогда, когда его можно представить в виде $w = w_1 \dots w_n$ для некоторого $n \in \omega$ и некоторых слов w_i, каждое из которых имеет длину 2, т. е. каждое $w_i \in \{aa, ab, ba, bb\}$.

Отсюда следует, что язык всех слов нечётной длины в алфавите $A = \{a, b\}$ можно получить как дополнение предыдущего языка, т. е. $\{a, b\}^* \setminus \{aa, ab, ba, bb\}^*$, а язык

всех слов чётной длины, содержащих букву a, — с помощью операции пересечения $(\{b\}^*\{a\}\{a,b\}^*) \cap \{aa, ab, ba, bb\}^*$.

Язык всех слов чётной длины, содержащих букву a, можно получить и другим способом, а именно как язык $\{bb\}^*\{aa, ab, ba\}\{aa, ab, ba, bb\}^*$.

Упражнения

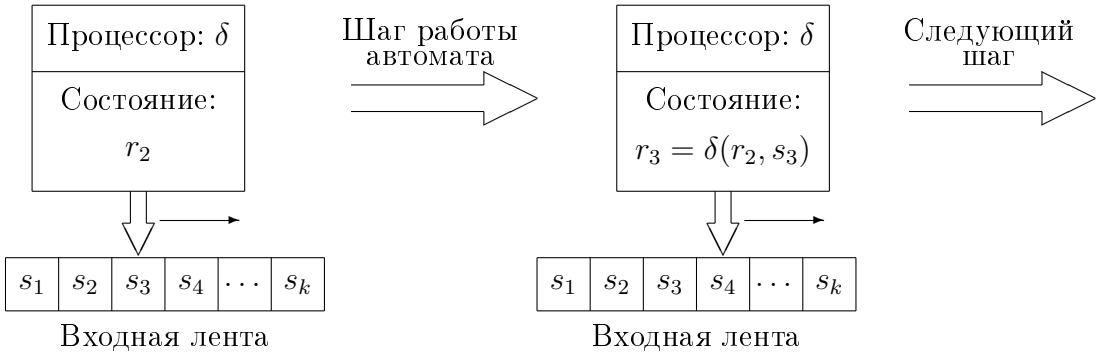
1. С помощью введённых выше операций над языками выразить следующие формальные языки в алвафите $A = \{a, b\}$:
   (а) язык всех слов, содержащих не более трёх символов a;
   (б) язык всех слов, количество вхождений символа a в которые кратно трём;
   (в) язык всех слов, имеющих ровно одно вхождение подслова aaa.

2. Доказать следующие равенства языков в алфавите $A = \{a, b\}$:
   (а) $\{a, b\}^* = \{a\}^*(\{b\}\{a\}^*)^*$;
   (б) $\{\Lambda\} \cup \{a\}\{ba, baa\}^*\{b, ba\} = \{ab, aba\}^*$;
   (в) $(\{a\} \cup \{b\}\{a\}^*\{a\})(\{b\}\{a\}^*\{a\})^*\{b\} = \{ab\} \cup (\{b\} \cup \{ab\})(\{a\} \cup \{ab\})^*\{ab\}$.

§ 9. Детерминированные конечные автоматы

Конечные автоматы относятся к классу словарных алгоритмов. С помощью конечных автоматов можно решать задачи из достаточно широкого семейства алгоритмических проблем. Например, технология проектирования микросхем основана на результатах теории автоматов. Другой пример — это компиляторы, перерабатывающие текст программы, написанной на языке программирования высокого уровня, в программу на машинном языке. Работа любого такого компилятора состоит из двух стадий. Первая стадия — лексический анализ текста, который реализуется с помощью определённого конечного автомата. Вторая стадия — синтаксический анализ, который осуществляется с помощью автомата с магазинной памятью. Однако, как будет видно позднее, не любая алгоритмическая задача поддаётся решению с помощью конечного автомата.

Дадим сначала неформальное определение конечных автоматов и описание их работы.



Входными данными любого конечного автомата являются слова в некотором заранее фиксированном алфавите, а выходными — две логические константы `true` и `false`. В каждый момент времени автомат находится в определённом *внутреннем*

состоянии. Число состояний автомата конечно. В начальный момент времени автомат находится в *начальном состоянии*. Кроме этого, некоторые состояния автомата считаются *выделенными*. Входное слово записывается на *входной ленте*, разбитой на ячейки: в каждой ячейке содержится одна буква слова. Автомат связан с лентой посредством специальной *управляющей головки*, которая последовательно считывает символы из ячеек, двигаясь слева направо. Очередной считанный символ отправляется в *процессор*, определяющий по текущему состоянию и по данному считанному символу новое состояние, в которое переводится автомат. Функция, вычисляющая это новое состояние, называется *функцией перехода*. Функция перехода расположена внутри процессора и не изменяется в ходе вычислений. Считав все символы слова, автомат останавливается в определённом состоянии: если это состояние оказывается выделенным, то на выход подаётся константа `true` — автомат распознал слово, в противном случае подается константа `false` — автомат не распознал слово.

Отличительной чертой конечных автоматов является отсутствие памяти вне процессора, т. е. вся память автомата занята программой. Эта особенность позволяет пролить свет на причины неспособности конечных автоматов решить любую алгоритмически разрешимую задачу (для алгоритмов определённого вида необходима дополнительная память).

Теперь перейдём к формальным определениям в терминах теории множеств.

Определение. *Детерминированным конечным автоматом* (сокращённо д.к.а.) называется упорядоченная пятёрка $\mathfrak{A} = \langle Q, A, \delta, q_0, F\rangle$, состоящая из следующих объектов:

(а) $Q = \{q_0, \ldots, q_m\}$ — конечный алфавит *внутренних состояний* автомата;

(б) $A = \{a_0, \ldots, a_n\}$ — конечный *входной алфавит* автомата;

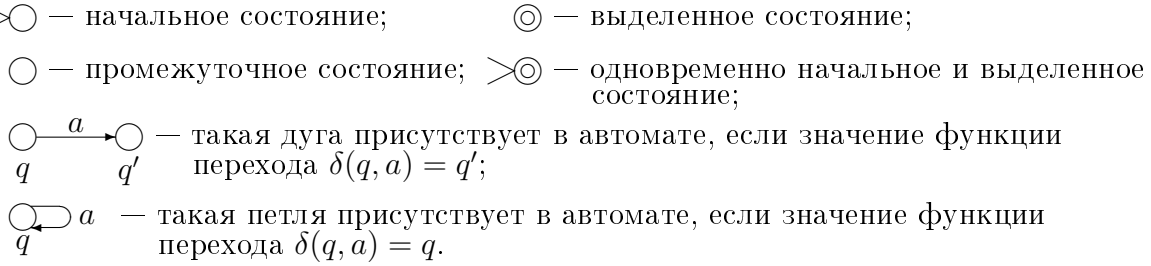
(в) $\delta : Q \times A \to Q$ — *функция перехода*;

(г) $q_0 \in Q$ — *начальное состояние*;

(д) $F \subseteq Q$ — множество *выделенных (конечных) состояний*.

Графическое изображение детерминированных автоматов

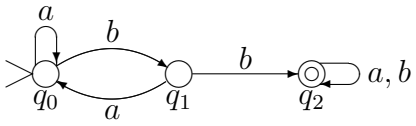
Автоматы удобно изображать в виде ориентированных графов с помеченными дугами, используя следующие геометрические фигуры:



Пример. Например, автомат $\mathfrak{A} = \langle Q, A, \delta, q_0, F\rangle$, где $A = \{a, b\}$, $Q = \{q_0, q_1, q_2\}$, $F = \{q_2\}$, а функция перехода задаётся соотношениями

$$\begin{array}{lll} \delta(q_0, a) = q_0, & \delta(q_1, a) = q_0, & \delta(q_2, a) = q_2, \\ \delta(q_0, b) = q_1, & \delta(q_1, b) = q_2, & \delta(q_2, b) = q_2, \end{array}$$

имеет следующее графическое изображение:



Определение. *Путём* в детерминированном конечном автомате $\mathfrak{A} = \langle Q, A, \delta, q_0, F\rangle$ назовём любую конечную последовательность $\langle r_0, s_1, r_1, \ldots, s_k, r_k\rangle$, где $r_0, \ldots, r_k \in Q$, $s_1, \ldots, s_k \in A$ и $\delta(r_i, s_{i+1}) = r_{i+1}$ для всех $i < k$. Если $\langle r_0, s_1, r_1, \ldots, s_k, r_k\rangle$ — путь в автомате, то будем обозначать его через

$$r_0 \xrightarrow{s_1} r_1 \xrightarrow{s_2} \ldots \xrightarrow{s_k} r_k$$

и говорить, что слово $w = s_1 s_2 \ldots s_k$ *читается вдоль дуг данного пути*. В частности, пустое слово Λ читается вдоль пути $\langle r_0\rangle$, состоящего из одного состояния и не содержащего ни одной дуги.

Определение. Пусть $\mathfrak{A} = \langle Q, A, \delta, q_0, F\rangle$ — д.к.а. Расширим функцию δ до функции $\delta^* : Q \times A^* \to Q$ следующим образом индукцией по длине слова:

$$\delta^*(q, \Lambda) = q, \quad \delta^*(q, wa) = \delta(\delta^*(q, w), a),$$

где $q \in Q$, $w \in A^*$, $a \in A$.

Определение. Говорят, что д.к.а. $\mathfrak{A} = \langle Q, A, \delta, q_0, F\rangle$ *распознаёт* слово $w \in A^*$, если $\delta^*(q_0, w) \in F$.

Другими словами, слово $w = s_1 s_2 \ldots s_k$ распознаётся автоматом, если в нём существует путь

$$q_0 = r_0 \xrightarrow{s_1} r_1 \xrightarrow{s_2} \ldots \xrightarrow{s_k} r_k \in F$$

такой, что он начинается в начальном состоянии q_0, вдоль его дуг читается слово $s_1 s_2 \ldots s_k$ (в детерминированном автомате такой путь определяется однозначно по слову w) и заканчивается в некотором выделенном состоянии.

Определение. Через $L(\mathfrak{A}) = \{w \in A^* \mid \delta^*(q_0, w) \in F\}$ будем обозначать *язык всех слов, распознаваемых конечным автоматом* $\mathfrak{A}$.

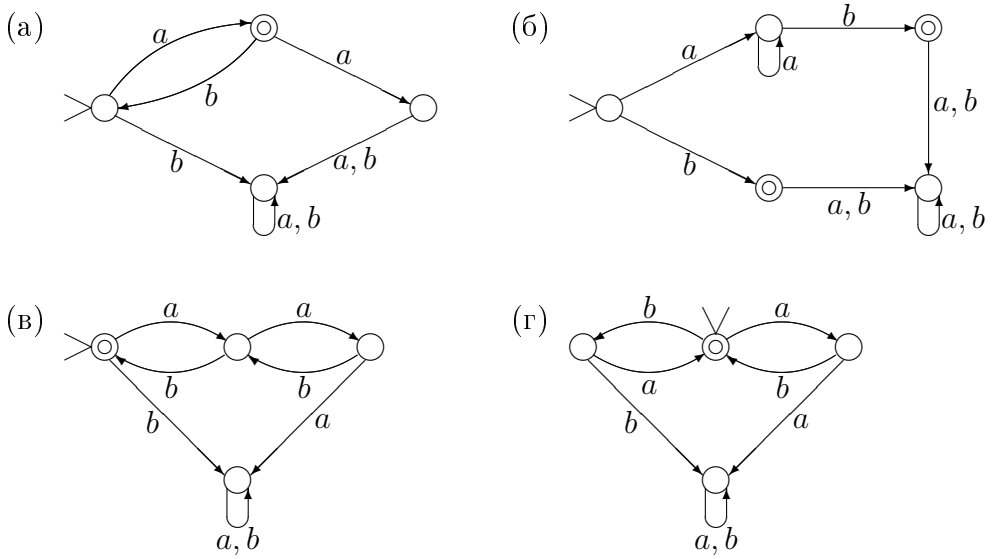
Определение. Язык $L \subseteq A^*$ называется *автоматным*, если существует детерминированный конечный автомат $\mathfrak{A}$ такой, что $L = L(\mathfrak{A})$.

Пример (продолжение)**.** Автомат из предыдущего примера распознаёт в точности все слова в алфавите $\{a, b\}$, которые содержат подслово bb, т. е. $L(\mathfrak{A}) = \{w \in \{a, b\}^* \mid w$ содержит подслово $bb\}$. Действительно, начав чтение произвольного слова w в начальном состоянии данного автомата, попасть в выделенное состояние можно, только пройдя по дуге, ведущей из q_0 в q_1 (помеченной b), а затем сразу же по дуге, ведущей из q_1 в q_2 (тоже помеченной b), — такое возможно лишь в случае, когда w содержит две буквы b подряд.

Упражнения

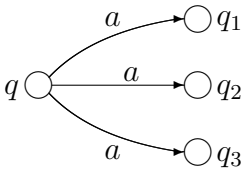
1. Построить детерминированный конечный автомат, распознающий следующий язык L над алфавитом $A = \{a, b\}$:
   (а) $L = \{a^{2n} \mid n \in \omega, n \geqslant 1\}$;
   (б) $L = \{w \mid$ в w разность числа символов a и числа символов b чётна$\}$;
   (в) $L = \{w \mid$ в слове w перед каждым символом a стоит символ $b\}$;
   (г) $L = \{w \mid$ слово w имеет подслово $abab\}$;
   (д) $L = \{w \mid$ слово w не имеет подслов вида aa и подслов вида $bb\}$;
   (е) $L = \{w \mid w$ имеет нечётное число символов a и чётное число символов $b\}$;
   (ж) $L = \{w \mid$ слово w имеет подслово ab и имеет подслово $ba\}$.

2. Описать язык, распознаваемый следующим детерминированным конечным автоматом:



3. Говорят, что д.к.а. $\mathfrak{A} = \langle Q, A, \delta, q_0, F\rangle$ обладает *свойством вахтёра*, если для любых $q \in Q$, $a \in A$ имеет место $\delta(q, a) \neq q_0$, т. е. в автомате нет дуг, входящих в начальное состояние. Доказать, что для любого д.к.а. $\mathfrak{A}$ существует д.к.а. $\mathfrak{A}'$ такой, что $L(\mathfrak{A}) = L(\mathfrak{A}')$ и $\mathfrak{A}'$ обладает свойством вахтёра.

§ 10. Недетерминированные конечные автоматы



Недетерминированные конечные автоматы отличаются от детерминированных тем, что из любого состояния q после считывания символа a возможны сразу несколько (или ни одного) переходов в состояния, принадлежащие *множеству возможных состояний* $\Delta(q, a)$. Это означает, что автомат может продолжить дальнейший процесс чтения символов, перейдя в любое из возможных состояний.

Можно считать, что в таких ситуациях работа автомата разветвляется на несколько параллельных ветвей вычислений, каждая из которых начинается с определённого состояния $q_i \in \Delta(q, a)$. Если $\Delta(q, a)$ пусто, то работа автомата вдоль данной ветви вычислений заканчивается в состоянии q, даже если на входной ленте остались

несчитанные символы. Таким образом, вдоль одних ветвей вычислений автомат может прочитать все входные символы и остановится в выделенном состоянии, вдоль других может прочитать все символы, но остановится в невыделенном состоянии, вдоль третьих ветвей работа автомата «обрывается на полуслове».

Перейдём к формальным определениям, связанным с недетерминированными автоматами.

Определение. *Недетерминированным конечным автоматом* (сокращённо н.к.а.) называется упорядоченная пятёрка $\mathfrak{A} = \langle Q, A, \Delta, q_0, F\rangle$, в которой Q, A, q_0, F определяются и называются так же, как в детерминированном случае, а *функция переходов* Δ является функцией вида $\Delta : Q \times A \to P(Q)$, где $P(Q)$ — множество всех подмножеств Q (см. аксиому степени из § 1).

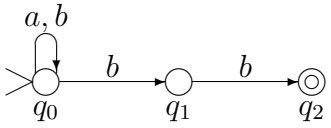
Графическое изображение недетерминированных автоматов

При графическом изображении недетерминированных автоматов используются те же обозначения, что и в детерминированном случае. При этом из одного состояния автомата могут выходить сразу несколько (или нисколько) стрелок, помеченных одной и той же буквой алфавита. Дуга, выходящая из состояния q, входящая в состояние q', помеченная символом a, присутствует в схеме автомата тогда и только тогда, когда $q' \in \Delta(q, a)$.

Пример. Например, автомат $\mathfrak{A} = \langle Q, A, \Delta, q_0, F\rangle$, где $A = \{a, b\}$, $Q = \{q_0, q_1, q_2\}$, $F = \{q_2\}$, а функция переходов задаётся соотношениями

$$\begin{array}{lll} \Delta(q_0, a) = \{q_0\}, & \Delta(q_1, a) = \varnothing, & \Delta(q_2, a) = \varnothing, \\ \Delta(q_0, b) = \{q_0, q_1\}, & \Delta(q_1, b) = \{q_2\}, & \Delta(q_2, b) = \varnothing, \end{array}$$

имеет следующее графическое изображение:



Определение. *Путь* в недетерминированном конечном автомате $\mathfrak{A}=\langle Q, A, \Delta, q_0, F\rangle$ определяется и обозначается так же, как в детерминированном случае, нужно лишь условие $\delta(r_i, s_{i+1}) = r_{i+1}$ заменить на условие $r_{i+1} \in \Delta(r_i, s_{i+1})$.

Определение. Говорят, что н.к.а. $\mathfrak{A}=\langle Q, A, \Delta, q_0, F\rangle$ *распознаёт* слово $s_1 s_2 \ldots s_k \in A^*$, если существует последовательность состояний $q_0 = r_0, r_1, \ldots, r_k$ такая, что

$$\begin{array}{l} r_1 \in \Delta(r_0, s_1), \\ r_2 \in \Delta(r_1, s_2), \\ \vdots \qquad\qquad \vdots \\ r_k \in \Delta(r_{k-1}, s_k), \end{array}$$

и при этом $r_k \in F$.

Другими словами, слово $w = s_1 s_2 \ldots s_k$ распознаётся автоматом, если в нём существует хотя бы один путь

$$q_0 = r_0 \xrightarrow{s_1} r_1 \xrightarrow{s_2} \ldots \xrightarrow{s_k} r_k \in F$$

такой, что он начинается в начальном состоянии q_0, вдоль его дуг читается слово $s_1 s_2 \ldots s_k$ (в недетерминированном автомате такой путь определяется неоднозначно по слову w) и заканчивается в некотором выделенном состоянии.

Определение. Как и раньше, через $L(\mathfrak{A})$ обозначается *язык всех слов, распознаваемых автоматом* $\mathfrak{A}$.

Пример (продолжение)**.** Автомат из предыдущего примера распознаёт в точности все слова в алфавите $\{a, b\}$, которые имеют суффикс bb, т. е. $L(\mathfrak{A}) = \{w \in \{a,b\}^* \mid \exists v \in \{a,b\}^*(w = vbb)\}$. Действительно, любой путь, заканчивающийся в выделенном состоянии данного автомата, обязан содержать участок $q_0 \xrightarrow{b} q_1 \xrightarrow{b} q_2$. Отсюда следует, что слова, распознаваемые $\mathfrak{A}$, должны содержать хотя бы одно вхождение подслова bb. Поскольку из выделенного состояния нет выходящих стрелок, то любой путь, содержащий описанный выше участок, обрывается как раз после прохождения данного участка. Тогда в словах, распознаваемых автоматом, после крайнего справа вхождения подслова bb нет букв.

Теорема 21 (о детерминизации)**.** *Для любого недетерминированного конечного автомата* $\mathfrak{A}$ *существует детерминированный конечный автомат* $\mathfrak{A}'$ *такой, что* $L(\mathfrak{A}) = L(\mathfrak{A}')$.

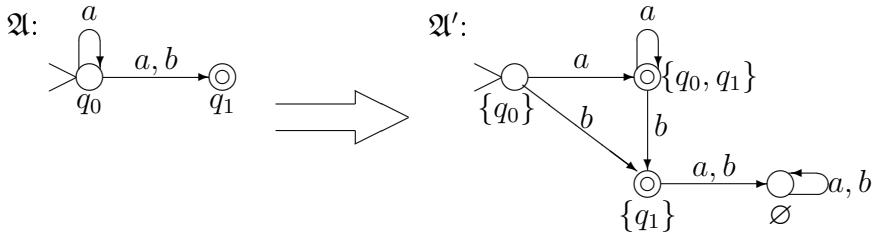
Доказательство. Пусть $\mathfrak{A} = \langle Q, A, \Delta, q_0, F\rangle$ — исходный н.к.а. Определим следующий д.к.а. $\mathfrak{A}' = \langle Q', A, \delta, q_0', F'\rangle$ (см. пример на рисунке):

(а) $Q' = P(Q)$;

(б) $\delta(q', a) = \bigcup\limits_{r \in q'} \Delta(r, a)$, где $q' \in Q'$ и $a \in A$;

(в) $q_0' = \{q_0\}$;

(г) $F' = \{q' \in Q' \mid q' \cap F \neq \varnothing\}$.



Докажем, что для любого слова $w \in A^*$ и любого состояния $p \in Q$ следующие условия эквивалентны:

(а) В н.к.а. $\mathfrak{A}$ существует путь, который начинается в q_0, заканчивается в p и вдоль дуг которого читается слово w.

(б) В д.к.а. $\mathfrak{A}'$ существует путь, который начинается в $\{q_0\}$, заканчивается в некотором $p' \ni p$ и вдоль дуг которого читается слово w.

Заметим, что если p — выделенное состояние $\mathfrak{A}$, то условие (а) эквивалентно условию $w \in L(\mathfrak{A})$, а условие (б) — условию $w \in L(\mathfrak{A}')$. Таким образом, $L(\mathfrak{A}) = L(\mathfrak{A}')$.

Докажем индукцией по длине слова w эквивалентность условий (а) и (б).

1^0. Пусть $|w| = 0$, т.е. $w = \Lambda$. Тогда условие (а) эквивалентно тому, что в $\mathfrak{A}$ существует путь, начинающийся в q_0, заканчивающийся в p и не содержащий ни одной дуги, что, в свою очередь, эквивалентно условию $q_0 = p$. Покажем, что условие $q_0 = p$ эквивалентно (б). Действительно, если $q_0 = p$, то, взяв $p' = \{q_0\}$, очевидно, получим условие (б). Если же в д.к.а. $\mathfrak{A}'$ существует путь, начинающийся в $\{q_0\}$, заканчивающийся в некотором $p' \ni p$ и вдоль дуг которого читается пустое слово, то с необходимостью $p' = \{q_0\}$. Следовательно, $p \in \{q_0\}$ и, значит, $q_0 = p$.

2^0. Допустим, эквивалентность (а) и (б) уже доказана для произвольных слов w длины k. Докажем эквивалентность (а) и (б) для произвольного слова w длины $k+1$. Пусть $w = s_1 \dots s_k s_{k+1}$, где $s_i \in A$.

Пусть для слова w выполняется условие (а). Следовательно, существует состояние $r \in Q$ такое, что в н.к.а. $\mathfrak{A}$ имеется путь

$$q_0 \xrightarrow{s_1} \dots \xrightarrow{s_k} r \xrightarrow{s_{k+1}} p.$$

В частности, слово $v = s_1 \dots s_k$ читается вдоль дуг пути в $\mathfrak{A}$, который начинается в q_0 и заканчивается в r, т. е. для слова v и состояния r выполняется условие (а). В силу индукционного предположения, для v и r выполняется условие (б), т. е. в д.к.а. $\mathfrak{A}'$ существует путь, который начинается в $\{q_0\}$, заканчивается в некотором $r' \ni r$ и вдоль дуг которого читается слово v:

$$\{q_0\} \xrightarrow{s_1} \dots \xrightarrow{s_k} r' \ni r.$$

Так как $p \in \Delta(r, s_{k+1})$ и $r \in r'$, то $p \in \bigcup\limits_{\overline{r} \in r'} \Delta(\overline{r}, s_{k+1}) = \delta(r', s_{k+1})$. Следовательно, в д.к.а. $\mathfrak{A}'$ существует дуга $r' \xrightarrow{s_{k+1}} p'$, где $p' = \delta(r', s_{k+1})$ и $p' \ni p$.

Таким образом, в $\mathfrak{A}'$ существует путь вида

$$\{q_0\} \xrightarrow{s_1} \dots \xrightarrow{s_k} r' \xrightarrow{s_{k+1}} p' \ni p.$$

Отсюда заключаем, что слово w удовлетворяет условию (б).

Пусть теперь для слова w выполняется условие (б). Следовательно, существует состояние $r' \in Q'$ такое, что в д.к.а. $\mathfrak{A}'$ имеется путь вида

$$\{q_0\} \xrightarrow{s_1} \dots \xrightarrow{s_k} r' \xrightarrow{s_{k+1}} p' \ni p.$$

Так как $p' = \delta(r', s_{k+1}) = \bigcup\limits_{r \in r'} \Delta(r, s_{k+1})$ и $p \in p'$, то существует состояние $r \in r'$ такое, что $p \in \Delta(r, s_{k+1})$.

Поскольку $r \in r'$ и в д.к.а. $\mathfrak{A}'$ существует путь

$$\{q_0\} \xrightarrow{s_1} \dots \xrightarrow{s_k} r',$$

то для слова $v = s_1 \dots s_k$ выполняется условие (б). Следовательно, в силу индукционного предположения, для v и состояния r выполняется условие (а), т. е. в н.к.а. $\mathfrak{A}$ существует путь

$$q_0 \xrightarrow{s_1} \dots \xrightarrow{s_k} r.$$

Так как $p \in \Delta(r, s_{k+1})$, то в $\mathfrak{A}$ имеется дуга

$$r \xrightarrow{s_{k+1}} p.$$

Окончательно получаем существование в $\mathfrak{A}$ следующего пути

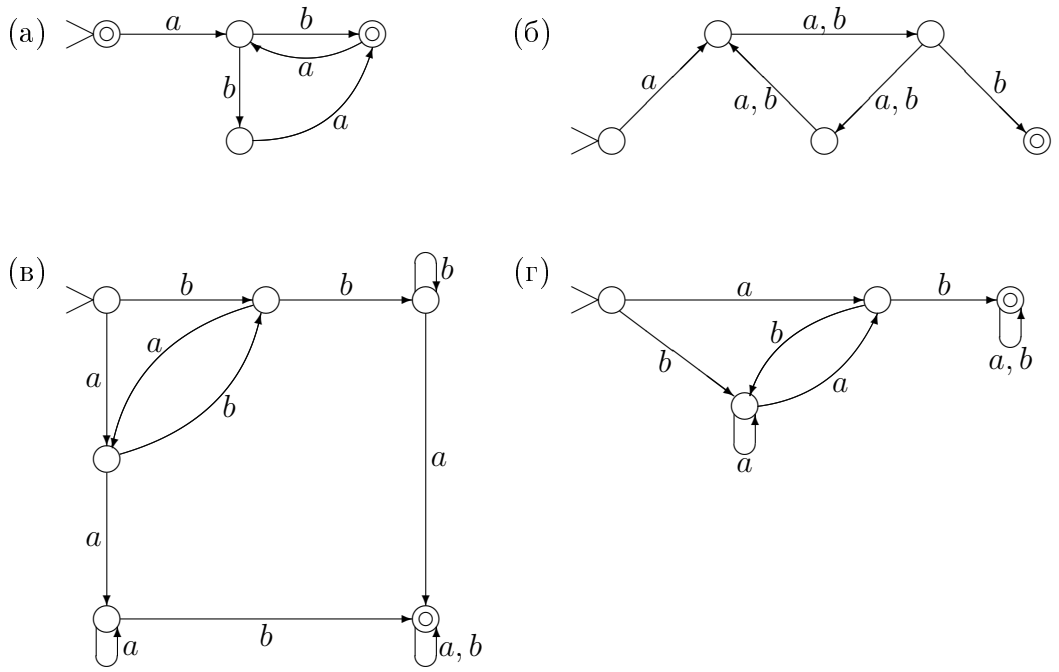
$$q_0 \xrightarrow{s_1} \dots \xrightarrow{s_k} r \xrightarrow{s_{k+1}} p,$$

вдоль дуг которого читается слово $s_1 \dots s_k s_{k+1} = w$, т. е. справедливо условие (а).

Таким образом, построенный д.к.а. $\mathfrak{A}'$ распознаёт язык $L(\mathfrak{A})$. □

Упражнения

1. Описать язык, распознаваемый следующим недетерминированным конечным автоматом:



2. Используя конструкцию из теоремы о детерминизации, построить детерминированные конечные автоматы, эквивалентные автоматам из пунктов (а) и (г) предыдущего упражнения.

§ 11. Недетерминированные конечные автоматы с пустыми переходами

В некоторых источниках (см., например, [5, 15]) используется более широкий класс недетерминированных конечных автоматов, который отличается от введённого выше тем, что функция переходов Δ имеет вид $\Delta : Q \times (A \cup \{\Lambda\}) \to P(Q)$. Автоматы данного типа называют *автоматами с пустыми переходами*, поскольку в них допускаются дуги, помеченные пустым словом Λ. Наличие дуги, помеченной Λ, выходящей из состояние q и входящей в состояние q', означает, что автомат, находясь в состоянии q, может перейти в состояние q', не считывая символа с ленты. Пользуясь терминами из программирования, можно сказать, что в такой ситуации автомат осуществляет *безусловный переход* из q в q'.

Такой вид автоматов удобно использовать при доказательстве некоторых свойств. Однако добавление Λ-переходов в определение недетерминированного автомата не изменяет класс автоматных языков, т. е. язык распознаётся некоторым недетерминированным конечным автоматом (согласно нашему определению) тогда и только тогда, когда он распознаётся некоторым недетерминированным конечным автоматом с Λ-переходами.

Определение. *Недетерминированным конечным автоматом с пустыми переходами* (сокращённо Λ-н.к.а.) называется упорядоченная пятёрка $\mathfrak{A} = \langle Q, A, \Delta, q_0, F\rangle$, в которой Q, A, q_0, F определяются и называются так же, как раньше, а *функция переходов* Δ является функцией вида $\Delta : Q \times (A \cup \{\Lambda\}) \to P(Q)$.

Определение. *Путь* в Λ-н.к.а. $\mathfrak{A} = \langle Q, A, \Delta, q_0, F\rangle$ определяется и обозначается так же, как в случае недетерминированного конечного автомата. Заметим лишь, что в условии $r_{i+1} \in \Delta(r_i, s_{i+1})$ допускается $s_{i+1} = \Lambda$.

Определение. Говорят, что Λ-н.к.а. $\mathfrak{A} = \langle Q, A, \Delta, q_0, F\rangle$ *распознаёт* слово $s_1 s_2 \dots s_k$, где s_i — буквы алфавита A, если в $\mathfrak{A}$ существует хотя бы один путь

$$r_0 \xrightarrow{t_1} r_1 \xrightarrow{t_2} \dots \xrightarrow{t_m} r_m$$

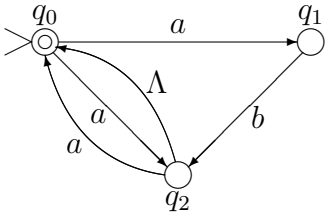
такой, что он начинается в начальном состоянии $r_0 = q_0$, заканчивается в некотором выделенном состоянии $r_m \in F$ и вдоль его дуг читается слово $t_1 t_2 \dots t_m = s_1 s_2 \dots s_k$, причём $m \geqslant k$, т. е. некоторые из символов $t_1, \dots, t_m$ могут быть пустыми.

Определение. Для произвольного Λ-н.к.а. $\mathfrak{A}$ через $L(\mathfrak{A})$ обозначим *язык всех слов, распознаваемых* $\mathfrak{A}$.

Пример. Автомат $\mathfrak{A} = \langle Q, A, \Delta, q_0, F\rangle$, где $A = \{a, b\}$, $Q = \{q_0, q_1, q_2\}$, $F = \{q_0\}$, а функция переходов задаётся соотношениями

$$\begin{array}{lll} \Delta(q_0, a) = \{q_1, q_2\}, & \Delta(q_1, a) = \varnothing, & \Delta(q_2, a) = \{q_0\}, \\ \Delta(q_0, b) = \varnothing, & \Delta(q_1, b) = \{q_2\}, & \Delta(q_2, b) = \varnothing, \\ \Delta(q_0, \Lambda) = \varnothing, & \Delta(q_1, \Lambda) = \varnothing, & \Delta(q_2, \Lambda) = \{q_0\}, \end{array}$$

имеет следующее графическое изображение:



Данный автомат содержит четыре пути, которые начинаются и заканчиваются в состоянии q_0, но в промежутках не попадают в q_0. Вдоль этих путей распознаются слова a, aa, ab и aba. Поскольку начальное состояние является единственным выделенным, заключаем, что $L(\mathfrak{A}) = \{a, aa, ab, aba\}^*$. Заметим, что слова aa и aba можно выразить через слова a и ab следующим образом: $aa = a{\cdot}a$, $aba = ab{\cdot}a$. Поэтому автомат распознает язык $L(\mathfrak{A}) = \{a, ab\}^*$.

Теорема 22 (об элиминации пустых переходов)**.** *Для любого недетерминированного конечного автомата с пустыми переходами* $\mathfrak{A}$ *существует недетерминированный конечный автомат* $\mathfrak{A}'$ *такой, что* $L(\mathfrak{A}) = L(\mathfrak{A}')$.

Доказательство. Пусть $\mathfrak{A} = \langle Q, A, \Delta, q_0, F\rangle$ — исходный Λ-н.к.а. Для любого состояния $q \in Q$ введём в рассмотрение множество

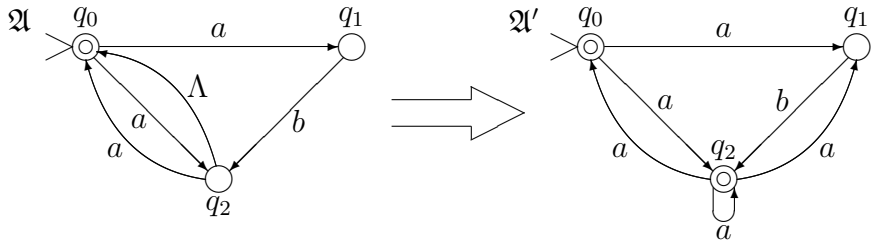
$$E(q) = \{q\} \cup \{p \in Q \mid \text{ в } \mathfrak{A} \text{ существует путь вида } q \xrightarrow{\Lambda} q_1 \xrightarrow{\Lambda} \dots \xrightarrow{\Lambda} q_k \xrightarrow{\Lambda} p\},$$

которое называется *орбитой* состояния q в автомате $\mathfrak{A}$.

Определим н.к.а. $\mathfrak{A}' = \langle Q, A, \Delta', q_0, F' \rangle$, который отличается от исходного автомата только приведёнными ниже функцией переходов Δ' и множеством выделенных состояний F' (см. пример на рисунке):

(а) $\Delta'(q,a) = \bigcup\limits_{p \in E(q)} \Delta(p,a)$, где $q \in Q$ и $a \in A$;

(б) $F' = \{q \in Q \mid E(q) \cap F \neq \varnothing\}$.



Докажем, что для любого слова $w \in A^*$ и любого состояния $p \in Q$ следующие условия эквивалентны:

(а) В Λ-н.к.а. $\mathfrak{A}$ существует путь, который начинается в q_0, заканчивается в p и вдоль дуг которого читается слово w.

(б) В н.к.а. $\mathfrak{A}'$ существует путь, который начинается в q_0, заканчивается в некотором p' с условием $E(p') \ni p$ и вдоль дуг которого читается слово w.

Заметим, что если p — выделенное состояние в $\mathfrak{A}$, то условие (а) эквивалентно условию $w \in L(\mathfrak{A})$, а условие (б) — условию $w \in L(\mathfrak{A}')$. Таким образом, $L(\mathfrak{A}) = L(\mathfrak{A}')$.

Докажем индукцией по длине слова w эквивалентность условий (а) и (б).

1^0. Пусть $|w| = 0$, т.е. $w = \Lambda$. Тогда условие (а) эквивалентно тому, что в $\mathfrak{A}$ существует путь вида

$$q_0 \xrightarrow{\Lambda} q_1 \xrightarrow{\Lambda} \ldots \xrightarrow{\Lambda} q_k \xrightarrow{\Lambda} p$$

(включая случай, когда путь не содержит дуг), что, в свою очередь, эквивалентно условию $p \in E(q_0)$. Покажем, что условие $p \in E(q_0)$ эквивалентно (б). Действительно, если $p \in E(q_0)$, то, взяв $p' = q_0$, очевидно, получим условие (б). Если же в н.к.а. $\mathfrak{A}'$ существует путь, начинающийся в q_0, заканчивающийся в некотором p' с условием $E(p') \ni p$ и вдоль дуг которого читается пустое слово, то с необходимостью $p' = q_0$, и значит, $p \in E(q_0)$.

2^0. Допустим, эквивалентность (а) и (б) уже доказана для произвольных слов w длины k. Докажем эквивалентность (а) и (б) для произвольного слова w длины $k+1$. Пусть $w = s_1 \ldots s_k s_{k+1}$, где $s_i \in A$.

Пусть для слова w выполняется условие (а). Следовательно, существуют состояния $r, p' \in Q$ такие, что в Λ-н.к.а. $\mathfrak{A}$ имеется путь

$$q_0 \xrightarrow{t_1} \ldots \xrightarrow{t_m} r \xrightarrow{s_{k+1}} p' \xrightarrow{\Lambda} \ldots \xrightarrow{\Lambda} p,$$

где $t_1 \ldots t_m = s_1 \ldots s_k$ и $m \geqslant k$. В частности, слово $v = s_1 \ldots s_k$ читается вдоль дуг пути в $\mathfrak{A}$, который начинается в q_0 и заканчивается в r, т.е. для слова v и состояния r выполняется условие (а). В силу индукционного предположения, для v и r выполняется условие (б), т.е. в н.к.а. $\mathfrak{A}'$ существует путь

$$q_0 \xrightarrow{s_1} \ldots \xrightarrow{s_k} r',$$

который начинается в q_0, заканчивается в некотором r' с условием $E(r') \ni r$ и вдоль дуг которого читается слово v. Поскольку Λ-н.к.а. $\mathfrak{A}$ содержит дугу $r \xrightarrow{s_{k+1}} p'$, заключаем, что $p' \in \Delta(r, s_{k+1}) \subseteq \bigcup\limits_{\overline{r} \in E(r')} \Delta(\overline{r}, s_{k+1}) = \Delta'(r', s_{k+1})$. Следовательно, в н.к.а. $\mathfrak{A}'$ существует дуга $r' \xrightarrow{s_{k+1}} p'$. Кроме этого, автомат $\mathfrak{A}$ содержит путь $p' \xrightarrow{\Lambda} \ldots \xrightarrow{\Lambda} p$, откуда следует, что $p \in E(p')$

Таким образом, в н.к.а. $\mathfrak{A}'$ существует путь вида

$$q_0 \xrightarrow{s_1} \ldots \xrightarrow{s_k} r' \xrightarrow{s_{k+1}} p',$$

причём $E(p') \ni p$. Отсюда заключаем, что слово w удовлетворяет условию (б).

Пусть теперь для слова w выполняется условие (б). Следовательно, существует состояние $r' \in Q$ такое, что в н.к.а. $\mathfrak{A}'$ имеется путь вида

$$q_0 \xrightarrow{s_1} \ldots \xrightarrow{s_k} r' \xrightarrow{s_{k+1}} p'$$

и при этом выполняется условие $E(p') \ni p$.

Так как $p' \in \Delta'(r', s_{k+1}) = \bigcup\limits_{r \in E(r')} \Delta(r, s_{k+1})$, то существует состояние $r \in E(r')$ такое, что $p' \in \Delta(r, s_{k+1})$. Следовательно, в Λ-н.к.а. $\mathfrak{A}$ имеется дуга

$$r \xrightarrow{s_{k+1}} p'.$$

Поскольку $r \in E(r')$ и в $\mathfrak{A}'$ есть путь $q_0 \xrightarrow{s_1} \ldots \xrightarrow{s_k} r'$, то для слова $v = s_1 \ldots s_k$ и состояния r выполняется условие (б). Следовательно, в силу индукционного предположения, для v и r выполняется условие (а), т.е. в Λ-н.к.а. $\mathfrak{A}$ существует путь

$$q_0 \xrightarrow{t_1} \ldots \xrightarrow{t_m} r,$$

вдоль дуг которого читается слово $t_1 \ldots t_m = s_1 \ldots s_k = v$ $(m \geqslant k)$.

Так как $p \in E(p')$, то в автомате $\mathfrak{A}$ найдётся путь

$$p' \xrightarrow{\Lambda} \ldots \xrightarrow{\Lambda} p.$$

Окончательно получаем существование в $\mathfrak{A}$ пути

$$q_0 \xrightarrow{t_1} \ldots \xrightarrow{t_m} r \xrightarrow{s_{k+1}} p' \xrightarrow{\Lambda} \ldots \xrightarrow{\Lambda} p,$$

вдоль дуг которого читается слово $t_1 \ldots t_m s_{k+1} = w$, т.е. справедливо условие (а).

Таким образом, построенный н.к.а. $\mathfrak{A}'$ распознает язык $L(\mathfrak{A})$. □

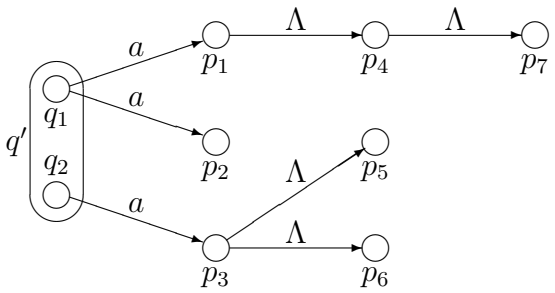
Теорема 23. *Для любого недетерминированного конечного автомата с пустыми переходами $\mathfrak{A}$ существует детерминированный конечный автомат $\mathfrak{A}'$ такой, что $L(\mathfrak{A}) = L(\mathfrak{A}')$.*

Доказательство. Конечно, утверждение теоремы является простым следствием теорем 21 и 22. Тем не менее мы приведём прямую конструкцию, преобразующую произвольный Λ-н.к.а. в эквивалентный ему д.к.а.

Пусть $\mathfrak{A} = \langle Q, A, \Delta, q_0, F \rangle$ — произвольный Λ-н.к.а. Так же как и в доказательстве теоремы 22, для каждого состояния $q \in Q$ обозначим через $E(q)$ его *орбиту* в автомате $\mathfrak{A}$ и определим д.к.а. $\mathfrak{A}' = \langle Q', A, \delta, q_0', F' \rangle$ следующим образом:

(а) $Q' = P(Q)$;
(б) $q'_0 = E(q_0)$;
(в) $F' = \{q' \subseteq Q \mid q' \cap F \neq \varnothing\}$;
(г) для любого $q' \subseteq Q$ и каждого $a \in A$ положим (см. пояснение на рисунке)

$$\delta(q', a) = \bigcup_{q \in q'} \bigcup_{p \in \Delta(q,a)} E(p).$$



$q' = \{q_1, q_2\}$
$\Delta(q_1, a) = \{p_1, p_2\}$
$\Delta(q_2, a) = \{p_3\}$
$E(p_1) = \{p_1, p_4, p_7\}$
$E(p_2) = \{p_2\}$
$E(p_3) = \{p_3, p_5, p_6\}$
$\delta(q', a) = \{p_1, \ldots, p_7\}$

Докажем, что для любого слова $w \in A^*$ и любого состояния $p \in Q$ следующие условия эквивалентны:

(а) В Λ-н.к.а. $\mathfrak{A}$ существует путь, который начинается в q_0, заканчивается в p и вдоль дуг которого читается слово w.

(б) В д.к.а. $\mathfrak{A}'$ существует путь, который начинается в $E(q_0)$, заканчивается в некотором $p' \ni p$ и вдоль дуг которого читается слово w.

Заметим, что если p — выделенное состояние $\mathfrak{A}$, то условие (а) эквивалентно условию $w \in L(\mathfrak{A})$, а условие (б) — условию $w \in L(\mathfrak{A}')$. Таким образом, $L(\mathfrak{A}) = L(\mathfrak{A}')$.

Докажем индукцией по длине слова w эквивалентность условий (а) и (б).

1^0. Пусть $|w| = 0$, т. е. $w = \Lambda$. Тогда условие (а) эквивалентно тому, что в $\mathfrak{A}$ существует путь вида

$$q_0 \xrightarrow{\Lambda} q_1 \xrightarrow{\Lambda} \ldots \xrightarrow{\Lambda} q_k \xrightarrow{\Lambda} p$$

(включая случай, когда путь не содержит дуг), что, в свою очередь, эквивалентно условию $p \in E(q_0)$. Покажем, что условие $p \in E(q_0)$ эквивалентно (б). Действительно, если $p \in E(q_0)$, то, взяв $p' = E(q_0)$, очевидно, получим условие (б). Если же в д.к.а. $\mathfrak{A}'$ существует путь, начинающийся в $E(q_0)$, заканчивающийся в некотором $p' \ni p$ и вдоль дуг которого читается пустое слово, то с необходимостью $p' = E(q_0)$. Следовательно, $p \in E(q_0)$.

2^0. Допустим, эквивалентность (а) и (б) уже доказана для произвольных слов w длины k. Докажем эквивалентность (а) и (б) для произвольного слова w длины $k+1$. Пусть $w = s_1 \ldots s_k s_{k+1}$, где $s_i \in A$.

Пусть для слова w выполняется условие (а). Следовательно, существуют состояния $r_1, r_2 \in Q$ такие, что в Λ-н.к.а. $\mathfrak{A}$ имеется путь

$$q_0 \xrightarrow{t_1} \ldots \xrightarrow{t_m} r_1 \xrightarrow{s_{k+1}} r_2 \xrightarrow{\Lambda} \ldots \xrightarrow{\Lambda} p,$$

где $t_1 \ldots t_m = s_1 \ldots s_k$ и $m \geqslant k$. В частности, слово $v = s_1 \ldots s_k$ читается вдоль дуг пути в $\mathfrak{A}$, который начинается в q_0 и заканчивается в r_1, т.е. для слова v и состояния r_1 выполняется условие (а). В силу индукционного предположения, для v

и r_1 выполняется условие (б), т.е. в д.к.а. $\mathfrak{A}'$ существует путь, который начинается в $E(q_0)$, заканчивается в некотором $r'_1 \ni r_1$ и вдоль дуг которого читается слово v:

$$E(q_0) \xrightarrow{s_1} \ldots \xrightarrow{s_k} r'_1 \ni r_1.$$

Так как $r_2 \in \Delta(r_1, s_{k+1})$ и $r_1 \in r'_1$, то $E(r_2) \subseteq \delta(r'_1, s_{k+1})$. Поскольку $p \in E(r_2)$, заключаем, что $p \in \delta(r'_1, s_{k+1})$. Следовательно, в д.к.а. $\mathfrak{A}'$ существует дуга $r'_1 \xrightarrow{s_{k+1}} p'$, где $p' = \delta(r'_1, s_{k+1})$ и $p' \ni p$.

Таким образом, в $\mathfrak{A}'$ существует путь вида

$$E(q_0) \xrightarrow{s_1} \ldots \xrightarrow{s_k} r'_1 \xrightarrow{s_{k+1}} p' \ni p.$$

Отсюда заключаем, что слово w удовлетворяет условию (б).

Пусть теперь для слова w выполняется условие (б). Следовательно, существует состояние $r'_1 \in Q'$ такое, что в д.к.а. $\mathfrak{A}'$ имеется путь вида

$$E(q_0) \xrightarrow{s_1} \ldots \xrightarrow{s_k} r'_1 \xrightarrow{s_{k+1}} p' \ni p.$$

Так как $p' = \delta(r'_1, s_{k+1}) = \bigcup\limits_{r_1 \in r'_1} \bigcup\limits_{r_2 \in \Delta(r_1, s_{k+1})} E(r_2)$ и $p \in p'$, то существуют состояния $r_1 \in r'_1$ и $r_2 \in \Delta(r_1, s_{k+1})$ такие, что $p \in E(r_2)$.

Поскольку $r_1 \in r'_1$ и в д.к.а. $\mathfrak{A}'$ существует путь

$$E(q_0) \xrightarrow{s_1} \ldots \xrightarrow{s_k} r'_1,$$

то для слова $v = s_1 \ldots s_k$ выполняется условие (б). Следовательно, в силу индукционного предположения, для v и состояния r_1 выполняется условие (а), т.е. в Λ-н.к.а. $\mathfrak{A}$ существует путь

$$q_0 \xrightarrow{t_1} \ldots \xrightarrow{t_m} r_1,$$

вдоль дуг которого читается слово $t_1 \ldots t_m = s_1 \ldots s_k = v$ $(m \geqslant k)$.

Так как $r_2 \in \Delta(r_1, s_{k+1})$, то в $\mathfrak{A}$ имеется дуга

$$r_1 \xrightarrow{s_{k+1}} r_2.$$

Так как $p \in E(r_2)$, то в $\mathfrak{A}$ найдётся путь

$$r_2 \xrightarrow{\Lambda} \ldots \xrightarrow{\Lambda} p.$$

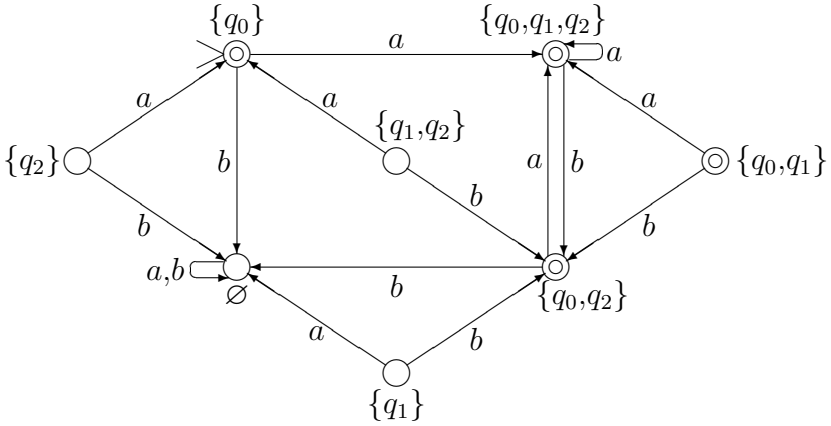
Окончательно получаем существование в $\mathfrak{A}$ пути

$$q_0 \xrightarrow{t_1} \ldots \xrightarrow{t_m} r_1 \xrightarrow{s_{k+1}} r_2 \xrightarrow{\Lambda} \ldots \xrightarrow{\Lambda} p,$$

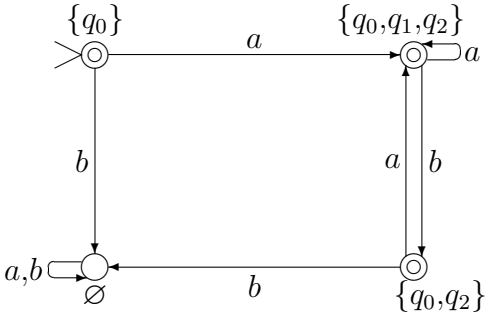
вдоль дуг которого читается слово $t_1 \ldots t_m s_{k+1} = w$, т.е. справедливо условие (а).

Таким образом, построенный д.к.а. $\mathfrak{A}'$ распознаёт язык $L(\mathfrak{A})$. □

Пример (продолжение)**.** Применим конструкцию из доказательства теоремы 23 к Λ-н.к.а. $\mathfrak{A}$ из предыдущего примера — получим следующий д.к.а. $\mathfrak{A}'$, эквивалентный исходному $\mathfrak{A}$:



Заметим, что в полученном автомате состояния $\{q_1\}$, $\{q_2\}$, $\{q_0, q_1\}$ и $\{q_1, q_2\}$ являются недостижимыми, т.е. в них невозможно попасть из начального состояния. Недостижимые состояния можно удалить из автомата, при этом распознаваемый автоматом язык не изменится.



Следствие 24. *Для любого языка L следующие утверждения эквивалентны:*

(1) L распознаётся некоторым детерминированным конечным автоматом.

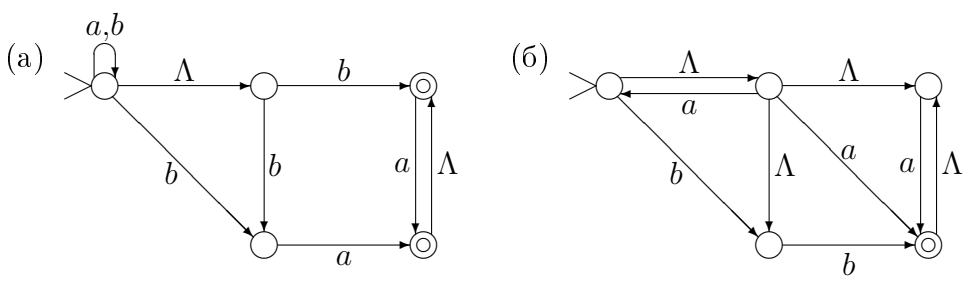
(2) L распознаётся некоторым недетерминированным конечным автоматом.

(3) L распознаётся некоторым недетерминированным конечным автоматом с пустыми переходами.

Доказательство. Импликации (1) $\Rightarrow$ (2) и (2) $\Rightarrow$ (3) очевидны. Справедливость импликации (3) $\Rightarrow$ (1) следует из теоремы 23. □

Упражнения

1. Описать язык, распознаваемый следующим недетерминированным конечным автоматом с пустыми переходами:



2. Используя конструкцию из теоремы 22, построить недетерминированные конечные автоматы, эквивалентные автоматам из упражнения 1.

3. Используя конструкцию из теоремы 23, построить детерминированные конечные автоматы, эквивалентные автоматам из упражнения 1.

§ 12. Свойства автоматных языков

В данном параграфе мы докажем важные теоретико-множественные свойства автоматных языков, а также покажем, что существуют неавтоматные языки.

Определение. Говорят, что подмножество X множества A *замкнуто относительно операции* $f : A^n \to A$, если для любых $x_1, \ldots, x_n \in X$ имеет место $f(x_1, \ldots, x_n) \in X$.

Теорема 25. *Автоматные языки замкнуты относительно объединения, пересечения, дополнения, конкатенации и звёздочки Клини.*

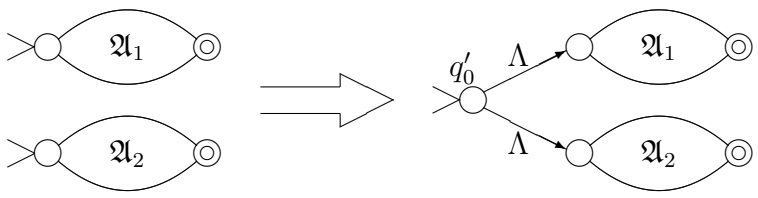
Доказательство. Для каждой из пяти операций мы неформально опишем, как по заданым автоматам, распознающим исходные языки, построить автомат, распознающий результат применения данной операции к исходным языкам. Заметим, что в силу теоремы 23 для каждого из случаев достаточно строить недетерминированный автомат с пустыми переходами.

Объединение. Пусть $\mathfrak{A}_1$, $\mathfrak{A}_2$ — детерминированные конечные автоматы над алфавитом A. Нам необходимо определить автомат $\mathfrak{A}$ такой, что $L(\mathfrak{A}) = L(\mathfrak{A}_1) \cup L(\mathfrak{A}_2)$. Можно считать, что множества состояний $\mathfrak{A}_1$ и $\mathfrak{A}_2$ не пересекаются, в противном случае следует переименовать состояния.

Автомат $\mathfrak{A}$ строится следующим образом (см. схему построения на рисунке):

а) Соединим графические диаграммы автоматов $\mathfrak{A}_1$ и $\mathfrak{A}_2$ в одну, введя новое начальное состояние q_0' и добавив две дуги, выходящие из q_0', входящие в старые начальные состояния и помеченные символом Λ. (Считаем, что начальные состояния исходных автоматов перестают быть таковыми.)

б) Множеством выделенных состояний $\mathfrak{A}$ будет объединение множеств выделенных состояний автоматов $\mathfrak{A}_1$ и $\mathfrak{A}_2$.



Такое описание однозначно задаёт автомат $\mathfrak{A}$. Рассмотрим произвольный путь по дугам автомата $\mathfrak{A}$, который начинается в q_0' и заканчивается в выделенном состоянии. Первым переходом такого пути обязан быть Λ-переход, а остальные переходы полностью содержатся внутри либо автомата $\mathfrak{A}_1$, либо автомата $\mathfrak{A}_2$. Следовательно, любое слово, распознаваемое $\mathfrak{A}$, — это слово, распознаваемое $\mathfrak{A}_1$ или $\mathfrak{A}_2$. С другой стороны, любое слово, распознаваемое $\mathfrak{A}_1$ или $\mathfrak{A}_2$, очевидно, распознаётся автоматом $\mathfrak{A}$. Таким образом, автомат $\mathfrak{A}$ — искомый.

Дополнение. Пусть д.к.а. $\mathfrak{A} = \langle Q, A, \delta, q_0, F\rangle$ распознаёт язык L, т. е. для любого слова $w \in A^*$ имеет место эквивалентность $w \in L \iff \delta^*(q_0, w) \in F$. Отсюда вытекает эквивалентность $w \notin L \iff \delta^*(q_0, w) \notin F$. Другими словами, имеет место условие $w \in A^* \setminus L \iff \delta^*(q_0, w) \in Q \setminus F$. Отсюда делаем вывод, что д.к.а $\mathfrak{A}' = \langle Q, A, \delta, q_0, Q \setminus F\rangle$ распознаёт дополнение $\overline{L} = A^* \setminus L$.

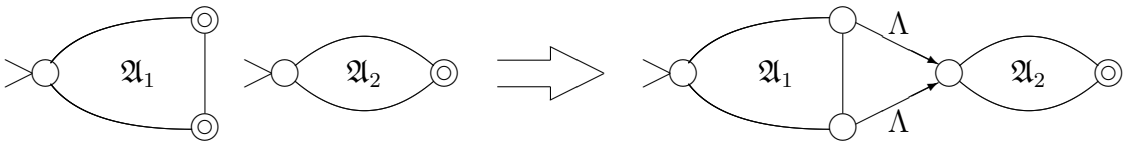
Пересечение. Замкнутость автоматных языков относительно пересечения следует из замкнутости относительно объединения и дополнения, а также из теоретико-множественного тождества $X \cap Y = \overline{\overline{X} \cup \overline{Y}}$.

Конкатенация. Пусть $\mathfrak{A}_1$, $\mathfrak{A}_2$ — детерминированные конечные автоматы над алфавитом A, множества состояний которых не пересекаются. Построим автомат $\mathfrak{A}$, распознающий язык $L(\mathfrak{A}_1)L(\mathfrak{A}_2)$, следующим образом (см. схему построения на рисунке):

а) Соединим графические диаграммы автоматов $\mathfrak{A}_1$ и $\mathfrak{A}_2$, добавив новые дуги, выходящие из всех выделенных состояний автомата $\mathfrak{A}_1$, входящие в начальное состояние автомата $\mathfrak{A}_2$ и помеченные символом Λ.

б) Начальным состоянием полученного автомата объявляется начальное состояние $\mathfrak{A}_1$.

в) Выделенными состояниями полученного автомата объявляются все выделенные состояния автомата $\mathfrak{A}_2$. Других выделенных состояний нет.



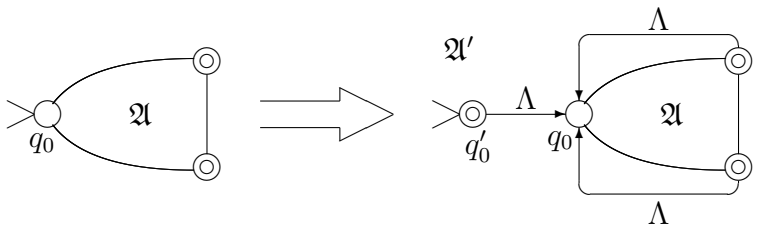
Любой путь вдоль дуг нового автомата, начинающийся в начальном состоянии и заканчивающийся в выделенном состоянии, разбивается на два участка. Первый участок состоит только из дуг автомата $\mathfrak{A}_1$ и заканчивается в одном из (бывших) выделенных состояний $\mathfrak{A}_1$. Второй участок начинается с Λ-перехода, остальные его переходы полностью содержатся внутри автомата $\mathfrak{A}_2$, и заканчивается в одном из выделенных состояний нового автомата. Отсюда следует, что $L(\mathfrak{A}) = L(\mathfrak{A}_1)L(\mathfrak{A}_2)$.

Звёздочка Клини. По заданному д.к.а. $\mathfrak{A}$ построим автомат $\mathfrak{A}'$, распознающий язык $(L(\mathfrak{A}))^*$, следующим образом (см. схему построения на рисунке):

а) Введём новое начальное состояние q_0', сделав его одновременно выделенным и добавив новую дугу, выходящую из q_0', входящую в начальное состояние q_0 автомата $\mathfrak{A}$ и помеченную символом Λ.

б) Добавим в автомат $\mathfrak{A}$ новые дуги, выходящие из всех выделенных состояний автомата $\mathfrak{A}$, входящие в q_0 и помеченные символом Λ.

в) Выделенными состояниями автомата $\mathfrak{A}'$ объявляются все выделенные состояния $\mathfrak{A}$ и состояние q_0'.



Докажем, что $(L(\mathfrak{A}))^* = L(\mathfrak{A}')$. Для этого сначала установим справедливость включения $(L(\mathfrak{A}))^* \subseteq L(\mathfrak{A}')$. Пусть слово $w \in (L(\mathfrak{A}))^*$. Если $w = \Lambda$, то $w \in L(\mathfrak{A}')$ в силу пункта (а). Если же $w \neq \Lambda$, то слово w представимо в виде $w = w_1 \dots w_n$, где $w_i \in L(\mathfrak{A})$ для каждого $1 \leqslant i \leqslant n$. Следовательно, для каждого $1 \leqslant i \leqslant n$ слово w_i читается вдоль следующего пути в автомате $\mathfrak{A}$:

$$q_0 = r_0^i \xrightarrow{s_1^i} r_1^i \xrightarrow{s_2^i} \dots \xrightarrow{s_{k_i}^i} r_{k_i}^i \in F,$$

в котором последнее состояние $r_{k_i}^i$ является выделенным в $\mathfrak{A}$. В силу пункта (а) для $i = 1$ в автомате $\mathfrak{A}'$ существует Λ-переход из q_0' в $q_0 = r_0^1$. Следовательно, слово w_1 будет читаться вдоль следующего пути в автомате $\mathfrak{A}'$:

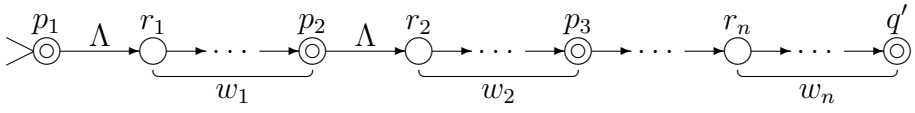
$$q_0' \xrightarrow{\Lambda} r_0^1 \xrightarrow{s_1^1} r_1^1 \xrightarrow{s_2^1} \dots \xrightarrow{s_{k_1}^1} r_{k_1}^1.$$

В силу пункта (б) для $2 \leqslant i \leqslant n$ в новом автомате $\mathfrak{A}'$ существует Λ-переход из состояния $r_{k_{i-1}}^{i-1}$ в состояние $r_0^i = q_0$. Таким образом, для каждого $2 \leqslant i \leqslant n$ слово w_i будет читаться вдоль следующего пути в автомате $\mathfrak{A}'$:

$$r_{k_{i-1}}^{i-1} \xrightarrow{\Lambda} r_0^i \xrightarrow{s_1^i} r_1^i \xrightarrow{s_2^i} \dots \xrightarrow{s_{k_i}^i} r_{k_i}^i.$$

Соединив последовательно все такие пути в одну цепочку, мы получим путь в автомате $\mathfrak{A}'$, который начинается в состоянии q_0', заканчивается в некотором выделенном состоянии автомата $\mathfrak{A}'$ и вдоль дуг которого читается w. Следовательно, $w \in L(\mathfrak{A}')$.

Теперь установим обратное включение $L(\mathfrak{A}') \subseteq (L(\mathfrak{A}))^*$. Пусть $w \in L(\mathfrak{A}')$. Можно считать, что $w \neq \Lambda$ (случай пустого слова тривиален). Следовательно, w читается вдоль пути в автомате $\mathfrak{A}'$, который начинается в состоянии q_0' и заканчивается в некотором выделенном состоянии q'. Поскольку в $\mathfrak{A}'$ нет дуг, входящих в состояние q_0', заключаем, что $q_0' \neq q'$. Поэтому состояние q' является выделенным и в исходном автомате $\mathfrak{A}$.



Поскольку $w \neq \Lambda$, первой дугой данного пути обязан быть Λ-переход $q_0' \xrightarrow{\Lambda} q_0$. Пусть в данном пути встречается ровно n пустых переходов. Для $1 \leqslant i \leqslant n$ введём следующие обозначения. Пусть i-й пустой переход, встретившийся в данном пути, имеет вид $p_i \xrightarrow{\Lambda} r_i$. Тогда $p_1 = q_0'$ и для каждого $2 \leqslant i \leqslant n$ состояние p_i является выделенным в исходном автомате $\mathfrak{A}$. Кроме этого, $r_i = q_0$ для всех $1 \leqslant i \leqslant n$. Для $1 \leqslant i \leqslant n-1$ обозначим через w_i слово, которое читается вдоль участка пути, начинающегося в r_i и заканчивающегося в p_{i+1}. Через w_n обозначим слово, которое читается вдоль участка пути, начинающегося в r_n и заканчивающегося в q'.

Для каждого $1 \leqslant i \leqslant n$ участок пути, вдоль которого читается w_i, полностью содержится в автомате $\mathfrak{A}$, начинается в состоянии q_0 и заканчивается в некотором выделенном состоянии автомата $\mathfrak{A}$. Следовательно, $w_i \in L(\mathfrak{A})$ для всех $1 \leqslant i \leqslant n$. Таким образом, $w = w_1 \dots w_n \in (L(\mathfrak{A}))^*$. $\square$

Замечание. В предыдущей теореме можно предложить *прямое* доказательство замкнутости автоматных языков относительно операции пересечения, а именно: если заданы два д.к.а. $\mathfrak{A}_1 = \langle Q_1, A, \delta_1, q_0^1, F_1\rangle$ и $\mathfrak{A}_2 = \langle Q_2, A, \delta_2, q_0^2, F_2\rangle$, то определим д.к.а. $\mathfrak{A}_1 \times \mathfrak{A}_2$, который называется *прямым произведением* исходных автоматов, следующим образом: $\mathfrak{A}_1 \times \mathfrak{A}_2 = \langle Q_1 \times Q_2, A, \delta, \langle q_0^1, q_0^2\rangle, F_1 \times F_2\rangle$, где функция перехода $\delta(\langle q_1, q_2\rangle, a) = \langle \delta_1(q_1, a), \delta_2(q_2, a)\rangle$ для любых $q_1 \in Q_1$, $q_2 \in Q_2$, $a \in A$. Несложно убедиться в том, что $L(\mathfrak{A}_1 \times \mathfrak{A}_2) = L(\mathfrak{A}_1) \cap L(\mathfrak{A}_2)$.

Теорема 26. *Любой конечный язык является автоматным.*

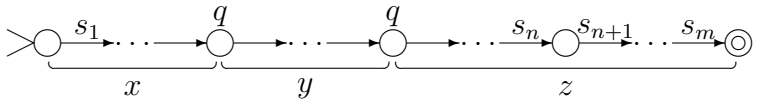
Доказательство. а) Нетрудно построить в явном виде автоматы, распознающие языки $\varnothing$, $\{\Lambda\}$, $\{a\}$, где a — буква.

б) Из (а) и теоремы 25 (конкатенация) следует, что любой язык вида $\{w\}$, где w — слово, является автоматным.

в) Из (б) и теоремы 25 (объединение) следует, что любой непустой конечный язык является автоматным. □

Лемма 27 (о накачивании)**.** *Пусть L — автоматный язык. Тогда найдётся $n \geqslant 1$ такое, что для любого слова $w \in L$, где $|w| \geqslant n$, существует представление в виде $w = xyz$, где $y \neq \Lambda$, $|xy| \leqslant n$ и $xy^iz \in L$ для всех $i \geqslant 0$.*

Доказательство. Пусть $\mathfrak{A}$ — д.к.а. такой, что $L(\mathfrak{A}) = L$, и пусть n — число состояний автомата $\mathfrak{A}$. Рассмотрим произвольное слово $w \in L$ со свойством $|w| \geqslant n$. Следовательно, можно представить $w = s_1 \dots s_n s_{n+1} \dots s_m$, где s_i — буквы. Так как w распознаётся автоматом $\mathfrak{A}$, то существует путь по дугам $\mathfrak{A}$, начинающийся в начальном состоянии, заканчивающийся в выделенном состоянии и вдоль которого читается слово w.



Рассмотрим первые n переходов в этом пути, вдоль которых читаются первые n букв слова w. Так как число состояний, пройденных на этом участке пути, равно $n + 1$, то существует хотя бы одно состояние q, которое встречается не менее двух раз.

Пусть x — часть слова $s_1 \dots s_n$, которая читается от начального состояния до первого попадания в состояние q; y — часть слова $s_1 \dots s_n$, которая читается от первого попадания в состояние q до последнего попадания в состояние q; z — остальная часть w (см. рисунок). Тогда $y \neq \Lambda$ и $|xy| \leqslant n$. Кроме этого, чтение подслова y начинается и заканчивается в состоянии q. Следовательно, этот участок можно удалить из нашего пути или пройти по нему произвольное количество раз. Отсюда заключаем, что слово $xy^iz \in L(\mathfrak{A})$ для всех $i \geqslant 0$. □

Замечание. Формальная запись утверждения леммы о накачивании имеет достаточно сложную логическую структуру:

$$\forall L\Big(L \text{ — автоматный } \longrightarrow \exists n\Big(n \geqslant 1 \;\&\; \forall w\big[(w \in L \;\&\; |w| \geqslant n) \longrightarrow$$
$$\exists x, y, z(w = xyz \;\&\; y \neq \Lambda \;\&\; |xy| \leqslant n \;\&\; \forall i(i \geqslant 0 \to xy^iz \in L))\big]\Big)\Big).$$

С точностью до эквивалентности записанная выше формула имеет кванторную приставку $\forall L \exists n \forall w \exists x, y, z \forall i$ с пятью группами чередующихся кванторов. Важно также помнить, что если в логической формуле вида $\Phi \to \Psi$ посылка Φ ложна, то независимо от значения Ψ формула $\Phi \to \Psi$ истинна. В частности, утверждение леммы о накачивании остаётся верным и для слов $w \in L$ с условием $|w| < n$.

Следствие 28. *Существуют неавтоматные языки.*

Доказательство. Рассмотрим язык $L = \{a^m b^m \mid m \in \omega\}$ над алфавитом $\{a, b\}$. Допустим, L — автоматный. Следовательно, существует $n \geqslant 1$, как в лемме о накачивании. Рассмотрим слово $a^n b^n \in L$. Длина $|a^n b^n| > n$. Тогда по лемме можно представить $a^n b^n = xyz$, где $y \neq \Lambda$, $|xy| \leqslant n$ и $xy^i z \in L$ для любого $i \geqslant 0$. Отсюда следует, что существует $k \geqslant 1$ такое, что $y = a^k$, и слово x не содержит букв b. Следовательно, слово $xz = a^{n-k} b^n \in L$, что невозможно, поскольку $n - k < n$. Таким образом, L не является автоматным. □

Замечание. Лемма о накачивании — это необходимое условие автоматности языка. Это условие является достаточно сильным и демонстрирует определённую ограниченность вычислительных возможностей конечных автоматов. Например, как мы заметили, никакой конечный автомат не способен распознать язык $\{a^m b^m \mid m \in \omega\}$. Однако несложно построить формальную грамматику, которая порождает этот язык. Другими словами, не любую алгоритмически разрешимую задачу можно решить с помощью конечных автоматов. Тем не менее язык конечных автоматов оказывается достаточным для алгоритмического описания многих важнейших классов задач в различных разделах математики и приложениях.

Упражнения

1. Доказать, что прямое произведение $\mathfrak{A}_1 \times \mathfrak{A}_2$ детерминированных конечных автоматов $\mathfrak{A}_1$ и $\mathfrak{A}_2$ распознаёт язык $L(\mathfrak{A}_1) \cap L(\mathfrak{A}_2)$.

2. Почему доказательство теоремы 25 (звёздочка Клини) будет неверным, если просто сделать начальное состояние q_0 выделенным и добавить новые Λ-переходы из всех выделенных состояний в состояние q_0 (не вводя новое начальное состояние q_0')? Привести пример автомата, показывающий, что такое доказательство не верно.

3. Доказать, что следующие языки не являются автоматными:
   (а) $\{a^n \mid n \in \omega, n$ — простое$\}$;
   (б) $\{a^{n^2} \mid n \in \omega\}$;
   (в) $\{ww^R \mid w \in \{a, b\}^*\}$;
   (г) $\{ww \mid w \in \{a, b\}^*\}$;
   (д) $\{w \in \{a, b\}^* \mid w$ имеет одинаковое число вхождений a и $b\}$;
   (е) $\{1^m 0 1^n 0 1^{m+n} \mid m, n \in \omega, m, n \geqslant 1\}$.

4. *Арифметической прогрессией* назовём любое множество $\{p + qn \mid n \in \omega\}$, где p и q — некоторые натуральные числа.
   (а) Доказать, что если $L \subseteq \{a\}^*$ и $\{n \in \omega \mid a^n \in L\}$ является арифметической прогрессией, то L — автоматный язык;

(б) Доказать, что если $L \subseteq \{a\}^*$ и $\{n \in \omega \mid a^n \in L\}$ является объединением конечного числа арифметических прогрессий, то L — автоматный язык;

(в) Доказать, что если $L \subseteq \{a\}^*$ — автоматный язык, то $\{n \in \omega \mid a^n \in L\}$ является объединением конечного числа арифметических прогрессий.

§ 13. Регулярные выражения и языки

В этом параграфе будет предложен другой подход для описания класса автоматных языков. Будет доказано, что автоматные языки — это в точности те языки, которые имеют «синтаксическое» описание в терминах регулярных выражений.

Определение. Пусть A — конечный алфавит, не содержащий символов $(,)$, $\cup$, $*$. Определим по индукции множество *регулярных выражений* над алфавитом A:

1^0. Множества $\varnothing$ и a, где $a \in A$, являются *регулярными выражениями.*

2^0. Если α и β — регулярные выражения, то $(\alpha\beta)$, $(\alpha \cup \beta)$ и (α^*) тоже являются *регулярными выражениями.*

Таким образом, слово в алфавите $A \cup \{(,), \cup, ^*\}$ называется регулярным выражением, если оно может быть получено конечным числом применений пунктов 1^0 и 2^0.

Замечание. В дальнейшем мы будем опускать некоторые (в том числе внешние) скобки при записи регулярных выражений, как это делается в обычной алгебре, считая, что операции имеют следующий приоритет: α^* — самая сильная операция, далее идёт $\alpha\beta$, а операция $\alpha \cup \beta$ — самая слабая. Например, запись $\alpha^*\beta \cup \beta\alpha$ на самом деле означает $(((\alpha^*)\beta) \cup (\beta\alpha))$.

Определение. Определим отображение L из множества всех регулярных выражений над алфавитом A в множество всех языков над A следующим образом:

$$
\begin{aligned}
&L(\varnothing) = \varnothing,\\
&L(a) = \{a\}, \text{ для любого } a \in A,\\
&L(\alpha\beta) = L(\alpha)L(\beta),\\
&L(\alpha \cup \beta) = L(\alpha) \cup L(\beta),\\
&L(\alpha^*) = L(\alpha)^*.
\end{aligned}
$$

Определение. Язык L над алфавитом A называется *регулярным*, если существует регулярное выражение α над алфавитом A такое, что $L(\alpha) = L$. При этом будем говорить, что выражение α *задаёт* язык L.

Замечание. Семейство всех регулярных языков над алфавитом A можно определить, не прибегая к понятию регулярного выражения. Такое определение эквивалентно предыдущему и использует индукцию по количеству применений операций конкатенации, объединения и звёздочки Клини:

1^0. Языки $\varnothing$ и $\{a\}$, где $a \in A$, являются *регулярными* над алфавитом A.

2^0. Если L_1 и L_2 — регулярные языки над алфавитом A, то языки L_1L_2, $L_1 \cup L_2$ и L_1^* тоже являются *регулярными* над алфавитом A.

Замечание. Из тождества $\varnothing^* = \{\Lambda\}$ следует, что язык $\{\Lambda\}$, состоящий из одного пустого слова, регулярен. Если $a_1, \ldots, a_s$ — произвольные символы из алфавита, то из тождества $\{a_1, \ldots, a_s\} = \{a_1\} \cup \ldots \cup \{a_s\}$ следует регулярность языка $\{a_1, \ldots, a_s\}$.

Пример. Язык $L = \{w \in \{a, b\}^* \mid w$ содержит подслово $bb\}$ является регулярным, поскольку его можно задать регулярным выражением $(a^* \cup ba)^* bb(a \cup b)^*$. Заметим, что для любого регулярного языка существует бесконечно много регулярных выражений, задающих его. Например, тот же язык L можно задать регулярным выражением $(a \cup b)^* bb(a \cup b)^*$.

Теорема 29. *Класс автоматных языков совпадает с классом регулярных языков.*

Доказательство. Сначала докажем, что любой регулярный язык автоматен. Пусть L — регулярный язык над алфавитом A. Следовательно, найдётся хотя бы одно регулярное выражение γ, которое задаёт L. Индукцией по длине выражения γ докажем, что L — автоматный.

1^0. Если γ является выражением вида $\varnothing$ или a, где $a \in A$, т.е. $L = \varnothing$ или $L = \{a\}$, то в силу теоремы 26 язык L является автоматным.

2^0. Если γ является выражением вида $(\alpha\beta)$, $(\alpha \cup \beta)$ или (α^*), где α, β — регулярные, то по индукционному предположению заключаем, что языки $L_1 = L(\alpha)$ и $L_2 = L(\beta)$ — автоматные. Тогда $L = L_1 L_2$, или $L = L_1 \cup L_2$, или $L = L_1^*$ соответственно. В любом случае по теореме 25 получаем, что L автоматен.

Теперь докажем, что любой автоматный язык регулярен. Пусть L — произвольный автоматный язык. Следовательно, существует д.к.а. $\mathfrak{A} = \langle Q, A, \delta, q_1, F\rangle$ с состояниями $Q = \{q_1, \ldots, q_n\}$ такой, что $L(\mathfrak{A}) = L$. Докажем, что L — регулярный. Для этого определим для всех $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant n$ и $0 \leqslant k \leqslant n$ множество слов:

$$R(i, j, k) = \{w \in A^* \mid w \text{ читается вдоль пути автомата } \mathfrak{A}, \text{который начинается в } q_i, \text{заканчивается в } q_j \text{ и который в промежутке между ними не заходит в состояния } q_{k+1}, q_{k+2}, \ldots, q_n\}.$$

(Здесь термином «промежуток между q_i и q_j» мы называем множество всех состояний пути, исключая его начало q_i и его конец q_j. Напомним также, что пустое слово Λ по определению читается вдоль пути, который содержит только одно состояние и не содержит ни одной дуги.)

Заметим, что при $k = n$ множество $R(i, j, n)$ состоит в точности из всех слов, читаемых вдоль путей нашего автомата, идущих из q_i в q_j. Кроме этого, ясно, что

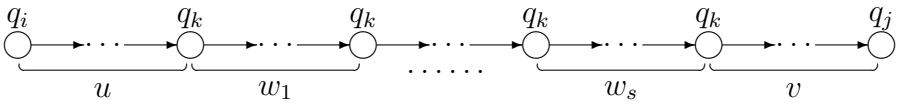
$$L = L(\mathfrak{A}) = \bigcup_{q_j \in F} R(1, j, n).$$

Так как регулярные языки замкнуты относительно объединения, то достаточно доказать, что все $R(i, j, k)$ регулярны. Докажем это утверждение индукцией по k.

1^0. При $k = 0$ множество $R(i, j, 0)$ — это все слова, читаемые вдоль путей, которые начинаются в q_i, заканчиваются в q_j и в промежутке между q_i и q_j не заходят ни в одно состояние. Возможны три случая. Если существуют дуги, ведущие из q_i в q_j и помеченные символами $a_1, \ldots, a_s$, то $R(i, j, 0) = \{a_1, \ldots, a_s\}$. Если таких дуг нет и $q_i = q_j$, то $R(i, j, 0) = \{\Lambda\}$. Если таких дуг нет и $q_i \neq q_j$, то $R(i, j, 0) = \varnothing$. Все такие языки регулярны, в силу определения и замечания о языках $\{\Lambda\}$ и $\{a_1, \ldots, a_s\}$.

2^0. Допустим, что утверждение доказано для $k - 1$, т. е. все языки $R(i, j, k - 1)$ регулярны. Рассмотрим произвольное слово $w \in R(i, j, k)$, оно читается вдоль пути,

который начинается в q_i, несколько раз (может и 0 раз) заходит в q_k и заканчивается в q_j. Если этот путь не заходит в q_k, то вдоль него читается слово из $R(i,j,k-1)$, т. е. $w \in R(i,j,k-1)$. Если же этот путь заходит в q_k, то пусть u — подслово w, которое читается вдоль участка пути, начинающегося в q_i и заканчивающегося в первом попадании в состояние q_k; w_1 — подслово w, которое читается вдоль участка, начинающегося в первом попадании в q_k и заканчивающегося во втором попадании в q_k; …; w_s — подслово w, которое читается вдоль участка, начинающегося в предпоследнем попадании в q_k и заканчивающегося в последнем попадании в q_k; и, наконец, пусть v — подслово w, которое читается вдоль участка, начинающегося в последнем попадании в q_k и заканчивающегося в q_j.



Тогда $u \in R(i,k,k-1)$, $w_1, \ldots, w_s \in R(k,k,k-1)$ и $v \in R(k,j,k-1)$. Отсюда следует, что слово $w = uw_1 \ldots w_s v \in R(i,k,k-1)R(k,k,k-1)^*R(k,j,k-1)$. Объединяя оба случая, заключаем, что имеет место включение

$$R(i,j,k) \subseteq R(i,j,k-1) \cup R(i,k,k-1)R(k,k,k-1)^*R(k,j,k-1).$$

Нетрудно проверить, что обратное включение тоже верно. Таким образом, мы доказали равенство

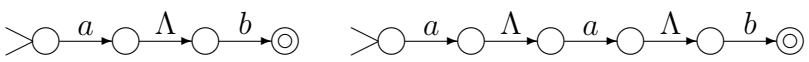
$$R(i,j,k) = R(i,j,k-1) \cup R(i,k,k-1)R(k,k,k-1)^*R(k,j,k-1).$$

Следовательно, в силу индукционного предположения и замкнутости регулярных языков относительно объединения, конкатенации и звёздочки Клини окончательно получаем, что $R(i,j,k)$ — регулярный. Что и требовалось доказать. □

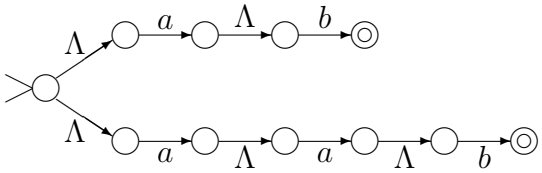
Пример. Используя методы из теоремы 25, построим по регулярному выражению $\alpha = (ab \cup aab)^*$ автомат, распознающий язык $L(\alpha)$.

На первом шаге строим н.к.а. для регулярных выражений a и b:

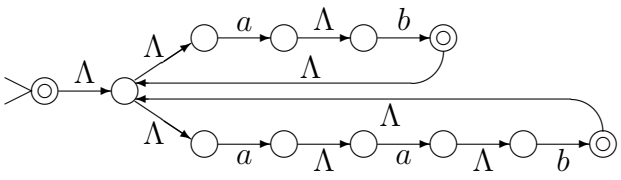
На втором шаге строим Λ-н.к.а. для регулярных выражений ab и aab:



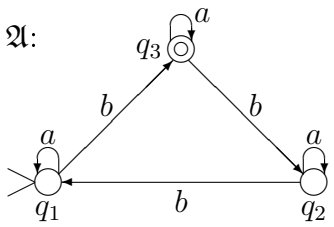
На третьем шаге из полученных автоматов строим Λ-н.к.а. для регулярного выражения $ab \cup aab$:



На четвёртом шаге мы строим итоговый Λ-н.к.а. для регулярного выражения $(ab \cup aab)^*$:



Пример. Пусть автомат $\mathfrak{A}$ задан своей графической диаграммой (см. рисунок). Легко видеть, что $L(\mathfrak{A}) = \{w \in \{a,b\}^* \mid w$ имеет $3k+1$ вхождений символа b для некоторого $k \in \omega\}$.



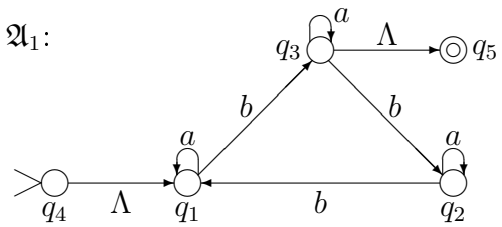
Используя метод доказательства теоремы 29, найдём регулярное выражение, которое задаёт язык $L(\mathfrak{A})$. Если применять данный метод напрямую, необходимо вычислить 36 регулярных выражений $R(i,j,k)$, где $1 \leqslant i \leqslant 3$, $1 \leqslant j \leqslant 3$, $0 \leqslant k \leqslant 3$. Заметим, что доказательство теоремы 29 справедливо не только для д.к.а., но и в общем случае для Λ-н.к.а. Можно упростить вычисления, добившись следующих свойств автомата:

(а) Автомат имеет только одно выделенное состояние;

(б) Не существует дуг, входящих в начальное состояние;

(в) Не существует дуг, выходящих из выделенного состояния.

Для выполнения этих свойств добавим в автомат новое начальное состояние q_4, единственное новое выделенное состояние q_5 и необходимые Λ-переходы. Полученный автомат $\mathfrak{A}_1$ эквивалентен исходному $\mathfrak{A}$.

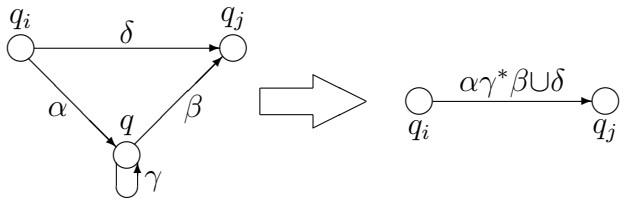


Таким образом, искомое регулярное выражение совпадает с $R(4,5,5)$. Мы будем последовательно вычислять все выражения $R(i,j,0)$, затем $R(i,j,1)$ и т. д. На каждом шаге будем помечать регулярным выражением $R(i,j,k)$ дугу, выходящую из q_i и входящую в q_j. При этом, очевидно, можно опускать дуги, помеченные символом $\varnothing$, и петли, помеченные символом Λ.

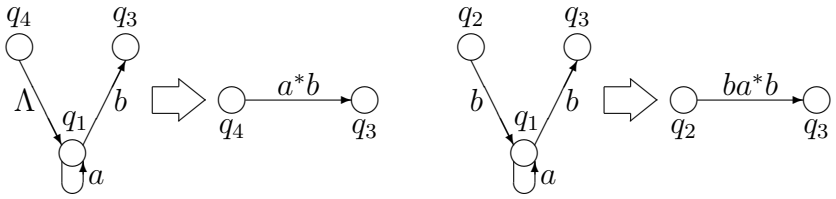
Дуги автомата $\mathfrak{A}_1$ уже помечены правильными значениями выражений $R(i,j,0)$. (Если бы в автомате было несколько дуг, идущих из q_i в q_j, то их следовало бы объединить в одну, используя операцию $\cup$ для регулярных выражений.)

Будем вычислять $R(i,j,k)$, где $k \geqslant 1$, одновременно элиминируя состояние q_k из нашего автомата. Действительно, если все $R(i,j,k)$ уже найдены и из автомата элиминированы состояния $q_1, \ldots, q_{k-1}$, то все пути, проходящие через q_k, уже учтены в выражениях $R(i,j,k)$.

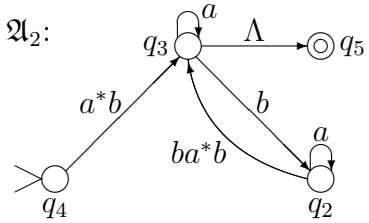
Принцип *элиминации* состояния можно описать в общем случае следующим образом. Пусть даны состояния $q_i \neq q$ и $q_j \neq q$ такие, что из q_i в q ведёт дуга, помеченная выражением α; из q в q_j ведёт дуга, помеченная выражением β; из q_i в q_j ведёт дуга, помеченная выражением δ; и, кроме этого, в состоянии q есть петля, помеченная выражением γ. (Состояния q_i и q_j могут совпадать, выражения γ и δ могут быть пустыми.) В таком случае состояние q можно элиминировать, оставив только одну дугу, выходящую из q_i, входящую в q_j и помеченную выражением $\alpha\gamma^*\beta \cup \delta$.



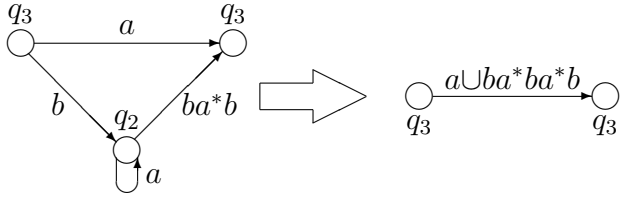
Элиминируем состояние q_1 в автомате $\mathfrak{A}_1$. Для этого найдём все пары состояний $\langle q_i, q_j\rangle$ такие, что существует путь из q_i в q_j, который в промежутке заходит только в состояние q_1. Для каждой такой пары (в нашем случае это $\langle q_4, q_3\rangle$ и $\langle q_2, q_3\rangle$) используем описанный выше принцип элиминации q_1:



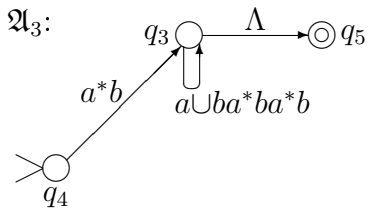
Отсюда получаем автомат $\mathfrak{A}_2$, в котором уже нет состояния q_1:



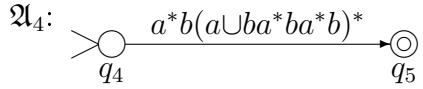
Элиминируем состояние q_2 в автомате $\mathfrak{A}_2$. Для этого найдём все пары состояний $\langle q_i, q_j\rangle$ такие, что существует путь из q_i в q_j, который в промежутке заходит только в состояние q_2. Этому условию удовлетворяет только пара $\langle q_3, q_3\rangle$ — используем для неё принцип элиминации q_2:



Отсюда получаем автомат $\mathfrak{A}_3$, в котором уже отсутствует состояние q_2:



Наконец, элиминируем q_3 в автомате $\mathfrak{A}_3$ — получим следующий автомат $\mathfrak{A}_4$:

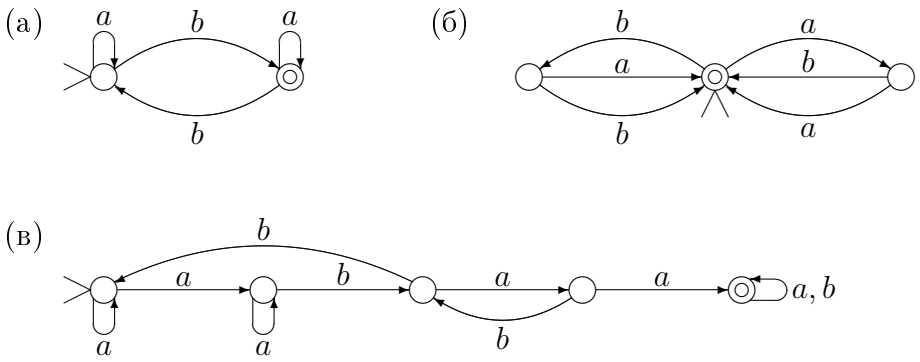


Автомат $\mathfrak{A}_4$ содержит только одну дугу, выходящую из начального состояния и входящую в выделенное состояние. Таким образом, искомое регулярное выражение имеет вид $a^*b(a \cup ba^*ba^*b)^*$.

Упражнения

1. Используя методы из теоремы 25, построить автомат, распознающий язык $L(\alpha)$ для следующего регулярного выражения α:
   (а) $\alpha = ((ab \cup aab)^*a^*)^*$;
   (б) $\alpha = ((a^*b^*a^*)^*b)^*$;
   (в) $\alpha = ((ab)^* \cup (bc)^*)ab$.

2. Используя метод доказательства теоремы 29, найти регулярное выражение, которое задаёт язык $L(\mathfrak{A})$, где $\mathfrak{A}$ — следующий автомат:



3. Пусть L — регулярный язык над алфавитом A, L' — произвольный язык над алфавитом A. Доказать, что следующие языки тоже являются регулярными:
   (а) $\mathrm{Pref}(L) = \{w \in A^* \mid \exists u \in A^*(wu \in L)\}$ (множество префиксов L);
   (б) $\mathrm{Subseq}(L) = \{w_1w_2 \dots w_k \mid k \in \omega, w_i \in A$ для $1 \leqslant i \leqslant k$ и $\exists u_0, u_1, \dots, u_k \in A^*$ $(u_0w_1u_1w_2u_2 \dots w_ku_k \in L)\}$ (множество подпоследовательностей L);
   (в) $L/L' = \{w \in A^* \mid \exists u \in L'(wu \in L)\}$ (правый фактор языка L по L').

§ 14. Формальные грамматики

Часто формальные языки описываются как совокупности слов, полученных с помощью правил замены одних цепочек символов на другие. Такой подход лежит в основе понятия формальной грамматики. В отличие от конечных автоматов, формальные грамматики не распознают слова, а порождают их. Любую формальную грамматику можно представлять как некоторый генератор языка. Запуск такого генератора производится с помощью «стартового сигнала», после чего начинается порождение слов языка. Действия генератора на каждом этапе порождения слов недетерминированны, но ограничены конечным списком правил. Время от времени этот процесс прерывается для того, чтобы выдать очередное выходное слово. Множество всех слов, появившихся на выходе в процессе работы, образует язык, порождённый данной грамматикой. Процесс порождения слов по правилам формальной грамматики безусловно является алгоритмическим.

Пример. Рассмотрим алгоритм порождения слов со сбалансированными скобками над алфавитом, состоящим из открывающей скобки «(» и закрывающей скобки «)». *Слова со сбалансированными скобками* (кратко: *сбалансированные* слова) определяются по индукции:

(1) Слово () сбалансированно.

(2) Если слово u сбалансированно, то слово (u) тоже сбалансированно.

(3) Если слова u и v сбалансированны, то слово uv тоже сбалансированно.

Введём вспомогательный символ S, которым будем обозначать любое сбалансированное слово. Алгоритм можно описать следующими правилами:

(1) Слово S можно заменить на слово ().

(2) Слово S можно заменить на слово (S).

(3) Слово S можно заменить на слово SS.

Любое сбалансированное слово получается с помощью конечного числа применений правил (1)–(3). Схематически правила (1)–(3) можно переписать следующим образом:

$$S \longrightarrow ()$$
$$S \longrightarrow (S)$$
$$S \longrightarrow SS$$

Например, процесс порождения слова (()())()() по данным правилам выглядит так:

$$S \to SS \to S() \to (S)() \to (SS)() \to (SSS)() \to (()SS)() \to (()()S)() \to (()()())().$$

Определение. *Формальной грамматикой* называется упорядоченная четвёрка $\Gamma = \langle V, T, P, S\rangle$, состоящая из следующих объектов:

(а) V — конечное множество *нетерминальных (вспомогательных)* символов;

(б) T — конечное множество *терминальных* символов такое, что $V \cap T = \varnothing$;

(в) P — конечное множество *продукций*, т. е. цепочек вида $\alpha \longrightarrow \beta$, где слова $\alpha, \beta \in (V \cup T)^*$, и в α содержится хотя бы один элемент из V;

(г) S — *начальный символ*, $S \in V$.

Определение. Пусть $\Gamma = \langle V, T, P, S\rangle$ — формальная грамматика, $\alpha, \beta \in (V \cup T)^*$. Говорят, что *слово β непосредственно получается из слова α в грамматике* Γ, и пишут $\alpha \underset{\Gamma}{\longrightarrow} \beta$, если существуют слова α_0, α_1, γ, δ такие, что $\alpha = \alpha_0\gamma\alpha_1$, $\beta = \alpha_0\delta\alpha_1$, и продукция $\gamma \longrightarrow \delta$ принадлежит множеству P.

Определение. Говорят, что *слово β выводимо из слова α в грамматике* Γ, и пишут $\alpha \overset{*}{\underset{\Gamma}{\longrightarrow}} \beta$, если существует конечная последовательность слов $\beta_0, \ldots, \beta_k$ такая, что $\beta_0 = \alpha$, $\beta_k = \beta$ и имеет место

$$\beta_0 \underset{\Gamma}{\longrightarrow} \beta_1 \underset{\Gamma}{\longrightarrow} \ldots \underset{\Gamma}{\longrightarrow} \beta_k. \qquad (*)$$

Цепочка $(*)$ называется *выводом слова β из слова α в грамматике* Γ.

Определение. Пусть $\Gamma = \langle V, T, P, S\rangle$ — формальная грамматика. Тогда множество $L(\Gamma) = \{w \in T^* \mid S \overset{*}{\underset{\Gamma}{\longrightarrow}} w\}$ называется *языком, порождённым грамматикой* Γ.

Пример. Вернёмся к примеру с алгоритмом порождения сбалансированных слов. С формальной точки зрения в данной грамматике множество нетерминальных символов имеет вид $V = \{S\}$, множеством терминальных символов является $T = \{(,)\}$. Множество продукций $P = \{S \to (), S \to (S), S \to SS\}$. Начальный символ — это символ S.

Пример. Рассмотрим теперь пример формальной грамматики, которая порождает язык всех десятичных записей натуральных чисел в алфавите из десяти терминальных цифр $T = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Каждая запись натурального числа либо есть 0, либо получается приписыванием ненулевой цифры слева к некоторой последовательности цифр. В свою очередь, каждая последовательность цифр либо есть пустое слово Λ, либо получается приписыванием любой цифры (включая 0) слева к некоторому слову, про которое уже известно, что оно является последовательностью цифр. Эти рассуждения позволяют формально выписать множество продукций P:

$$\begin{array}{ll} S \longrightarrow 0 & A \longrightarrow \Lambda \\ S \longrightarrow 1A & A \longrightarrow 0A \\ S \longrightarrow 2A & A \longrightarrow 1A \\ \ldots & \ldots \\ S \longrightarrow 9A & A \longrightarrow 9A \end{array}$$

Таким образом, алфавит нетерминальных символов $V = \{S, A\}$ состоит из двух букв. Символ S — начальный, он соответствует термину «запись натурального числа». Символ A — термину «последовательность цифр».

Определение. Формальная грамматика $\Gamma = \langle V, T, P, S\rangle$ называется:

(а) *контекстно-свободной*, если все её продукции имеют вид $A \longrightarrow \beta$, где $A \in V$, $\beta \in (V \cup T)^*$;

(б) *регулярной*, если все её продукции имеют вид $A \longrightarrow aB$ или $A \longrightarrow \Lambda$, где $A, B \in V$, $a \in T$.

Замечание. В некоторых источниках регулярными называют формальные грамматики, в которых продукции имеют вид $A \longrightarrow aB$, $A \longrightarrow a$ или $A \longrightarrow \Lambda$, где $A, B \in V$, $a \in T$. Однако такие грамматики легко сводятся к регулярным грамматикам из определения выше. Для этого достаточно каждую продукцию $p = A \longrightarrow a$ заменить на

пару продукций $A \longrightarrow aA_p$ и $A_p \longrightarrow \Lambda$, где A_p — новый нетерминальный символ. Например, рассмотренный выше язык десятичных записей натуральных чисел порождается регулярной грамматикой. Ясно также, что любая регулярная грамматика является контекстно-свободной.

Регулярные грамматики также иногда называют *праволинейными*, поскольку при порождении слов в таких грамматиках каждый новый терминальный символ появляется справа от предыдущего. Любой вывод слова $a_1a_2\ldots a_k \in T^*$ из начального символа S в регулярной грамматике имеет вид

$$S \underset{\Gamma}{\longrightarrow} a_1A_1 \underset{\Gamma}{\longrightarrow} a_1a_2A_2 \underset{\Gamma}{\longrightarrow} \ldots \underset{\Gamma}{\longrightarrow} a_1a_2\ldots a_kA_k \underset{\Gamma}{\longrightarrow} a_1a_2\ldots a_k,$$

где $A_1, \ldots, A_k$ — нетерминальные символы, при этом сначала в выводе многократно используются продукции вида $A \longrightarrow aB$ и лишь в конце применяется продукция вида $A \longrightarrow \Lambda$.

Можно сказать, что в регулярных грамматиках слова порождаются слева направо без возможности возврата в начальные символы слова. Иначе говоря, если на промежуточном шаге было выведено слово $a_1a_2\ldots a_sA_s$, то на всех последующих шагах префикс $a_1a_2\ldots a_s$ останется неизменным. Похожим свойством обладают конечные автоматы, в которых символы слов также прочитываются слева направо. Это наблюдение лежит в основе следующей теоремы.

Теорема 30. *Язык L порождается регулярной грамматикой тогда и только тогда, когда L — автоматный.*

Доказательство. ($\Longrightarrow$) Пусть формальный язык L порождается регулярной грамматикой $\Gamma = \langle V, T, P, S\rangle$. Определим Λ-н.к.а. $\mathfrak{A}$ следующим образом:

(а) Множеством состояний автомата является множество $V \cup \{q\}$, где q — новый символ, т. е. $q \notin V$.

(б) Внешним алфавитом автомата будет множество T.

(в) Начальным состоянием является S.

(г) Единственным выделенным состоянием будет q.

(д) Переходы в автомате $\mathfrak{A}$ определяются по следующему принципу. Для каждой продукции вида $A \longrightarrow aB$ добавляем дугу $\underset{A}{\bigcirc}\xrightarrow{a}\underset{B}{\bigcirc}$. Для каждой продукции $A \longrightarrow \Lambda$ — дугу $\underset{A}{\bigcirc}\xrightarrow{\Lambda}\underset{q}{\circledcirc}$.

Заметим, что по построению автомата в нём нет дуг, выходящих из единственного выделенного состояния q. Отсюда следует, что любой путь в $\mathfrak{A}$, вдоль которого распознаётся некоторое слово $w \in T^*$, должен заканчиваться Λ-переходом, входящим в состояние q, а все остальные переходы данного пути помечены непустыми символами из T.

Покажем, что $L = L(\mathfrak{A})$. Пусть $w = a_1a_2\ldots a_k \in T^*$ — произвольное слово (возможно, пустое). Тогда имеет место следующая цепочка эквивалентных утверждений:

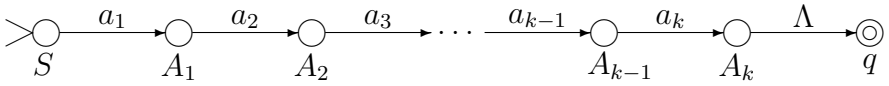
$w \in L \Longleftrightarrow$ Существует вывод

$$S \underset{\Gamma}{\longrightarrow} a_1A_1 \underset{\Gamma}{\longrightarrow} a_1a_2A_2 \underset{\Gamma}{\longrightarrow} \ldots \underset{\Gamma}{\longrightarrow} a_1a_2\ldots a_kA_k \underset{\Gamma}{\longrightarrow} a_1a_2\ldots a_k$$

слова w в грамматике Г. $\Longleftrightarrow$ В множестве P имеются продукции вида

$$\begin{aligned} S &\longrightarrow a_1 A_1 \\ A_1 &\longrightarrow a_2 A_2 \\ \dots \quad &\dots \quad \dots \\ A_{k-1} &\longrightarrow a_k A_k \\ A_k &\longrightarrow \Lambda \end{aligned}$$

$\Longleftrightarrow$ В построенном автомате $\mathfrak{A}$ имеется путь вида



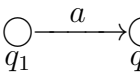
$\Longleftrightarrow w \in L(\mathfrak{A})$. Что и требовалось доказать.

($\Longleftarrow$) Пусть теперь L — автоматный. Тогда существует д.к.а. $\mathfrak{A} = \langle Q, A, \delta, q_0, F\rangle$ такой, что $L = L(\mathfrak{A})$. Определим регулярную грамматику Г следующим образом:

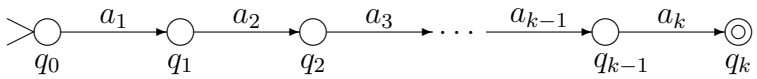
(а) Множеством нетерминальных символов грамматики является множество Q.

(б) Множеством терминальных символов будет множество A.

(в) Начальным символом является q_0.

(г) Множество продукций определяется по следующему принципу. Для каждой дуги автомата вида  добавляем продукцию $q_1 \longrightarrow aq_2$. Кроме этого, для каждого выделенного состояния $q \in F$ добавляем продукцию $q \longrightarrow \Lambda$.

Покажем, что $L = L(\Gamma)$. Пусть $w = a_1 a_2 \dots a_k \in A^*$ — произвольное слово (возможно, пустое). Тогда имеет место следующая цепочка эквивалентных утверждений:
$w \in L \Longleftrightarrow$ В д.к.а. $\mathfrak{A}$ существует путь



вдоль которого распознается слово w. $\Longleftrightarrow$ В построенной грамматике Г имеются продукции

$$\begin{aligned} q_0 &\longrightarrow a_1 q_1 \\ q_1 &\longrightarrow a_2 q_2 \\ \dots \quad &\dots \quad \dots \\ q_{k-2} &\longrightarrow a_{k-1} q_{k-1} \\ q_{k-1} &\longrightarrow a_k q_k \\ q_k &\longrightarrow \Lambda. \end{aligned}$$

$\Longleftrightarrow$ Существует вывод

$$q_0 \underset{\Gamma}{\longrightarrow} a_1 q_1 \underset{\Gamma}{\longrightarrow} a_1 a_2 q_2 \underset{\Gamma}{\longrightarrow} \dots \underset{\Gamma}{\longrightarrow} a_1 \dots a_{k-1} q_{k-1} \underset{\Gamma}{\longrightarrow} a_1 \dots a_{k-1} a_k q_k \underset{\Gamma}{\longrightarrow} a_1 \dots a_{k-1} a_k$$

слова w в грамматике Г. $\Longleftrightarrow w \in L(\Gamma)$. Что и требовалось доказать. $\square$

Замечание. Не всякий язык, порождённый контекстно-свободной грамматикой, можно определить регулярной грамматикой. Так, грамматика, порождающая язык L всех слов со сбалансированными скобками (см. пример выше), контекстно-свободна, но данный язык не задаётся никаким регулярным выражением.

Действительно, если бы L был регулярным, то по лемме о накачивании существовал бы $n \geqslant 1$ такой, что для любого $w \in L$, $|w| \geqslant 1$, имеется представление в виде $w = xyz$, где $|xy| \leqslant n$, $y \neq \Lambda$ и $xy^iz \in L$ для всех $i \in \omega$. Если взять $w = \underbrace{((\ldots(}_{n}\underbrace{)\ldots))}_{n}$, то в соответствующем представлении xyz для слова w подслово y обязано иметь вид $y = \underbrace{((\ldots(}_{m}$, где $1 \leqslant m \leqslant n$. Тогда слово $xz = \underbrace{((\ldots(}_{n-m}\underbrace{)\ldots))}_{n}$ принадлежит L, но, очевидно, не является сбалансированным.

Упражнения

1. Доказать, что слово w в алфавите $\{(,)\}$ имеет сбалансированные скобки тогда и только тогда, когда выполняются следующие два условия:
   (1) Количество открывающих скобок в w равно количеству закрывающих скобок в w.
   (2) В любом префиксе слова w количество открывающих скобок не меньше, чем количество закрывающих скобок.

2. Построить формальную грамматику, порождающую следующий язык:
   (а) $\{a^mb^na^mb^n \mid m, n \in \omega,\ m, n \geqslant 1\}$;
   (б) $\{ww \mid w \in \{a, b\}^*\}$;
   (в) $\{w \in \{a, b, c\}^* \mid w \neq \Lambda$ и число букв a в слове w равно числу букв b, равному числу букв $c\}$.

3. Описать язык, порождаемый грамматикой с продукциями $S \longrightarrow bSS$, $S \longrightarrow a$ над алфавитом $\{a, b\}$.

4. Формальная грамматика $\Gamma = \langle\{S, B, C\}, \{a, b, c\}, P, S\rangle$ задаётся множеством продукций $P = \{S \longrightarrow aSBC, S \longrightarrow abC, CB \longrightarrow BC, bB \longrightarrow bb, bC \longrightarrow bc, cC \longrightarrow cc\}$. Описать язык, порождаемый данной грамматикой.

5. Построить контекстно-свободную грамматику, порождающую язык:
   (а) $\{ww^R \mid w \in \{a, b\}^*\}$;
   (б) $\{w \in \{a, b\}^* \mid w = w^R\}$;
   (в) $\{a^mb^nc^pd^q \mid m, n, p, q \in \omega, m + n = p + q\}$;
   (г) $\{a^mb^n \mid m, n \in \omega, m \leqslant 2n\}$.

6. Контекстно-свободная грамматика $\Gamma = \langle\{S, A, B\}, \{a, b\}, P, S\rangle$ имеет множество продукций $P = \{S \longrightarrow aB, S \longrightarrow bA, A \longrightarrow a, A \longrightarrow aS, A \longrightarrow BAA, B \longrightarrow b, B \longrightarrow bS, B \longrightarrow ABB\}$. Описать язык, порождаемый данной грамматикой.

7. Построить регулярную грамматику, порождающую следующий язык:
   (а) $\{w \in \{0, 1\}^* \mid$ число единиц в слове w делится на три$\}$;
   (б) $\{w \in \{0, 1\}^* \mid w$ содежит чётное число нулей и чётное число единиц$\}$;
   (в) $\{w \in \{a, b\}^* \mid$ слово w не содержит подслов aa и $bb\}$.

8. Назовём Λ-н.к.а. $\mathfrak{A} = \langle Q, A, \Delta, q_0, F\rangle$ *регулярным*, если он удовлетворяет следующим четырём условиям:

   (а) автомат $\mathfrak{A}$ содержит единственное выделенное состояние q', причём $q' \neq q_0$;

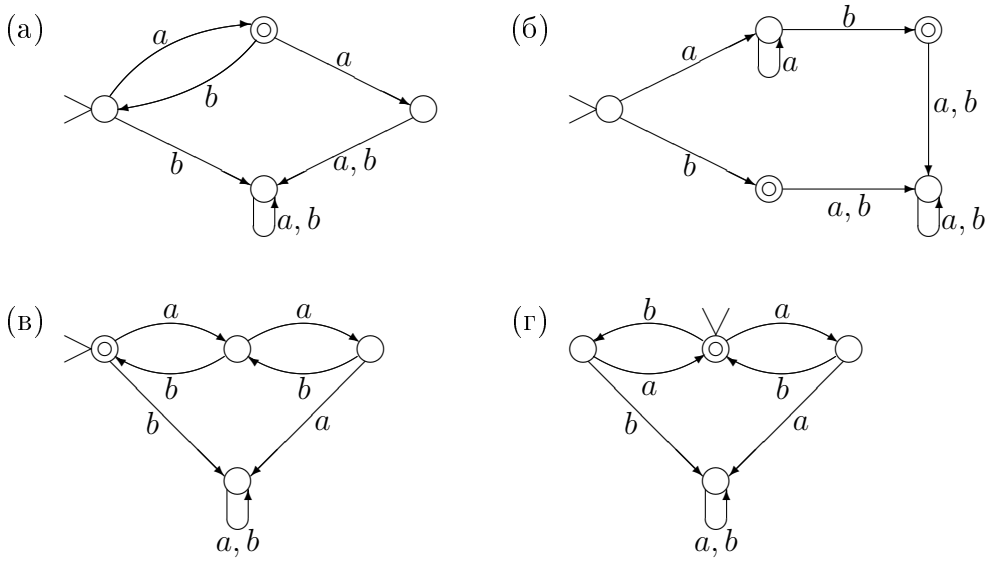
   (б) в автомате $\mathfrak{A}$ нет дуг, выходящих из состояния q';

   (в) в автомате $\mathfrak{A}$ нет дуг, входящих в состояние q' и помеченных непустым символом $a \in A$;

   (г) любой Λ-переход в $\mathfrak{A}$ имеет вид $q \xrightarrow{\Lambda} q'$ для некоторого $q \in Q$, $q \neq q'$.

   Доказать, что произвольный язык L порождается регулярной грамматикой тогда и только тогда, когда L распознается некоторым регулярным Λ-н.к.а. (Указание: см. доказательство теоремы 30.)

9. Используя метод доказательства теоремы 30, построить регулярную грамматику, порождающую язык, распознаваемый следующим детерминированным конечным автоматом:



Глава IV

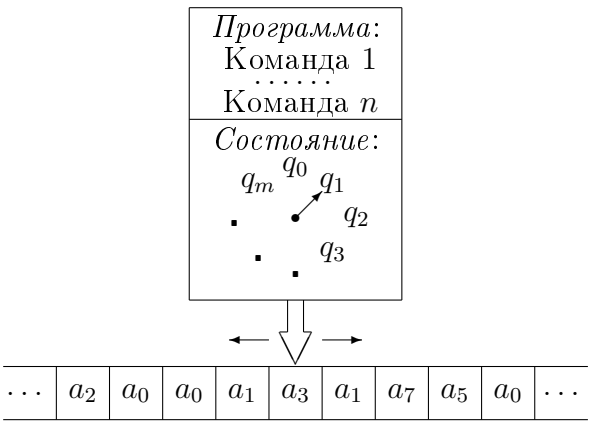
Формализации понятия вычислимой функции

Основной целью данной главы является формализация понятия вычислимой частичной функции, действующей на натуральных числах. Напомним, что *k-местной частичной функцией на ω* мы называем любую функцию вида $f : X \to \omega$, где $X \subseteq \omega^k$, $k \in \omega$. Существует несколько различных подходов, приводящих к тому или иному определению частичной вычислимой функции. Мы рассмотрим два основных подхода и покажем, что они эквивалентны. Первая из формализаций, с которой мы познакомимся в следующем параграфе, связана с машинами Тьюринга [8, 9, 15].

§ 15. Определение машины Тьюринга

Машины Тьюринга — это один из подходов к формализации понятия вычислимой функции. Модель для машины Тьюринга имеет механическое описание, хотя окончательная их формализация является совершенно математической.

Перейдём к неформальному описанию машин Тьюринга.



Машина Тьюринга состоит из *ленты*, разбитой на одинаковые ячейки. В ячейках могут содержаться символы из внешнего алфавита A. Содержимое ячеек может меняться. Лента — это механизм входов и выходов машины. Перед запуском машины входные данные записываются в конечном участке ленты, и в дальнейшем на каждом шаге вычислений используется только конечное множество ячеек. Однако в процессе работы лента может достраиваться новыми ячейками как слева, так и справа. Другими словами, лента является потенциально бесконечной в обе стороны.

Также машина Тьюринга обладает *считывающей головкой*, которая в процессе работы машины может обозревать только одну ячейку и распознавать её содержимое. В зависимости от исполняемой команды головка способна изменять содержимое обозреваемой ячейки и сдвигаться за один шаг на одну ячейку влево или вправо. Таким образом, машина Тьюринга — это устройство с *последовательным* доступом к памяти: чтобы получить доступ к содержащейся в некоторой ячейке информации,

машине приходится перебирать одну за другой все ячейки между текущей и требуемой.

Каждая машина Тьюринга имеет конечное число *состояний* $q_0, q_1, \ldots, q_m$, в которых она может находиться. В процессе работы машина может несколько раз возвращаться в одно и то же состояние. Работа всегда начинается с выделенного *начального состояния*. Кроме этого, есть выделенное *конечное состояние*. Если когда-нибудь машина попадает в конечное состояние, то она останавливается, иначе работает бесконечно.

Любая машина Тьюринга имеет свою уникальную программу. *Программа* — это конечный список команд, каждая из которых есть директива вида $q_i a_j \to q_k a_l S$, где q_i, q_k — состояния, a_j, a_l — символы внешнего алфавита, $S \in \{R, L, \Lambda\}$, причём для каждого неконечного состояния q_i и любого символа a_j в программе имеется ровно одна команда, начинающаяся с символов $q_i a_j \to \ldots$ Если же q_i — конечное состояние, то команда вида $q_i a_j \to \ldots$ отсутствует в программе.

Опишем один шаг работы машины Тьюринга. Пусть машина находится в некотором состоянии q_i, а головка в текущий момент обозревает ячейку с символом a_j. Если q_i — конечное состояние, то машина останавливается. В противном случае в программе находится единственная команда вида $q_i a_j \to q_k a_l S$, которая затем исполняется следующим образом: в обозреваемую ячейку записывается символ a_l, затем головка сдвигается по ленте на одну ячейку вправо (если $S = R$) или влево (если $S = L$) или вообще не двигается (если S — пустое слово), и затем машина переходит в состояние q_k.

Если при исполнении некоторой команды головке необходимо сдвинуться за левый край ленты, то в этот момент происходит автоматическое достраивание новой ячейки слева, в неё записывается выделенный символ (как правило, это символ 0) и головка передвигается на достроенную ячейку. Аналогичным образом происходит достраивание ячеек справа.

Теперь мы введём строгие, формальные определения.

Определение. *Машиной Тьюринга* над алфавитом A называется упорядоченная пятёрка $T = \langle A, Q, P, q_1, q_0 \rangle$, состоящая из следующих объектов:

(а) $A = \{a_0, a_1, \ldots, a_n\}$ — конечный *внешний алфавит* (мы будем всегда предполагать, что $n \geqslant 1$ и $a_0 = 0, a_1 = 1$);

(б) $Q = \{q_0, q_1, \ldots, q_m\}$ — конечный алфавит *внутренних состояний*, $m \geqslant 1$;

(в) $P = \{T(i,j) \mid 1 \leqslant i \leqslant m, 0 \leqslant j \leqslant n\}$ — *программа*, состоящая из команд $T(i,j)$, каждая из которых есть слово вида: $q_i a_j \to q_k a_l$, или $q_i a_j \to q_k a_l R$, или $q_i a_j \to q_k a_l L$, где $0 \leqslant k \leqslant m$, $0 \leqslant l \leqslant n$;

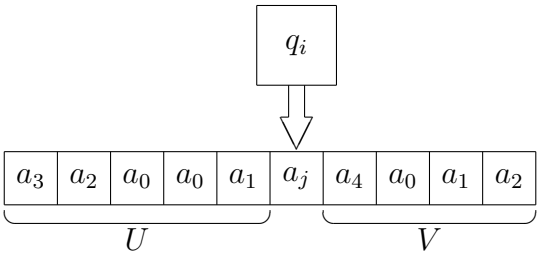
(г) q_1 — *начальное состояние*;

(д) q_0 — *конечное состояние*.

Определение. *Машинным словом (конфигурацией)* называется любое слово вида $U q_i a_j V$, где $q_i \in Q$, $a_j \in A$, U и V — слова в алфавите A (возможно, пустые).

Машинное слово $U q_i a_j V$ — это формальное представление текущего положения всех деталей машины: q_i — текущее состояние, a_j — содержимое ячейки, обозреваемой головкой в данный момент, слово U состоит из всех символов, содержащихся в

ячейках слева от обозреваемой, а слово V — из всех символов, записанных в ячейках справа от обозреваемой. Исходя из такого интуитивного смысла, несложно определить формально, как преобразуются машинные слова за один шаг работы машины.



Определение. Пусть $M = Uq_ia_jV$ — машинное слово. Говорят, что машинное слово M'_T *получается из* M *за один шаг работы машины Тьюринга* T, если выполняется одно из условий:

(1) $i = 0$, $M'_T = M$.

(2) $i > 0$ и выполняется один из случаев:

   (а) $T(i, j)$ имеет вид $q_ia_j \to q_ka_l$, и $M'_T = Uq_ka_lV$;

   (б) $T(i, j)$ имеет вид $q_ia_j \to q_ka_lR$, $V = a_sV'$, и $M'_T = Ua_lq_ka_sV'$;

   (в) $T(i, j)$ имеет вид $q_ia_j \to q_ka_lR$, $V = \Lambda$, и $M'_T = Ua_lq_ka_0$ (достраивается новая ячейка справа);

   (г) $T(i, j)$ имеет вид $q_ia_j \to q_ka_lL$, $U = U'a_s$, и $M'_T = U'q_ka_sa_lV$;

   (д) $T(i, j)$ имеет вид $q_ia_j \to q_ka_lL$, $U = \Lambda$, и $M'_T = q_ka_0a_lV$ (достраивается новая ячейка слева).

Определение. Пусть M — машинное слово. Положим $M_T^{(0)} = M$, $M_T^{(n+1)} = \left(M_T^{(n)}\right)'_T$ для всех $n \in \omega$.

Говорят, что машина T *перерабатывает слово* M *в слово* M_1 *без достраивания ячеек слева*, и пишут $M \underset{T}{\Longrightarrow} M_1$, если существует $n \in \omega$ такое, что $M_T^{(n)} = M_1$ и при этом не используется пункт (д).

Говорят, что машина T *перерабатывает слово* M *в слово* M_1 *без достраивания ячеек слева и справа*, и пишут $M \underset{T}{\Longmapsto} M_1$, если существует $n \in \omega$ такое, что $M_T^{(n)} = M_1$ и при этом не используются пункты (в) и (д).

Определение. Частичная функция $f : \mathrm{dom}(f) \subseteq \omega^k \longrightarrow \omega$ называется *вычислимой по Тьюрингу*, если существует машина Тьюринга T над алфавитом $\{0, 1\}$ такая, что для любых $x_1, \ldots, x_k \in \omega$ выполняются условия:

(а) если $\langle x_1, \ldots, x_k\rangle \in \mathrm{dom}(f)$, то $q_101^{x_1}0 \ldots 01^{x_k}0 \underset{T}{\Longrightarrow} q_001^{f(x_1,\ldots,x_k)}00^s$ для некоторого $s \geqslant 0$;

(б) если $\langle x_1, \ldots, x_k\rangle \notin \mathrm{dom}(f)$, то машина T, начав свою работу с машинного слова $M = q_101^{x_1}0 \ldots 01^{x_k}0$, работает бесконечно, т.е. q_0 не входит в $M_T^{(n)}$ ни для какого $n \in \omega$.

При этом говорят, что машина T *вычисляет* частичную функцию f.

Определение. Если P — программа, то через $\big(P\big)_{q_j}^{q_i}$ будем обозначать множество команд, полученных из P заменой всех вхождений q_i на q_j.

Определение. Пусть $T_1 = \langle A, Q_1, P_1, q_1, q_0\rangle$, $T_2 = \langle A, Q_2, P_2, q_1, q_0\rangle$ — машины Тьюринга над алфавитом A с множествами внутренних состояний

$$Q_1 = \{q_0, q_1, \ldots, q_r\}, \quad Q_2 = \{q_0, q_1, \ldots, q_s\}.$$

Композицией машин T_1 *и* T_2 называется машина $T_1 \circ T_2 = \langle A, Q, P, q_1, q_0\rangle$ с множеством состояний $Q = \{q_0, q_1, \ldots, q_{r+s}\}$ и программой $P = \big(P_1\big)_{q_{r+1}}^{q_0} \cup \big(P_2\big)_{q_{r+1},\ldots,q_{r+s}}^{q_1,\ldots,q_s}$.

Композиция $T_1 \circ T_2$ работает следующим образом. На входных данных сначала запускается машина T_1, а затем на выходных данных T_1 — машина T_2. Выходные данные T_2 считаются выходом для композиции $T_1 \circ T_2$. Конечное состояние машины T_1 отождествляется с начальным состоянием машины T_2.

Определение. Пусть $T_1{=}\langle A, Q_1, P_1, q_1, q_0\rangle$, $T_2{=}\langle A, Q_2, P_2, q_1, q_0\rangle$, $T_3{=}\langle A, Q_3, P_3, q_1, q_0\rangle$ — машины Тьюринга над алфавитом A с множествами внутренних состояний

$$Q_1 = \{q_0, q_1, \ldots, q_r\}, \quad Q_2 = \{q_0, q_1, \ldots, q_s\}, \quad Q_3 = \{q_0, q_1, \ldots, q_t\}.$$

Разветвлением машины T_1 *на* (T_2, T_3) *по* (q_i, q_j), где $q_i, q_j \in Q_1$, называется машина $T_1[q_i \to T_2|q_j \to T_3] = \langle A, Q, P, q_1, q_0\rangle$, где $Q = \{q_0, q_1, \ldots, q_{r+s+t-2}\}$, а программа P получается следующим образом: из P_1 исключаются все команды вида $q_i a_k \to \ldots$ и вида $q_j a_k \to \ldots$; полученное множество обозначим через P_1'. Тогда

$$P = P_1' \cup \big(P_2\big)_{q_i, q_{r+1},\ldots,q_{r+s-1}}^{q_1,q_2,\ldots,q_s} \cup \big(P_3\big)_{q_j, q_{r+s},\ldots,q_{r+s+t-2}}^{q_1,q_2,\ldots,q_t}.$$

Разветвление работает следующим образом. На входных данных запускается машина T_1. Если T_1 попадает в состояние q_0, то происходит остановка. Если T_1 попадает в состояние q_i, то на текущих данных запускается машина T_2, при этом начальное состояние машины T_2 заменяется на q_i. Если T_1 попадает в состояние q_j, то на текущих данных запускается машина T_3, при этом начальное состояние машины T_3 заменяется на q_j. Конечные состояния машин T_1, T_2 и T_3 отождествляются.

Замечание. Существует несколько модернизаций и обобщений машин Тьюринга. Наиболее известной и естественной модернизацией является многоленточная машина Тьюринга, т.е. машина, которая имеет k ($k \geqslant 1$) лент и такое же число считывающих головок. Однако вычислительные возможности таких машин не отличаются от возможностей обычных машин Тьюринга.

Другое обобщение — недетерминированные машины Тьюринга, т. е. машины, в программе которых для некоторых состояний q_i и некоторых внешних символов a_j могут присутствовать несколько различных команд, начинающихся с символов $q_i a_j \to \ldots$ Машины такого типа используюся в качестве традиционной модели вычислений в теории сложности алгоритмов [5, 6, 15].

§ 16. Базовые машины Тьюринга

Для доказательства основных результатов данной главы нам потребуются следующий набор так называемых *базовых* машин Тьюринга. Внешний алфавит всех машин в данном параграфе состоит из двух символов: 0 и 1.

Заметим, что если машина Тьюринга в некотором состоянии q_i никогда не обозревает символ a_j, то команду вида $q_i a_j \to \ldots$ можно не писать в программе. Ниже мы будем использовать данное соглашение для упрощения записи программ.

Перенос нуля А: $q_1 001^x 0 \underset{\text{А}}{\Longmapsto} q_0 01^x 00$, *где* $x \geqslant 0$.

Машина А имеет следующую программу:

$$\begin{array}{ll} q_1 0 \to q_2 0R & q_4 1 \to q_5 0L \\ q_2 0 \to q_3 1R & q_5 1 \to q_5 1L \\ q_3 1 \to q_3 1R & q_5 0 \to q_0 0 \\ q_3 0 \to q_4 0L & \end{array}$$

Машина А работает следующим образом. Сначала второй 0 заменяется на 1 (команды $q_1 0 \to q_2 0R$ и $q_2 0 \to q_3 1R$), затем машина сдвигается на 0, стоящий после массива 1^x (команда $q_3 1 \to q_3 1R$), и заменяет последний символ 1 из этого массива на 0 (команды $q_3 0 \to q_4 0L$ и $q_4 1 \to q_5 0L$), после чего машина возвращается влево на самое начало (команды $q_5 1 \to q_5 1L$ и $q_5 0 \to q_0 0$). Заметим, что новые ячейки справа и слева не достраиваются.

Обнуление О: $q_1 01^x 0 \underset{\text{О}}{\Longmapsto} q_0 00^x 0$, *где* $x \geqslant 0$.

Машина О имеет следующую программу:

$$\begin{array}{ll} q_1 0 \to q_2 0R & q_3 1 \to q_3 0L \\ q_2 1 \to q_2 1R & q_3 0 \to q_0 0 \\ q_2 0 \to q_3 0L & \end{array}$$

Правый сдвиг Б$^+$: $q_1 01^x 0 \underset{\text{Б}^+}{\Longmapsto} 01^x q_0 0$, *где* $x \geqslant 0$.

Программа для машины Б$^+$ состоит из трёх команд:

$$\begin{array}{l} q_1 0 \to q_2 0R \\ q_2 1 \to q_2 1R \\ q_2 0 \to q_0 0 \end{array}$$

Левый сдвиг Б$^-$: $01^x q_1 0 \underset{\text{Б}^-}{\Longmapsto} q_0 01^x 0$, *где* $x \geqslant 0$.

Программа для машины Б$^-$ состоит из трёх команд:

$$\begin{array}{l} q_1 0 \to q_2 0L \\ q_2 1 \to q_2 1L \\ q_2 0 \to q_0 0 \end{array}$$

Транспозиция В: $01^x q_1 01^y 0 \underset{\text{В}}{\Longmapsto} 01^y q_0 01^x 0$, *где* $x, y \geqslant 0$.

Машина В работает следующим образом. Сначала преобразуем слово $01^x q_1 01^y 0$ в слово $01^x q_5 1^y 00$, равное слову $01^{x-i} q_5 1^y 01^i 0$ при $i = 0$. Затем для произвольного i слово $01^{x-i} q_5 1^y 01^i 0$ преобразуем в слово $01^{x-(i+1)} q_5 1^y 01^{i+1} 0$, если $x - i > 0$, и в слово $01^y q_0 01^x 0$, если $x - i = 0$.

Программа для В имеет следующий вид:

$q_10 \to q_20R$	
$q_21 \to q_21R$	
$q_20 \to q_30L$	Преобразуем входное машинное
$q_31 \to q_40L$	слово следующим образом:
$q_30 \to q_50$	$01^xq_101^y0 \Longmapsto 01^xq_51^y00$.
$q_41 \to q_41L$	
$q_40 \to q_51$	
	В состоянии q_5, в котором начинается цикл,
$q_51 \to q_61L$	сдвигаемся на одну ячейку влево и проверяем,
$q_50 \to q_60L$	есть ли ещё единицы в массиве 1^{x-i},
	т.е. верно ли, что $x - i = 0$.
$q_60 \to q_70R$	
$q_71 \to q_71R$	Если $x - i = 0$, то выходим из цикла:
$q_70 \to q_00$	$q_601^y01^x0 \Longmapsto 01^yq_001^x0$.
$q_61 \to q_80R$	
$q_81 \to q_81R$	Если $x - i \geqslant 1$ и $y \geqslant 1$, то преобразуем
$q_80 \to q_91L$	$01^{x-(i+1)}q_611^y01^i0 \Longmapsto 01^{x-(i+1)}q_51^y01^{i+1}0$.
$q_91 \to q_{10}0L$	Если $x - i \geqslant 1$ и $y = 0$, то преобразуем
$q_90 \to q_50$	$01^{x-(i+1)}q_6101^i0 \Longmapsto 01^{x-(i+1)}q_501^{i+1}0$.
$q_{10}1 \to q_{10}1L$	После этого переходим к следующей
$q_{10}0 \to q_51$	итерации в цикле.

Удвоение Г: $q_101^x0 \underset{\Gamma}{\Longrightarrow} q_001^x01^x0$, *где* $x \geqslant 0$.

Машина Г работает следующим образом. Сначала преобразуем слово q_101^x0 в слово $0q_21^x0$, которое совпадает со словом $01^iq_21^{x-i}01^i$ при $i = 0$. Затем для произвольного i преобразуем слово $01^iq_21^{x-i}01^i$ в слово $01^{i+1}q_21^{x-(i+1)}01^{i+1}$, если $x - i > 0$, и в слово $q_001^x01^x0$, если $x - i = 0$.

Программа для Г имеет следующий вид:

$q_10 \to q_20R$	
$q_21 \to q_30R$	
$q_31 \to q_31R$	
$q_30 \to q_40R$	Если $x - i > 0$, то $01^iq_21^{x-i}01^i \Longrightarrow$
$q_41 \to q_41R$	$\Longrightarrow 01^i0q_31^{x-(i+1)}01^i \Longrightarrow 01^i01^{x-(i+1)}01^iq_40 \Longrightarrow$
$q_40 \to q_51L$	$\Longrightarrow 01^iq_601^{x-(i+1)}01^{i+1} \Longrightarrow 01^{i+1}q_21^{x-(i+1)}01^{i+1}$,
$q_51 \to q_51L$	и переходим к следующей итерации.
$q_50 \to q_60L$	
$q_61 \to q_61L$	
$q_60 \to q_21R$	
$q_20 \to q_70R$	
$q_71 \to q_71R$	
$q_70 \to q_80L$	Если $x - i = 0$, то $01^xq_201^x \Longrightarrow$
$q_81 \to q_81L$	$\Longrightarrow 01^x01^xq_70 \Longrightarrow q_001^x01^x0$.
$q_80 \to q_90L$	
$q_91 \to q_91L$	
$q_90 \to q_00$	

Циклический сдвиг $\mathbf{Ц_n}$: $q_101^{x_1}01^{x_2}0\ldots01^{x_n}0 \underset{Ц_n}{\Longmapsto} q_001^{x_2}0\ldots01^{x_n}01^{x_1}0$, *где* $n \geqslant 1$ — *фиксированное натуральное число.*

Машина $Ц_n$ работает по следующей схеме с использованием базовых машин $Б^+$, $Б^-$ и В:

$$q_1 01^{x_1} 01^{x_2} 0 \ldots 01^{x_n} 0 \models\!\Longrightarrow 01^{x_2} q_{\alpha_1} 01^{x_1} 01^{x_3} 0 \ldots 01^{x_n} 0 \models\!\Longrightarrow$$
$$\models\!\Longrightarrow 01^{x_2} 01^{x_3} q_{\alpha_2} 01^{x_1} 01^{x_4} 0 \ldots 01^{x_n} 0 \models\!\Longrightarrow \ldots \models\!\Longrightarrow 01^{x_2} \ldots 01^{x_n} q_{\alpha_{n-1}} 01^{x_1} 0 \models\!\Longrightarrow$$
$$\models\!\Longrightarrow q_0 01^{x_2} 0 \ldots 01^{x_n} 01^{x_1} 0.$$

Таким образом, $Ц_n = \underbrace{(Б^+ \circ В) \circ \ldots \circ (Б^+ \circ В)}_{n-1} \circ \underbrace{Б^- \circ \ldots \circ Б^-}_{n-1} = (Б^+ \circ В)^{n-1} \circ (Б^-)^{n-1}$.

Копирование $К_n$: $q_1 01^{x_1} 0 \ldots 01^{x_n} 0 \underset{К_n}{\Longrightarrow} q_0 01^{x_1} 0 \ldots 01^{x_n} 01^{x_1} 0 \ldots 01^{x_n} 0$, *где* $n \geqslant 1$ — *фиксированное натуральное число.*

Построим машину $К_n$ индукцией по $n \geqslant 1$.

1^0. При $n = 1$ машина $К_1$ совпадает с машиной Г.

2^0. Пусть $n > 1$ и машина $К_{n-1}$ уже определена. Тогда машина $К_n$ работает по следующей схеме:

$$q_1 01^{x_1} 0 \ldots 01^{x_n} 0 \Longrightarrow 01^{x_1} 0 \ldots 01^{x_{n-1}} q_{\alpha_1} 01^{x_n} 0 \Longrightarrow 01^{x_1} 0 \ldots 01^{x_{n-1}} q_{\alpha_2} 01^{x_n} 01^{x_n} 0 \Longrightarrow$$
$$\Longrightarrow q_{\alpha_3} 01^{x_1} 0 \ldots 01^{x_{n-1}} 01^{x_n} 01^{x_n} 0 \Longrightarrow q_{\alpha_4} 01^{x_n} 01^{x_n} 01^{x_1} 0 \ldots 01^{x_{n-1}} 0 \Longrightarrow$$
$$\Longrightarrow 01^{x_n} 01^{x_n} q_{\alpha_5} 01^{x_1} 0 \ldots 01^{x_{n-1}} 0 \underset{К_{n-1}}{\Longrightarrow} 01^{x_n} 01^{x_n} q_{\alpha_6} 01^{x_1} 0 \ldots 01^{x_{n-1}} 01^{x_1} 0 \ldots 01^{x_{n-1}} 0 \Longrightarrow$$
$$\Longrightarrow q_{\alpha_7} 01^{x_n} 01^{x_n} 01^{x_1} 0 \ldots 01^{x_{n-1}} 01^{x_1} 0 \ldots 01^{x_{n-1}} 0 \Longrightarrow$$
$$\Longrightarrow q_{\alpha_8} 01^{x_n} 01^{x_1} 0 \ldots 01^{x_{n-1}} 01^{x_1} 0 \ldots 01^{x_{n-1}} 01^{x_n} 0 \Longrightarrow q_0 01^{x_1} 0 \ldots 01^{x_n} 01^{x_1} 0 \ldots 01^{x_n} 0.$$

Таким образом, $К_n = (Б^+)^{n-1} \circ Г \circ (Б^-)^{n-1} \circ (Ц_{n+1})^{n-1} \circ (Б^+)^2 \circ К_{n-1} \circ (Б^-)^2 \circ Ц_{2n} \circ Ц_n$.

Упражнения

1. Построить машину Тьюринга T такую, что $q_1 01^{2x} 0 \underset{T}{\models\!\Longrightarrow} q_0 0(10)^x 0$, где $x \geqslant 0$.
2. Построить машину Тьюринга T такую, что $q_1 01^{3x} 0 \underset{T}{\models\!\Longrightarrow} 01^x q_0 1^{2x} 0$, где $x \geqslant 0$.
3. Построить машину Тьюринга T такую, что $q_1 01^{3x} 0 \underset{T}{\models\!\Longrightarrow} q_0 01^x 0^x 1^x 0$, где $x \geqslant 0$.
4. Построить машину Тьюринга T_1 такую, что для $x \in \omega$ справедливо
$$01^x q_1 0 \underset{T_1}{\models\!\Longrightarrow} \begin{cases} 01^x q_3 0, & \text{если } x > 0, \\ 01^x q_4 0, & \text{если } x = 0. \end{cases}$$
Используя машину T_1, базовые машины, а также композицию и разветвление машин, определить машину Тьюринга T_2 такую, что для $x, y, z \in \omega$ справедливо
$$q_1 01^x 01^y 01^z 0 \underset{T_2}{\models\!\Longrightarrow} \begin{cases} q_0 01^y 0 \ldots 0, & \text{если } x > 0, \\ q_0 01^z 0 \ldots 0, & \text{если } x = 0. \end{cases}$$
5. Построить машину Тьюринга, вычисляющую следующую частичную функцию:
(а) $f(x, y) = x + y$;
(б) $f(x, y) = x - y$ (при $x < y$ значение функции не определено);
(в) $f(x) = x/2$ (при нечётном x значение функции не определено);
(г) $f(x) = [x/2]$;
(д) $f(x, y) = \begin{cases} x + y - 2, & \text{если } x \geqslant 1 \text{ и } y \geqslant 1, \\ \text{не определено}, & \text{иначе.} \end{cases}$

§ 17. Частично рекурсивные функции

В этом параграфе мы рассмотрим другой подход к формализации понятия вычислимой функции. Его можно назвать алгебраическим, поскольку определяемый здесь класс функций будет порождаться из некоторых простейших функций с помощью определённых операций. Совокупность функций, возникающих в результате подобного алгебраического порождения, называется классом частично рекурсивных функций.

В рамках данного параграфа рассматриваются только частичные функции на ω, т. е. всевозможные функции вида $f : X \to \omega$, где множество $X \subseteq \omega^k$ является областью определения функции, а число $k \in \omega$ — её местностью. Любую 0-местную всюду определённую функцию $f = a$ мы отождествляем с константой $a \in \omega$. Нигде не определённая функция единственна и имеет вид $f = \varnothing$, причем любое натуральное число является местностью нигде не определённой функции.

Если f — n-местная, а g — m-местная частичная функция, то для любых значений $x_1, \ldots, x_n, y_1, \ldots, y_m \in \omega$ мы пишем $f(x_1, \ldots, x_n) = g(y_1, \ldots, y_m)$ тогда и только тогда, когда либо эти значения одновременно не определены, либо они оба определены и совпадают.

Определение. *Простейшими* функциями называются нульместная функция 0, всюду определённая одноместная функция $s(x) = x + 1$ и всюду определённые n-местные функции $I^n_m(x_1, \ldots, x_n) = x_m$ для всех m, n таких, что $1 \leqslant m \leqslant n$.

Определение. Говорят, что функция $f(x_1, \ldots, x_n)$ получается с помощью *оператора суперпозиции* S из функций $h(y_1, \ldots, y_m), g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n)$, если для любых $x_1, \ldots, x_n$ выполняется

$$f(x_1, \ldots, x_n) = h(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n)).$$

Замечание. С неформальной точки зрения оператор суперпозиции соответствует принципу последовательного запуска вычислительных устройств, когда результаты вычислений одних устройств служат входными данными для других. Если f получена с помощью суперпозиции из $h, g_1, \ldots, g_m$, то вычисление значения $f(x_1, \ldots, x_n)$ происходит следующим образом. Сначала на входных данных $\overline{x} = \langle x_1, \ldots, x_n \rangle$ вычисляются значения $y_1 = g_1(\overline{x}), \ldots, y_m = g_m(\overline{x})$. Если хотя бы одно из этих значений не определено, то значение $f(\overline{x})$ тоже не определено. Если же все $y_1, \ldots, y_m$ определены, то затем они подаются на вход функции h. Если выходное значение $h(y_1, \ldots, y_m)$ не определено, то $f(\overline{x})$ считается неопределённым, иначе $f(\overline{x}) = h(y_1, \ldots, y_m)$ определено и является окончательным выходным значением.

Пример. Функция $f(x, y, z) = z + 1$ получается с помощью оператора суперпозиции из простейших функций $s(x)$ и $I^3_3(x, y, z)$, поскольку $z + 1 = s(I^3_3(x, y, z))$.

Определение. Говорят, что функция $f(x_1, \ldots, x_n, y)$ получается с помощью *оператора примитивной рекурсии* R из функций $g(x_1, \ldots, x_n)$ и $h(x_1, \ldots, x_n, y, z)$, если для любых $x_1, \ldots, x_n, y$ выполняется схема примитивной рекурсии:

$$\begin{cases} f(x_1, \ldots, x_n, 0) = g(x_1, \ldots, x_n) \\ f(x_1, \ldots, x_n, y + 1) = h(x_1, \ldots, x_n, y, f(x_1, \ldots, x_n, y)). \end{cases}$$

Замечание. Оператор примитивной рекурсии формализует циклическую структуру, вычисляющую функции, заданные рекуррентными соотношениями (или по индукции). Если f получена с помощью примитивной рекурсии из g и h, то вычисление значения $f(\overline{x}, y)$ происходит следующим образом. Сначала на входных данных $\overline{x}$ вычисляется значение $g(\overline{x})$, которое совпадает с $f(\overline{x}, 0)$. Если это значение не определено, то $f(\overline{x}, y)$ тоже не определено, иначе, подав $\overline{x}$, 0 и $f(\overline{x}, 0)$ на вход функции h, мы вычисляем $h(\overline{x}, 0, f(\overline{x}, 0))$, которое совпадает с $f(\overline{x}, 1)$. Если последнее значение не определено, то $f(\overline{x}, y)$ не определено, иначе мы подаем $\overline{x}$, 1 и $f(\overline{x}, 1)$ на вход функции h и т.д. Данные вычисления продолжаются до тех пор, пока мы не достигнем значения $f(\overline{x}, y)$. Если при этом на промежуточных шагах хотя бы одно вычисляемое значение функции h не определено, то $f(\overline{x}, y)$ считается неопределённым.

Пример. Запишем схему примитивной рекурсии для функции $f(x, y) = x + y$:

$$\begin{cases} f(x, 0) = x, \\ f(x, y + 1) = (x + y) + 1 = s(f(x, y)) = s(I_3^3(x, y, f(x, y))). \end{cases}$$

Таким образом, функция $f(x, y) = x + y$ получается с помощью оператора примитивной рекурсии из функций $g(x) = x$ и $h(x, y, z) = s(I_3^3(x, y, z))$.

Пример. Определим одноместную функцию *усечённой разности*:

$$x \dot{-} 1 = \begin{cases} 0, & \text{если } x = 0; \\ x - 1, & \text{если } x > 0. \end{cases}$$

Запишем для функции $f(x) = x \dot{-} 1$ схему примитивной рекурсии:

$$\begin{cases} f(0) = 0, \\ f(x + 1) = (x + 1) - 1 = x = I_1^2(x, f(x)). \end{cases}$$

Таким образом, функция $f(x) = x \dot{-} 1$ получается с помощью оператора примитивной рекурсии из константы 0 и функции $h(x, y) = I_1^2(x, y)$.

Определение. Говорят, что функция $f(x_1, \ldots, x_n)$ получается с помощью *оператора минимизации* M из функции $g(x_1, \ldots, x_n, y)$ и обозначается $f(x_1, \ldots, x_n) = \mu y[g(x_1, \ldots, x_n, y) = 0]$, если для любых $x_1, \ldots, x_n$ выполняется

$$f(x_1, \ldots, x_n) = \begin{cases} y, & \text{если } g(x_1, \ldots, x_n, 0), \ldots, g(x_1, \ldots, x_n, y - 1) \\ & \text{определены и не равны 0, а значение} \\ & g(x_1, \ldots, x_n, y) \text{ определено и равно 0,} \\ \text{не определено} & \text{в противном случае.} \end{cases}$$

Замечание. Оператор минимизации последовательно перебирает числа из натурального ряда и проверяет, удовлетворяет ли текущее число фиксированному эффективному условию. Эффективное условие записывается в виде уравнения $g(\overline{x}, y) = 0$. Вычисление значения $f(\overline{x})$ происходит следующим образом. Сначала на входных данных $\overline{x}, 0$ вычисляется значение $g(\overline{x}, 0)$. Если это значение не определено, то $f(\overline{x})$ тоже не определено. Иначе если $g(\overline{x}, 0) = 0$, то вычисления заканчиваются — число 0 будет подано на выход. Если же $g(\overline{x}, 0) \neq 0$, то на входных данных $\overline{x}, 1$ вычисляется

значение $g(\overline{x}, 1)$. Если это значение не определено, то $f(\overline{x})$ тоже не определено. Иначе если $g(\overline{x}, 1) = 0$, то вычисления заканчиваются — число 1 будет подано на выход. Если же $g(\overline{x}, 1) \neq 0$, то данный процесс продолжается. В результате мы либо вычислим наименьшее решение уравнения $g(\overline{x}, y) = 0$, либо ничего не вычислим. Последнее возможно по двум причинам: либо некоторое вычисляемое значение функции g окажется неопределённым, либо $g(\overline{x}, y)$ определено для всех y, но не равно нулю.

Пример. Применим оператор минимизации к функции $g(x, y) = (x+y)\dot{-}1$ — получим функцию $f(x) = \mu y[(x + y)\dot{-}1 = 0]$.

Если $x = 0$, то $f(0) = \mu y[y\dot{-}1{=}0] = 0$. Если $x = 1$, то $f(1) = \mu y[(1 + y)\dot{-}1{=}0] = = \mu y[y{=}0] = 0$. Если же $x \geqslant 2$, то $(x + y)\dot{-}1 > 0$ при любом значении y, и значит, минимизация в этом случае не определена.

Таким образом, функция $f(x)$ имеет следующее кусочное представление:

$$f(x) = \begin{cases} 0, & \text{если } x \leqslant 1; \\ \text{не определено}, & \text{если } x \geqslant 2. \end{cases}$$

Определение. Определим семейство *примитивно рекурсивных функций* (сокращённо п.р.ф.) по индукции:

(1) любая простейшая функция — п.р.ф.;

(2) если $h, g_1, \ldots, g_m$ — п.р.ф., а f получена из $h, g_1, \ldots, g_m$ с помощью оператора суперпозиции, то f тоже п.р.ф.;

(3) если g, h — п.р.ф., а f получена из g, h с помощью оператора примитивной рекурсии, то f тоже п.р.ф.

Любая п.р.ф. получается конечным числом применений пунктов (1)–(3).

Определение. Определим семейство *частично рекурсивных функций* (сокращённо ч.р.ф.) по индукции:

(1) любая простейшая функция — ч.р.ф.;

(2) если $h, g_1, \ldots, g_m$ — ч.р.ф., а f получена из $h, g_1, \ldots, g_m$ с помощью оператора суперпозиции, то f тоже ч.р.ф.;

(3) если g, h — ч.р.ф., а f получена из g, h с помощью оператора примитивной рекурсии, то f тоже ч.р.ф.;

(4) если g — ч.р.ф., а f получена из g с помощью оператора минимизации, то f тоже ч.р.ф.

Любая ч.р.ф. получается конечным числом применений пунктов (1)–(4).

Определение. Всюду определённые частично рекурсивные функции называются *рекурсивными* (сокращённо р.ф.).

Замечание. Таким образом, частичная функция f является частично рекурсивной (примитивно рекурсивной), если существует конечная последовательность функций $f_0, \ldots, f_m$ такая, что $f_m = f$ и для любого $i \leqslant m$ функция f_i либо простейшая, либо

получается из некоторых предыдущих f_j, $j < i$, с помощью одного из операторов S, R или M (операторов S или R).

Очевидно, между введёнными классами функций имеют место следующие теоретико-множественные включения:

$$\text{ПРФ} \subseteq \text{РФ} \subseteq \text{ЧРФ}.$$

Пример. Рассмотрим последовательность частичных функций:
$f_0(x) = x = I_1^1(x)$ — простейшая функция;
$f_1(x) = s(x)$ — простейшая функция;
$f_2(x, y, z) = I_3^3(x, y, z)$ — простейшая функция;
$f_3(x, y, z) = z + 1$ — функция, полученная с помощью оператора S из f_1 и f_2;
$f_4(x, y) = x + y$ — функция, полученная с помощью оператора R из f_0 и f_3;
$f_5 = 0$ — простейшая функция;
$f_6(x, y) = I_1^2(x, y)$ — простейшая функция;
$f_7(x) = x \dot{-} 1$ — функция, полученная с помощью оператора R из f_5 и f_6;
$f_8(x, y) = (x + y) \dot{-} 1$ — функция, полученная с помощью оператора S из f_7 и f_4;

$$f_9(x) = \begin{cases} 0, & \text{если } x \leqslant 1; \\ \text{не определено}, & \text{если } x \geqslant 2. \end{cases}$$

Функция f_9 получена с помощью оператора M из f_8. Значит, f_9 — ч.р.ф. Более того, все функции $f_0, \ldots, f_9$ частично рекурсивны, а функции $f_0, \ldots, f_8$ даже примитивно рекурсивны.

Замечание. Часто в число простейших функций включают также одноместную всюду определённую функцию $o(x) = 0$. Заметим, что класс частично рекурсивных функций при этом не изменяется. Это следует из того, что $o(x)$ можно получить из 0-местной функции 0 и 2-местной функции $I_2^2(x, y)$ с помощью оператора примитивной рекурсии.

Предложение 31. *Любая простейшая функция вычислима по Тьюрингу.*

Доказательство. Нульместная константа 0 вычисляется программой, состоящей из одной команды $q_1 0 \to q_0 0$.

Одноместная функция $s(x)$ вычисляется с помощью следующей программы:

$$\begin{array}{ll} q_1 0 \to q_2 0 R & q_3 0 \to q_4 0 L \\ q_2 1 \to q_2 1 R & q_4 1 \to q_4 1 L \\ q_2 0 \to q_3 1 R & q_4 0 \to q_0 0. \end{array}$$

Машина, вычисляющая функцию $I_m^n(x_1, \ldots, x_n)$, работает по следующей схеме:

$$q_1 01^{x_1} 0 \ldots 01^{x_m} 0 \ldots 01^{x_n} 0 \underset{\text{Ц}_n^{m-1}}{\Longrightarrow} q_\alpha 01^{x_m} 0 \ldots 01^{x_n} 01^{x_1} 0 \ldots 01^{x_{m-1}} 0 \underset{(\text{Б}^+)^{n-1}}{\Longrightarrow}$$
$$\underset{(\text{Б}^+)^{n-1}}{\Longrightarrow} 01^{x_m} 0 \ldots 01^{x_n} 01^{x_1} 0 \ldots q_\beta 01^{x_{m-1}} 0 \underset{(\text{О}\cdot\text{Б}^-)^{n-1}}{\Longrightarrow} q_0 01^{x_m} 0 \ldots 0.$$

Таким образом, функция I_m^n вычисляется с помощью следующей композиции базовых машин:

$$(\text{Ц}_n)^{m-1} \circ (\text{Б}^+)^{n-1} \circ (\text{О} \circ \text{Б}^-)^{n-1}.$$

□

Предложение 32. *Семейство вычислимых по Тьюрингу функций замкнуто относительно оператора суперпозиции.*

Доказательство. Пусть функция $f(x_1,\ldots,x_n)$ получена с помощью оператора суперпозиции из частичных функций $h(y_1,\ldots,y_m)$, $g_1(x_1,\ldots,x_n),\ldots,g_m(x_1,\ldots,x_n)$ и пусть функции $h, g_1,\ldots,g_m$ вычислимы на машинах Тьюринга H, $G_1,\ldots,G_m$ соответственно.

Тогда машина, вычисляющая функцию $f(\overline{x}) = h(g_1(\overline{x}),\ldots,g_m(\overline{x}))$, работает по следующей схеме:

$$q_1 01^{x_1}0\ldots01^{x_n}0 \underset{\text{К}_n}{\Longrightarrow} q_{\alpha_1}01^{x_1}0\ldots01^{x_n}01^{x_1}0\ldots01^{x_n}0 \underset{(\text{Б}^+)^n}{\Longrightarrow}$$
$$\underset{(\text{Б}^+)^n}{\Longrightarrow} 01^{x_1}0\ldots01^{x_n}q_{\beta_1}01^{x_1}0\ldots01^{x_n}0 \underset{G_1}{\Longrightarrow} 01^{x_1}0\ldots01^{x_n}q_{\gamma_1}01^{g_1(\overline{x})}0\ldots0 \underset{(\text{Б}^-)^n}{\Longrightarrow}$$
$$\underset{(\text{Б}^-)^n}{\Longrightarrow} q_{\delta_1}01^{x_1}0\ldots01^{x_n}01^{g_1(\overline{x})}0\ldots0 \underset{(\text{Ц}_{n+1})^n}{\Longrightarrow} q_{\varepsilon_1}01^{g_1(\overline{x})}01^{x_1}0\ldots01^{x_n}0\ldots0 \underset{\text{Б}^+}{\Longrightarrow}$$
$$\underset{\text{Б}^+}{\Longrightarrow} 01^{g_1(\overline{x})}q_{\zeta_1}01^{x_1}0\ldots01^{x_n}0\ldots0 \underset{\text{К}_n}{\Longrightarrow}\ldots$$
$$\ldots\underset{\text{Б}^+}{\Longrightarrow} 01^{g_1(\overline{x})}0\ldots01^{g_{m-1}(\overline{x})}q_{\zeta_{m-1}}01^{x_1}0\ldots01^{x_n}0\ldots0 \underset{G_m}{\Longrightarrow}$$
$$\underset{G_m}{\Longrightarrow} 01^{g_1(\overline{x})}0\ldots01^{g_{m-1}(\overline{x})}q_{\gamma_m}01^{g_m(\overline{x})}0\ldots0 \underset{(\text{Б}^-)^{m-1}}{\Longrightarrow}$$
$$\underset{(\text{Б}^-)^{m-1}}{\Longrightarrow} q_{\delta_m}01^{g_1(\overline{x})}0\ldots01^{g_m(\overline{x})}0\ldots0 \underset{H}{\Longrightarrow} q_0 01^{f(\overline{x})}0\ldots0.$$

Таким образом, следующая машина Тьюринга вычисляет функцию f:

$$\big(\text{К}_n\circ(\text{Б}^+)^n\circ G_1\circ(\text{Б}^-)^n\circ(\text{Ц}_{n+1})^n\circ\text{Б}^+\big)\circ\ldots\circ\big(\text{К}_n\circ(\text{Б}^+)^n\circ G_{m-1}\circ(\text{Б}^-)^n\circ(\text{Ц}_{n+1})^n\circ\text{Б}^+\big)\circ$$
$$\circ G_m\circ(\text{Б}^-)^{m-1}\circ H.$$

□

Предложение 33. *Семейство вычислимых по Тьюрингу функций замкнуто относительно оператора примитивной рекурсии.*

Доказательство. Пусть функция $f(x_1,\ldots,x_n,y)$ получена по схеме примитивной рекурсии из частичных функций $g(x_1,\ldots,x_n)$ и $h(x_1,\ldots,x_n,y,z)$ и пусть функции g и h вычислимы на машинах Тьюринга G и H соответственно. Определим машину F, вычисляющую функцию f.

Используя базовые машины, можно построить такие машины Тьюринга T_1, T_2, T_3, T_4, что схема работы F выглядит следующим образом:

$$q_1 01^{x_1}0\ldots01^{x_n}01^y0 \underset{T_1}{\Longrightarrow} 01^{x_1}0\ldots01^{x_n}001^y q_\alpha 01^{x_1}0\ldots01^{x_n}0 \underset{G}{\Longrightarrow}$$
$$\underset{G}{\Longrightarrow} 01^{x_1}0\ldots01^{x_n}001^y q_\beta 01^{g(\overline{x})}0\ldots0.$$

Заметим, что при $i=0$ имеет место равенство

$$01^{x_1}0\ldots01^{x_n}001^y q_\beta 01^{g(\overline{x})}0\ldots0 = 01^{x_1}0\ldots01^{x_n}01^i01^{y-i}q_\beta 01^{f(\overline{x},i)}0\ldots0.$$

Далее, для произвольного i в состоянии q_β проверяем, есть ли ещё единицы в массиве 1^{y-i}. Если единицы ещё есть, то переходим в состояние q_γ, если нет — в

состояние q_δ, т.е. происходит следующее разветвление:

$$01^{x_1}0\ldots01^{x_n}01^i01^{y-i}q_\beta01^{f(\overline{x},i)}0\ldots0 \underset{T_2}{\Longrightarrow}$$
$$\underset{T_2}{\Longrightarrow}\begin{cases}01^{x_1}0\ldots01^{x_n}01^i01^{y-i}q_\gamma01^{f(\overline{x},i)}0\ldots0, & \text{если } y-i>0,\\ 01^{x_1}0\ldots01^{x_n}01^i01^{y-i}q_\delta01^{f(\overline{x},i)}0\ldots0, & \text{если } y-i=0.\end{cases}$$

В состоянии q_γ, используя машину H, переходим к следующей итерации в цикле:

$$01^{x_1}0\ldots01^{x_n}01^i01^{y-i}q_\gamma01^{f(\overline{x},i)}0\ldots0 \underset{T_3}{\Longrightarrow}$$
$$\underset{T_3}{\Longrightarrow}01^{x_1}0\ldots01^{x_n}01^{i+1}01^{y-(i+1)}q_\varepsilon01^{x_1}0\ldots01^{x_n}01^i01^{f(\overline{x},i)}0\ldots0 \underset{H}{\Longrightarrow}$$
$$\underset{H}{\Longrightarrow}01^{x_1}0\ldots01^{x_n}01^{i+1}01^{y-(i+1)}q_\beta01^{f(\overline{x},i+1)}0\ldots0.$$

В состоянии q_δ выходим из цикла:

$$01^{x_1}0\ldots01^{x_n}01^y0q_\delta01^{f(\overline{x},y)}0\ldots0 \underset{T_4}{\Longrightarrow} q_001^{f(\overline{x},y)}0\ldots0.$$

Таким образом, $F=T_1\circ G\circ T_2[q_\gamma\to T_3\circ H|q_\delta\to T_4]$. □

Предложение 34. *Семейство вычислимых по Тьюрингу функций замкнуто относительно оператора минимизации.*

Доказательство. Пусть функция $f(x_1,\ldots,x_n)$ получена с помощью оператора минимизации из частичной функции $g(x_1,\ldots,x_n,y)$ и пусть функция g вычислима на машине Тьюринга G. Определим машину F, вычисляющую функцию f.

Используя базовые машины, можно построить такие машины Тьюринга T_1, T_2, T_3, T_4, что схема работы F выглядит следующим образом:

$$q_101^{x_1}0\ldots01^{x_n}0 \underset{T_1}{\Longrightarrow}01^{x_1}0\ldots01^{x_n}0q_\alpha01^{x_1}0\ldots01^{x_n}00.$$

Заметим, что при $i=0$ имеет место равенство

$$01^{x_1}0\ldots01^{x_n}0q_\alpha01^{x_1}0\ldots01^{x_n}00=01^{x_1}0\ldots01^{x_n}01^iq_\alpha01^{x_1}0\ldots01^{x_n}01^i0.$$

Далее, для произвольного i в состоянии q_α запускаем машину G:

$$01^{x_1}0\ldots01^{x_n}01^iq_\alpha01^{x_1}0\ldots01^{x_n}01^i0 \underset{G}{\Longrightarrow}01^{x_1}0\ldots01^{x_n}01^iq_\beta01^{g(\overline{x},i)}0\ldots0.$$

Если машина G остановилась, то в состоянии q_β проверяем, выполняется ли тождество $g(x_1,\ldots,x_n,i)=0$. Если $g(x_1,\ldots,x_n,i)>0$, то переходим в состояние q_γ, а если $g(x_1,\ldots,x_n,i)=0$, то в состояние q_δ, т.е. происходит следующее разветвление:

$$01^{x_1}0\ldots01^{x_n}01^iq_\beta01^{g(\overline{x},i)}0\ldots0 \underset{T_2}{\Longrightarrow}\begin{cases}01^{x_1}0\ldots01^{x_n}01^iq_\gamma01^{g(\overline{x},i)}0\ldots0, & \text{если } g(\overline{x},i)>0,\\ 01^{x_1}0\ldots01^{x_n}01^iq_\delta01^{g(\overline{x},i)}0\ldots0, & \text{если } g(\overline{x},i)=0.\end{cases}$$

В состоянии q_γ переходим к следующей итерации в цикле:

$$01^{x_1}0\ldots01^{x_n}01^iq_\gamma01^{g(\overline{x},i)}0\ldots0 \underset{T_3}{\Longrightarrow}01^{x_1}0\ldots01^{x_n}01^{i+1}q_\alpha01^{x_1}0\ldots01^{x_n}01^{i+1}0.$$

В состоянии q_δ выходим из цикла:

$$01^{x_1}0\ldots01^{x_n}01^iq_\delta00\ldots0 \underset{T_4}{\Longrightarrow}q_001^i0\ldots0.$$

Таким образом, $F=T_1\circ G\circ T_2[q_\gamma\to T_3|q_\delta\to T_4]$. □

Теорема 35 (о вычислимости ч.р.ф. на машинах Тьюринга)**.** *Любая частично рекурсивная функция является вычислимой по Тьюрингу.*

Доказательство. Пусть f — ч.р.ф. Следовательно, по определению существует конечная последовательность функций $f_0, \ldots, f_k$ такая, что $f_k = f$ и для любого $i \leqslant k$ функция f_i либо простейшая, либо получается из некоторых предыдущих с помощью оператора S, R или M. Индукцией по числу $k \in \omega$ докажем, что f вычислима по Тьюрингу.

Если $k = 0$, то f является простейшей. Следовательно, в силу предложения 31, функция f вычислима по Тьюрингу.

Пусть $k > 0$. Достаточно рассмотреть случай, когда f получена из некоторых f_j, $j < k$, с помощью одного из трёх операторов. В силу индукционного предположения, все функции f_j, где $j < k$, вычислимы по Тьюрингу. Отсюда, используя предложения 32–34, заключаем, что функция f тоже является вычислимой по Тьюрингу.

Теорема доказана. □

Упражнения

1. Доказать, что функции $s(x) = x + 1$ и $f(x) = 2x$ нельзя получить из простейших функций 0 и I_m^n, $m, n \in \omega$, конечным числом применений операторов суперпозиции и примитивной рекурсии.

2. Рассмотрим следующие *функции Аккермана*:

$$\begin{aligned} &B(0, y) = 2 + y; \\ &B(x + 1, 0) = \operatorname{sg}(x); \\ &B(x + 1, y + 1) = B(x, B(x + 1, y)); \\ &A(x) = B(x, x). \end{aligned}$$

   Назовём всюду определённую функцию $f(x_1, \ldots, x_n)$ *B-мажорируемой*, если существует $m \in \omega$ такое, что

$$f(x_1, \ldots, x_n) < B(m, \max(x_1, \ldots, x_n) + 3).$$

   Доказать, что:
   (а) $B(x, y)$ и $A(x)$ рекурсивны;
   (б) $B(x + 2, y + 1) \geqslant 2^{y+1}$;
   (в) $B(x + 1, y + 2) \geqslant B(x + 1, y + 1)$;
   (г) $B(x + 2, y + 3) \geqslant B(x + 1, y + 4)$;
   (д) простейшие функции B-мажорируемы;
   (е) функция, полученная из B-мажорируемых функций с помощью суперпозиции, B-мажорируема;
   (ж) функция, полученная из B-мажорируемых функций с помощью примитивной рекурсии, B-мажорируема;
   (з) функция $A(x)$ не является примитивно рекурсивной.

3. Пусть $f : \omega \xrightarrow{1-1} \omega$ — всюду определённая инъективная функция, вычислимая по Тьюрингу. Доказать, что обратная функция $f^{-1} : \operatorname{range}(f) \longrightarrow \omega$ тоже вычислима по Тьюрингу.

4. Построить машину Тьюринга, вычисляющую функцию $f(x)$, которая порождает все числа Фибоначчи, т. е. $f(0) = 0$, $f(1) = 1$, $f(x + 2) = f(x) + f(x + 1)$.

§ 18. Рекурсивность некоторых функций и отношений

Нашей дальнейшей целью является доказательство утверждения, обратного теореме 35. Для этого нам потребуется целая серия предварительных фактов и новых понятий. Всюду далее через $\overline{x}$ мы обозначаем кортеж $\langle x_1, \ldots, x_n \rangle$, при $n = 0$ этот кортеж считается пустым.

Лемма 36. *Все нульместные всюду определённые функции являются примитивно рекурсивными.*

Доказательство. Пусть $a \in \omega$ — произвольная константа. Возможны два случая. Если $a = 0$, то это по определению простейшая функция, и значит, она является п.р.ф. Если же $a > 0$, то $a = \underbrace{s(s \ldots s}_{a}(0)\ldots)$, т. е. a получена из простейших функций 0 и $s(x)$ с помощью a-кратного применения оператора S. □

Лемма 37. *Следующие функции являются примитивно рекурсивными:*

(а) $f(x, y) = x + y$;

(б) $f(x, y) = x \cdot y$;

(в) $f(x, y) = x^y$ (здесь $0^0 = 1$);

(г) $\operatorname{sg}(x) = \begin{cases} 0, & x = 0; \\ 1, & x > 0; \end{cases}$

(д) $\overline{\operatorname{sg}}(x) = \begin{cases} 1, & x = 0; \\ 0, & x > 0; \end{cases}$

(е) $f(x) = x \dot{-} 1 = \begin{cases} 0, & x = 0; \\ x - 1, & x > 0; \end{cases}$

(ж) $f(x, y) = x \dot{-} y = \begin{cases} 0, & x \leqslant y; \\ x - y, & x > y; \end{cases}$

(з) $f(x, y) = |x - y|$.

Доказательство. (а) См. пример в предыдущем параграфе.

(б) Для функции $f(x, y) = x \cdot y$ имеет место следующая схема примитивной рекурсии:

$$\begin{cases} f(x, 0) = 0, \\ f(x, y + 1) = xy + x = \varphi(xy, x) = \varphi(I^3_3(x, y, f(x, y)), I^3_1(x, y, f(x, y))), \end{cases}$$

где $\varphi(u, v) = u + v$ — п.р.ф. из предыдущего пункта, т. е. $f(x, y)$ получена с помощью оператора R из п.р.ф. $g(x) = 0 = o(x)$ и п.р.ф. $h(x, y, z) = \varphi(I^3_3(x, y, z), I^3_1(x, y, z))$, которая, в свою очередь, получена из п.р.ф. φ, I^3_3, I^3_1 с помощью оператора S.

(в) Доказывается аналогично путем выписывания соответствующей схемы примитивной рекурсии.

(г) Выписываем схему примитивной рекурсии:

$$\begin{cases} \operatorname{sg}(0) = 0, \\ \operatorname{sg}(x + 1) = 1 = s(0) = s(o(I^2_1(x, \operatorname{sg}(x)))). \end{cases}$$

Другими словами, $\operatorname{sg}(x)$ получена с помощью оператора R из 0-местной функции 0, которая является простейшей, и 2-местной функции $s(o(I^2_1(x, y)))$, которая является п.р.ф., поскольку получена суперпозициями из п.р.ф.

(д) Выписываем схему примитивной рекурсии:

$$\begin{cases} \overline{\mathrm{sg}}(0) = 1, \\ \overline{\mathrm{sg}}(x+1) = 0 = o(I_1^2(x, \overline{\mathrm{sg}}(x))). \end{cases}$$

Таким образом, $\overline{\mathrm{sg}}(x)$ получена с помощью оператора R из 0-местной функции 1, которая является п.р.ф. в силу предыдущей леммы, и 2-местной функции $o(I_1^2(x,y))$, которая является п.р.ф., так как получена суперпозициями из п.р.ф.

(е) См. пример в предыдущем параграфе.

(ж) Обозначим $f(x,y) = x \dot{-} y$. Тогда имеет место схема примитивной рекурсии:

$$\begin{cases} f(x,0) = x = I_1^1(x), \\ f(x,y+1) = x \dot{-} (y+1) = (x \dot{-} y) \dot{-} 1 = \psi(f(x,y)), \end{cases}$$

где $\psi(u) = u \dot{-} 1$ — п.р.ф. из предыдущего пункта. Выше мы воспользовались тождеством $x \dot{-} (y+1) = (x \dot{-} y) \dot{-} 1$, которое верно для любых $x, y \in \omega$.

(з) Заметим, что $|x-y| = (x \dot{-} y) + (y \dot{-} x)$ — суперпозиция примитивно рекурсивных функций. □

Лемма 38. *Если функция $g(\overline{x}, y)$ является ч.р.ф. (п.р.ф.), то функции $f(\overline{x}, y) = \sum\limits_{i=0}^{y} g(\overline{x}, i)$ и $h(\overline{x}, y) = \prod\limits_{i=0}^{y} g(\overline{x}, i)$ тоже являются ч.р.ф. (п.р.ф.).*

Доказательство. Частичная (примитивная) рекурсивность функции f вытекает из пункта (а) леммы 37 и следующей схемы примитивной рекурсии:

$$\begin{cases} f(\overline{x}, 0) = g(\overline{x}, 0), \\ f(\overline{x}, y+1) = f(\overline{x}, y) + g(\overline{x}, y+1). \end{cases}$$

Утверждение для функции h доказывается аналогично с использованием пункта (б) леммы 37. □

Определение. Говорят, что функция $f(\overline{x})$ получается с помощью *ограниченной минимизации* из всюду определённых функций $g(\overline{x}, y)$ и $h(\overline{x})$ и обозначается $f(\overline{x}) = \mu y \leqslant h(\overline{x})[g(\overline{x}, y) = 0]$, если для любых значений $\overline{x}$ выполняется

$$f(\overline{x}) = \begin{cases} y, & \text{если } g(\overline{x}, i) \neq 0 \text{ для всех } i < y, g(\overline{x}, y) = 0, \text{ и } y \leqslant h(\overline{x}), \\ h(\overline{x}) + 1 & \text{в противном случае.} \end{cases}$$

Предложение 39 (об ограниченной минимизации)**.** *Если функции g и h примитивно рекурсивны и функция f получена из g и h с помощью ограниченной минимизации, то f тоже примитивно рекурсивна.*

Доказательство. Докажем, что для любых $x_1, \ldots, x_n$ справедливо тождество

$$f(\overline{x}) = \sum_{i=0}^{h(\overline{x})} \mathrm{sg}\Big(\prod_{j=0}^{i} g(\overline{x}, j)\Big). \qquad (*)$$

Если $f(\overline{x}) = y \leqslant h(\overline{x})$, то для всех $i < y$ справедливо $\prod_{j=0}^{i} g(\overline{x}, j) > 0$ и, значит, $\mathrm{sg}\big(\prod_{j=0}^{i} g(\overline{x}, j)\big) = 1$, а для всех $i \geqslant y$ имеет место $\prod_{j=0}^{i} g(\overline{x}, j) = 0$. Следовательно, в правой части тождества $(*)$ единица суммируется y раз, т.е. правая часть равна y.

Если же $f(\overline{x}) = h(\overline{x})+1$, то $\prod_{j=0}^{i} g(\overline{x}, j) > 0$ для всех $0 \leqslant i \leqslant h(\overline{x})$. Следовательно, в правой части тождества $(*)$ единица суммируется $h(\overline{x}) + 1$ раз, т.е. правая часть тоже равна $h(\overline{x}) + 1$.

Таким образом, примитивная рекурсивность функции f следует из лемм 37, 38 и тождества $(*)$. □

Распространим понятие рекурсивности на класс всех отношений, заданных на ω. Для этого достаточно связать с каждым отношением некоторую частичную функцию, которая однозначно его задаёт, и назвать отношение рекурсивным, если таковой является данная функция.

Определение. Отношение (предикат) $R \subseteq \omega^n$ называется *рекурсивным (примитивно рекурсивным)*, если его характеристическая функция

$$\chi_R(x_1, \ldots, x_n) = \begin{cases} 1, & \text{если } \langle x_1, \ldots, x_n\rangle \in R, \\ 0, & \text{если } \langle x_1, \ldots, x_n\rangle \notin R \end{cases}$$

является рекурсивной (примитивно рекурсивной).

Напомним, что вместо $\langle x_1, \ldots, x_n\rangle \in R$ иногда пишут $R(x_1, \ldots, x_n)$ и говорят, что предикат R истинен на элементах $x_1, \ldots, x_n$. Если же $\langle x_1, \ldots, x_n\rangle \notin R$, то пишут $\neg R(x_1, \ldots, x_n)$ и говорят, что предикат R ложен на элементах $x_1, \ldots, x_n$.

Определение. Для $P, Q \subseteq \omega^n$ введём следующие обозначения для n-местных отношений:

$$\begin{aligned} &P \,\&\, Q = \{\overline{x} \in \omega^n \mid P(\overline{x}) \text{ истинно, и } Q(\overline{x}) \text{ истинно}\},\\ &P \vee Q = \{\overline{x} \in \omega^n \mid P(\overline{x}) \text{ истинно, или } Q(\overline{x}) \text{ истинно}\},\\ &P \to Q = \{\overline{x} \in \omega^n \mid \text{если } P(\overline{x}) \text{ истинно, то } Q(\overline{x}) \text{ истинно}\},\\ &\neg P = \{\overline{x} \in \omega^n \mid P(\overline{x}) \text{ ложно}\}. \end{aligned}$$

Кроме этого, для $R \subseteq \omega^{n+1}$ введём следующие обозначения для $(n+1)$-местных отношений:

$$\begin{aligned} &\exists i \leqslant y\, R(\overline{x}, i) = \{\langle \overline{x}, y\rangle \in \omega^{n+1} \mid \text{существует } i \leqslant y \text{ такой, что } R(\overline{x}, i) \text{ истинно}\},\\ &\forall i \leqslant y\, R(\overline{x}, i) = \{\langle \overline{x}, y\rangle \in \omega^{n+1} \mid \text{для всех } i \leqslant y \text{ отношение } R(\overline{x}, i) \text{ истинно}\},\\ &\exists i < y\, R(\overline{x}, i) = \{\langle \overline{x}, y\rangle \in \omega^{n+1} \mid \text{существует } i < y \text{ такой, что } R(\overline{x}, i) \text{ истинно}\},\\ &\forall i < y\, R(\overline{x}, i) = \{\langle \overline{x}, y\rangle \in \omega^{n+1} \mid \text{для всех } i < y \text{ отношение } R(\overline{x}, i) \text{ истинно}\}. \end{aligned}$$

Замечание. С теоретико-множественной точки зрения отношение $P\&Q$ совпадает с пересечением $P \cap Q$, отношение $P \vee Q$ — с объединением $P \cup Q$, а отношение $\neg P$ — с дополнением $\omega^n \setminus P$. Кроме этого, имеет место тождество $P \to Q = \neg P \vee Q$.

Предложение 40. *Если отношения $P(\overline{x})$ и $Q(\overline{x})$ рекурсивны (примитивно рекурсивны), то отношения $P(\overline{x})\&Q(\overline{x})$, $P(\overline{x}) \vee Q(\overline{x})$, $\neg P(\overline{x})$, $P(\overline{x}) \to Q(\overline{x})$ тоже рекурсивны (примитивно рекурсивны).*

Доказательство. Следует из леммы 37 и следующих тождеств:

$$\chi_{P\&Q}(\overline{x}) = \chi_P(\overline{x}) \cdot \chi_Q(\overline{x}), \qquad \chi_{P\vee Q}(\overline{x}) = \operatorname{sg}(\chi_P(\overline{x}) + \chi_Q(\overline{x})), \qquad \chi_{\neg P}(\overline{x}) = \overline{\operatorname{sg}}(\chi_P(\overline{x})),$$
$$\chi_{P\to Q}(\overline{x}) = \chi_{\neg P\vee Q}(\overline{x}) = \operatorname{sg}(\overline{\operatorname{sg}}(\chi_P(\overline{x})) + \chi_Q(\overline{x})).$$

□

Предложение 41. *Бинарные отношения* $=$, $\neq$, $<$, $>$, $\leqslant$, $\geqslant$ *являются примитивно рекурсивными.*

Доказательство. Примитивная рекурсивность данных отношений следует из леммы 37 и тождеств

$$\chi_{=}(x,y) = \overline{\mathrm{sg}}|x-y|, \qquad \chi_{\neq}(x,y) = \mathrm{sg}|x-y|,$$
$$\chi_{<}(x,y) = \mathrm{sg}(y \dot{-} x), \qquad \chi_{>}(x,y) = \mathrm{sg}(x \dot{-} y),$$
$$\chi_{\leqslant}(x,y) = \overline{\mathrm{sg}}(x \dot{-} y), \qquad \chi_{\geqslant}(x,y) = \overline{\mathrm{sg}}(y \dot{-} x).$$

□

Предложение 42. *Если отношение* $R(\overline{x}, i)$ *рекурсивно (примитивно рекурсивно), то отношения* $\exists i \leqslant y\, R(\overline{x}, i)$, $\forall i \leqslant y\, R(\overline{x}, i)$, $\exists i < y\, R(\overline{x}, i)$, $\forall i < y\, R(\overline{x}, i)$ *тоже рекурсивны (примитивно рекурсивны).*

Доказательство. Утверждение для отношения $P(\overline{x}, y) = \exists i \leqslant y\, R(\overline{x}, i)$ следует из леммы 38 и тождества

$$\chi_P(\overline{x}, y) = \mathrm{sg}\Big(\sum_{i=0}^{y} \chi_R(\overline{x}, i)\Big).$$

Утверждение для остальных отношений вытекает из предложений 40, 41 и эквивалентностей

$$\forall i \leqslant y\, R(\overline{x}, i) \Longleftrightarrow \neg\exists i \leqslant y\, \neg R(\overline{x}, i),$$
$$\exists i < y\, R(\overline{x}, i) \Longleftrightarrow \exists i \leqslant y (R(\overline{x}, i)\,\&\, i \neq y),$$
$$\forall i < y\, R(\overline{x}, i) \Longleftrightarrow \neg\exists i < y\, \neg R(\overline{x}, i).$$

□

Предложение 43 (о кусочном задании функции)**.** *Пусть* $R_0, \ldots, R_k \subseteq \omega^n$ *— рекурсивные (примитивно рекурсивные) отношения, такие что* $R_0 \cup \ldots \cup R_k = \omega^n$ *и* $R_i \cap R_j = \varnothing$ *при* $i \neq j$*. Пусть далее* $f_0(x_1, \ldots, x_n), \ldots, f_k(x_1, \ldots, x_n)$ *— рекурсивные (примитивно рекурсивные) функции. Тогда функция*

$$F(\overline{x}) = \begin{cases} f_0(\overline{x}), & \text{если } R_0(\overline{x}), \\ \cdots & \cdots \\ f_k(\overline{x}), & \text{если } R_k(\overline{x}) \end{cases}$$

тоже рекурсивна (примитивно рекурсивна).

Доказательство. Достаточно заметить, что $F(\overline{x}) = f_0(\overline{x}) \cdot \chi_{R_0}(\overline{x}) + \ldots + f_k(\overline{x}) \cdot \chi_{R_k}(\overline{x})$. □

Лемма 44. *Если функция* $g(\overline{x}, y)$ *рекурсивна (примитивно рекурсивна), то функция*

$$h(\overline{x}, y, z) = \begin{cases} \prod\limits_{i=y}^{z} g(\overline{x}, i), & \text{если } y \leqslant z, \\ 1, & \text{если } y > z, \end{cases}$$

тоже рекурсивна (примитивно рекурсивна).

Доказательство. (Примитивная) рекурсивность функции h следует из леммы 38, предложения 43 и следующей кусочной схемы:

$$h(\overline{x},y,z)=\begin{cases}\prod\limits_{i=0}^{z\dot{-}y} g(\overline{x},y+i), & \text{если } y\leqslant z,\\ 1, & \text{если } y>z.\end{cases}$$

□

Определение. Пусть $R(\overline{x},y)$ — отношение, $h(\overline{x})$ — всюду определённая функция. Обозначим через $\mu y[R(\overline{x},y)]$ функцию $\mu y[|\chi_R(\overline{x},y)-1|=0]$, а через $\mu y\leqslant h(\overline{x})[R(\overline{x},y)]$ — функцию $\mu y\leqslant h(\overline{x})[|\chi_R(\overline{x},y)-1|=0]$. Ясно, что если $R(\overline{x},y)$ рекурсивно, то $\mu y[R(\overline{x},y)]$ — ч.р.ф. Кроме этого, из предложения 39 следует, что если $R(\overline{x},y)$ и $h(\overline{x})$ примитивно рекурсивны, то $\mu y\leqslant h(\overline{x})[R(\overline{x},y)]$ — п.р.ф.

Лемма 45. (а) *Функция $[x/y]$, равная целой части от частного x/y, примитивно рекурсивна (по определению считаем, что $[x/0]=x$).*

(б) *Отношение* $\mathrm{Div}(x,y)$, *истинное тогда и только тогда, когда x делит y, примитивно рекурсивно.*

(в) *Отношение* $\mathrm{Prime}(x)$, *истинное тогда и только тогда, когда x — простое число, примитивно рекурсивно.*

Доказательство. Докажем утверждение пункта (а). Имеет место цепочка эквивалентностей $[x/y]=z \iff (y=0\ \&\ x=z)\vee(y\neq 0\ \&\ z\leqslant x/y<z+1) \iff$ $\iff (y=0\ \&\ x=z)\vee(y\neq 0\ \&\ zy\leqslant x<(z+1)y) \iff (y=0\ \&\ x=z)\vee$ $\vee\,(y\neq 0\ \&\ z$ – наименьшее, такое что $x<(z+1)y)$. Кроме этого, ясно, что $[x/y]\leqslant x$. Отсюда получаем

$$[x/y]=\mu z\leqslant x\Big[(y=0\ \&\ x=z)\vee(y\neq 0\ \&\ x<(z+1)y)\Big].$$

Таким образом, функция $[x/y]$ получена ограниченной минимизацией из примитивно рекурсивных функций, а значит, является п.р.ф.

Утверждения пунктов (б) и (в) следуют из эквивалентностей:

$$\mathrm{Div}(x,y)\iff\exists z\leqslant y(xz=y)$$
$$\mathrm{Prime}(x)\iff(x\geqslant 2)\ \&\ \forall y\leqslant x\Big(\mathrm{Div}(y,x)\longrightarrow(y=1\vee y=x)\Big).$$

□

Лемма 46. (а) *Функция $p(x)=p_x$, где $p_0=2,p_1=3,p_2=5,\ldots$ — перечисление всех простых чисел в порядке возрастания, является примитивно рекурсивной.*

(б) *Функция* $\mathrm{ex}(i,x)$, *равная показателю степени p_i в каноническом разложении числа x на простые множители, является примитивно рекурсивной (здесь* $\mathrm{ex}(i,0)=0$*).*

(в) *Функция* $\mathrm{long}(x)$, *равная номеру наибольшего простого делителя числа x, является примитивно рекурсивной (здесь* $\mathrm{long}(0)=\mathrm{long}(1)=0$*).*

Доказательство. (а) В силу теоремы Чебышёва, утверждающей, что для любого $n\geqslant 1$ среди чисел $n+1,n+2,\ldots,2n$ найдётся хотя бы одно простое, заключаем, что имеет место оценка $p_{x+1}\leqslant 2p_x$.

Отсюда следует, что функцию $p(x) = p_x$ можно получить по схеме примитивной рекурсии

$$\begin{cases} p_0 = 2, \\ p_{x+1} = \mu y \leqslant 2p_x\big[\mathrm{Prime}(y)\ \&\ y > p_x\big], \end{cases}$$

т.е. $p(x)$ получена с помощью оператора примитивной рекурсии из функций $g = 2$ и $h(x,z) = \mu y \leqslant 2z\big[\mathrm{Prime}(y)\ \&\ y > z\big]$, где функция $h(x,z)$, в свою очередь, получена с помощью ограниченной минимизации из примитивно рекурсивных функций и предикатов.

Так как участвующие в схеме функции g и $h(x,z)$ примитивно рекурсивны, то $p(x)$ является п.р.ф.

(б) Заметим, что для $x{>}0$ справедливы неравенства $\mathrm{ex}(i,x) < 2^{\mathrm{ex}(i,x)} \leqslant p_i^{\mathrm{ex}(i,x)} \leqslant x$. Отметим также, что для $x = 0$ неравенство $\mathrm{ex}(i,x) \leqslant x$ тоже верно. Поэтому функция $\mathrm{ex}(i,x)$ получается с помощью ограниченной минимизации

$$\mathrm{ex}(i,x) = \mu y \leqslant x[\neg\mathrm{Div}(p_i^{y+1}, x) \vee x = 0].$$

(в) Заметим, что если простое число p_i входит с ненулевым показателем в каноническое разложение числа $x > 1$, то $i < p_i \leqslant x$. Поэтому функция $\mathrm{long}(x)$ получается с помощью ограниченной минимизации

$$\mathrm{long}(x) = \mu y \leqslant x[x \leqslant 1 \vee \forall i \leqslant x(i > y \longrightarrow \mathrm{ex}(i,x) = 0)].$$

□

Упражнения

1. Доказать, что следующие функции примитивно рекурсивны:
   (а) $f(x) = x!$ (здесь $0! = 1$);
   (б) $\max(x,y)$;
   (в) $\min(x,y)$;
   (г) $\mathrm{rest}(x,y)$ — остаток от деления x на y (здесь $\mathrm{rest}(x,0) = 0$);
   (д) $f(x) = [\sqrt{x}]$;
   (е) $\tau(x)$ — число делителей числа x (здесь $\tau(0) = 0$);
   (ж) $\sigma(x)$ — сумма делителей числа x (здесь $\sigma(0) = 0$);
   (з) $\mathrm{lh}(x)$ — число простых делителей числа x (здесь $\mathrm{lh}(0) = 0$).

2. Доказать, что следующие функции частично рекурсивны:
   (а) нигде не определённая функция;
   (б) $f(x,y) = x - y$ (при $x < y$ значение функции не определено);
   (в) $f(x,y) = x/y$ (при x, не кратном y, значение функции не определено);
   (г) любая функция, область определения которой — конечное множество.

3. Построить примитивно рекурсивные функции $c : \omega^2 \to \omega$, $l : \omega \to \omega$, $r : \omega \to \omega$ такие, что функция c является биекцией и выполняются тождества $l(c(x,y))=x$, $r(c(x,y))=y$ (Указание: воспользоваться нумерацией пар натуральных чисел по принципу «канторовской таблицы»).

4. Доказать, что следующие отношения примитивно рекурсивны:
   (а) $\{x \in \omega \mid x$ — совершенное число$\}$ (число x называется *совершенным*, если оно совпадает с суммой всех своих делителей, отличных от x);
   (б) $\{x \in \omega \mid \exists n\,(x = 1^2 + \ldots + n^2)\}$;
   (в) $\{\langle x, y\rangle \in \omega^2 \mid 2 \leqslant x \leqslant 9$, и все цифры в десятичной записи y делятся на $x\}$.

5. Пусть рекурсивная функция $f(x)$ удовлетворяет условию: $f(x) \geqslant x$ для всех $x \in \omega$. Доказать, что область значений $\mathrm{range}(f)$ функции f рекурсивна.

6. Доказать, что бесконечное множество $A \subseteq \omega$ рекурсивно тогда и только тогда, когда $A = \mathrm{range}(f)$ для некоторой строго возрастающей одноместной рекурсивной функции f.

7. Пусть $\sqrt{2} = a_0.a_1a_2a_3\ldots$ — запись числа $\sqrt{2}$ в виде бесконечной десятичной дроби. Доказать, что функция $f(n) = a_n$ примитивно рекурсивна.

8. Пусть $e = a_0.a_1a_2a_3\ldots$ — запись числа e в виде бесконечной десятичной дроби. Доказать, что функция $f(n) = a_n$ примитивно рекурсивна.

§ 19. Кодирование машин Тьюринга

Мы постепенно приближаемся к доказательству теоремы о том, что любая функция, вычислимая на машине Тьюринга, является частично рекурсивной. Основная идея доказательства этой теоремы заключается в следующем: мы закодируем все вычисления на машинах Тьюринга натуральными числами таким образом, что все необходимые операции с полученными кодами (т. е. операции, имитирующие работу машины) можно будет производить с помощью частично рекурсивных функций. В этом параграфе будет формально описан математический аппарат, реализующий эту идею. Нам предстоит закодировать натуральными числами машинные слова, команды, программы и, наконец, преобразования машинных слов на машинах Тьюринга.

Все используемые ниже коды основаны на следующем способе однозначного кодирования кортежей натуральных чисел произвольной длины. Если $\langle x_0, \ldots, x_k\rangle$ — конечная последовательность элементов ω, то её *кодом* служит натуральное число $p_0^{x_0+1} \cdot p_1^{x_1+1} \cdot \ldots \cdot p_k^{x_k+1}$. Кодом пустой последовательности будем считать число 1. Если x — код некоторого кортежа, то можно эффективно найти его длину как $\mathrm{long}(x) + 1$ и восстановить все его элементы как $\mathrm{ex}(i, x) \dot{-} 1$, где $0 \leqslant i \leqslant \mathrm{long}(x)$.

Заметим также, что если длина рассматриваемых кортежей фиксирована, то можно не прибавлять единицы в показателях степеней простых чисел, т.е. в этом случае код вида $p_0^{x_0} \cdot p_1^{x_1} \cdot \ldots \cdot p_k^{x_k}$ тоже будет однозначным.

Определение. Пусть U — произвольное (возможно, пустое) слово в счётном алфавите $\{a_0, a_1, a_2, \ldots\}$, при этом мы считаем, что $a_0 = 0$, $a_1 = 1$. Определим *левый код* $\lfloor U\rfloor_l$ и *правый код* $\lfloor U\rfloor_r$ *слова* U следующим образом:

$$\lfloor U\rfloor_l = \begin{cases} 1, & \text{если } U = \Lambda, \\ p_k^{i_0+1} p_{k-1}^{i_1+1} \ldots p_0^{i_k+1}, & \text{если } U = a_{i_0}a_{i_1}\ldots a_{i_k}. \end{cases}$$

$$\lfloor U\rfloor_r = \begin{cases} 1, & \text{если } U = \Lambda, \\ p_0^{i_0+1} p_1^{i_1+1} \ldots p_k^{i_k+1}, & \text{если } U = a_{i_0}a_{i_1}\ldots a_{i_k}. \end{cases}$$

Определение. *Кодом машинного слова* $M{=}Uq_ia_jV$, где $q_i{\in}\{q_0, q_1, ...\}$, $a_j{\in}\{a_0, a_1, ...\}$, $U, V \in \{a_0, a_1, \ldots\}^*$, называется число

$$\lfloor M \rfloor = 2^{\lfloor U \rfloor_l} \cdot 3^i \cdot 5^j \cdot 7^{\lfloor V \rfloor_r}.$$

Определение. *Кодом команды* $T(i,j) = q_ia_j \to q_ka_lS$, где $i \geqslant 1$, $j \geqslant 0$, $k \geqslant 0$, $l \geqslant 0$, $S \in \{\Lambda, L, R\}$, называется число

$$\lfloor T(i,j) \rfloor = p_{2^i3^j}^{2^k3^l5^s},$$

где $s = 0$, если $S = \Lambda$; $s = 1$, если $S = L$; $s = 2$, если $S = R$.

Определение. Пусть $T = \langle \{a_0, \ldots, a_n\}, \{q_0, \ldots, q_m\}, P, q_1, q_0 \rangle$ — машина Тьюринга с программой $P = \{T(i,j) \mid 1 \leqslant i \leqslant m, 0 \leqslant j \leqslant n\}$. *Кодом машины* T называется произведение кодов всех её команд, т.е. число

$$\lfloor T \rfloor = \prod_{\substack{1 \leqslant i \leqslant m \\ 0 \leqslant j \leqslant n}} \lfloor T(i,j) \rfloor.$$

Определение. *Функцией одношагового преобразования кодов машинных слов* назовём любую всюду определённую функцию $f(t,w)$, удовлетворяющую условию: если $t = \lfloor T \rfloor$, где T — некоторая машина Тьюринга, $w = \lfloor M \rfloor$, где M — машинное слово в алфавите машины T, то

$$f(t,w) = \lfloor M'_T \rfloor.$$

Предложение 47. *Существует п.р.ф.* $\mathrm{step}(t,w)$, *которая является функцией одношагового преобразования кодов машинных слов.*

Доказательство. Пусть t является кодом некоторой машины T, а w — кодом некоторого машинного слова в алфавите машины T. Следовательно, $w = 2^u{\cdot}3^i{\cdot}5^j{\cdot}7^v$, где q_i входит в алфавит внутренних состояний данной машины, a_j — в её внешний алфавит, u — левый код некоторого слова U во внешнем алфавите, а v — правый код некоторого слова V во внешнем алфавите.

Параметры i, j, u, v можно вычислить с помощью следующих п.р.ф., зависящих от аргумента w:

$$i(w) = \mathrm{ex}(1,w), \quad j(w) = \mathrm{ex}(2,w), \quad u(w) = \mathrm{ex}(0,w), \quad v(w) = \mathrm{ex}(3,w).$$

Рассмотрим следующие шесть случаев, в каждом из которых мы определим значение $\mathrm{step}(t,w) = \lfloor (Uq_ia_jV)'_T \rfloor$.

(1) Пусть $i = 0$. Тогда $q_i = q_0$ и текущее состояние машины T является конечным. Следовательно, $(Uq_ia_jV)'_T = Uq_ia_jV$, и в данном случае

$$\mathrm{step}(t,w) = f_1(t,w) = 2^{u(w)} \cdot 3^0 \cdot 5^{j(w)} \cdot 7^{v(w)}.$$

Если $i > 0$, то в программе машины T найдётся команда $T(i,j) = q_ia_j \to q_ka_lS$, которая будет исполняться в данный момент. Поскольку код $p_{2^i3^j}^{2^k3^l5^s}$ данной команды входит в код t машины T, то параметры k, l, s можно вычислить с помощью следующих п.р.ф.:

$$k(t,w) = \mathrm{ex}(0, \mathrm{ex}(2^{i(w)}3^{j(w)}, t)),$$
$$l(t,w) = \mathrm{ex}(1, \mathrm{ex}(2^{i(w)}3^{j(w)}, t)),$$
$$s(t,w) = \mathrm{ex}(2, \mathrm{ex}(2^{i(w)}3^{j(w)}, t)).$$

(2) Пусть $i > 0$, $s = 0$. Тогда команда $T(i, j)$ имеет вид $q_i a_j \to q_k a_l$. Следовательно, $(Uq_i a_j V)'_T = Uq_k a_l V$, и в данном случае

$$\mathrm{step}(t, w) = f_2(t, w) = 2^{u(w)} \cdot 3^{k(t,w)} \cdot 5^{l(t,w)} \cdot 7^{v(w)}.$$

(3) Пусть $i > 0$, $s = 1$, $u = 1$. Тогда команда $T(i, j)$ имеет вид $q_i a_j \to q_k a_l L$ и применяется к машинному слову $M = q_i a_j V$. Следовательно, $(M)'_T = q_k a_0 a_l V$, и в данном случае

$$\mathrm{step}(t, w) = f_3(t, w) = 2^1 \cdot 3^{k(t,w)} \cdot 5^0 \cdot 7^{g_r(t,w)},$$

где $g_r(t, w) = \lfloor a_l V \rfloor_r = 2^{l(t,w)+1} \cdot \prod\limits_{e=0}^{\mathrm{long}(v(w))} p_{e+1}^{\mathrm{ex}(e,v(w))}$.

(4) Пусть $i > 0$, $s = 1$, $u > 1$. Тогда команда $q_i a_j \to q_k a_l L$ применяется к машинному слову $M = Uq_i a_j V$, причём слово $U = U' a_p$ не пусто. Следовательно, $(M)'_T = U' q_k a_p a_l V$, где $p = \mathrm{ex}(0, u(w)) \dot{-} 1$. Таким образом,

$$\mathrm{step}(t, w) = f_4(t, w) = 2^{h_l(w)} \cdot 3^{k(t,w)} \cdot 5^{\mathrm{ex}(0,u(w)) \dot{-} 1} \cdot 7^{g_r(t,w)},$$

где $h_l(w) = \lfloor U' \rfloor_l = \prod\limits_{e=0}^{\mathrm{long}(u(w)) \dot{-} 1} p_e^{\mathrm{ex}(e+1,u(w))}$, а функция $g_r(t, w)$ такая же, как выше.

(5) Пусть $i > 0$, $s = 2$, $v = 1$. Тогда команда $q_i a_j \to q_k a_l R$ применяется к машинному слову $M = Uq_i a_j$. Следовательно, $(M)'_T = Ua_l q_k a_0$, и в этом случае получаем

$$\mathrm{step}(t, w) = f_5(t, w) = 2^{g_l(t,w)} \cdot 3^{k(t,w)} \cdot 5^0 \cdot 7^1,$$

где $g_l(t, w) = \lfloor U a_l \rfloor_l = 2^{l(t,w)+1} \cdot \prod\limits_{e=0}^{\mathrm{long}(u(w))} p_{e+1}^{\mathrm{ex}(e,u(w))}$.

(6) Пусть $i > 0$, $s = 2$, $v > 1$. Тогда команда $q_i a_j \to q_k a_l R$ применяется к машинному слову $M = Uq_i a_j V$, причём слово $V = a_p V'$ не пусто. Следовательно, $(M)'_T = Ua_l q_k a_p V'$, где $p = \mathrm{ex}(0, v(w)) \dot{-} 1$. Таким образом,

$$\mathrm{step}(t, w) = f_6(t, w) = 2^{g_l(t,w)} \cdot 3^{k(t,w)} \cdot 5^{\mathrm{ex}(0,v(w)) \dot{-} 1} \cdot 7^{h_r(w)},$$

где $h_r(w) = \lfloor V' \rfloor_r = \prod\limits_{e=0}^{\mathrm{long}(v(w)) \dot{-} 1} p_e^{\mathrm{ex}(e+1,v(w))}$, а функция $g_l(t, w)$ такая же, как выше.

Заметим, что выше в рекурсивных схемах, определяющих значения $i(w)$, $j(w)$, $u(w)$, $v(w)$, $k(t, w)$, $l(t, w)$, $s(t, w)$, $f_1(t, w), \ldots, f_6(t, w)$, используются примитивно рекурсивные функции, которые определены на любых натуральных значениях t, w, в том числе на значениях t, w, не удовлетворяющих определению функции одношагового преобразования кодов машинных слов. Поэтому мы можем рассматривать $i(w)$, $j(w)$, $u(w)$, $v(w)$, $k(t, w)$, $l(t, w)$, $s(t, w)$, $f_1(t, w), \ldots, f_6(t, w)$ как всюду определённые функции, и значит, по построению они примитивно рекурсивны.

Определим теперь для произвольных $t, w \in \omega$ значение $\mathrm{step}(t, x)$ по следующей кусочной схеме:

$$\mathrm{step}(t, w) = \begin{cases} f_1(t, w), & \text{если } i(w) = 0, \\ f_2(t, w), & \text{если } i(w) > 0, s(t, w) = 0, \\ f_3(t, w), & \text{если } i(w) > 0, s(t, w) = 1, u(w) = 1, \\ f_4(t, w), & \text{если } i(w) > 0, s(t, w) = 1, u(w) > 1, \\ f_5(t, w), & \text{если } i(w) > 0, s(t, w) = 2, v(w) = 1, \\ f_6(t, w), & \text{если } i(w) > 0, s(t, w) = 2, v(w) > 1, \\ 0 & \text{в остальных случаях.} \end{cases}$$

В силу предложения 43, функция $\mathrm{step}(t, w)$ примитивно рекурсивна. Если $t = \lfloor T \rfloor$, где T — некоторая машина Тьюринга, $w = \lfloor M \rfloor$, где M — машинное слово в алфавите машины T, то по построению $\mathrm{step}(t, w) = \lfloor M'_T \rfloor$. Следовательно, $\mathrm{step}(t, w)$ является функцией одношагового преобразования кодов машинных слов. Что и требовалось доказать. □

Замечание. Заметим, что если t не является кодом никакой машины Тьюринга, или w не является кодом никакого машинного слова, или w — код машинного слова, содержащего символы, не входящие в алфавит машины, то всё равно функция $\mathrm{step}(t, w)$ из предожения 47 определена и вычисляет некоторое значение, которое нам на самом деле не важно. Важны только те значения $\mathrm{step}(t, w)$, которые получены из t, w, удовлетворяющих определению функции одношагового преобразования кодов машинных слов. Аналогичное замечание относится и к нескольким следующим предложениям.

Определение. *Функцией многошагового преобразования кодов машинных слов* назовём любую всюду определённую функцию $f(t, w, y)$, удовлетворяющую условию: если $t = \lfloor T \rfloor$, где T — некоторая машина Тьюринга, $w = \lfloor M \rfloor$, где M — машинное слово в алфавите машины T, то

$$f(t, w, y) = \left\lfloor M_T^{(y)} \right\rfloor.$$

Предложение 48. *Существует п.р.ф.* $\mathrm{run}(t, w, y)$*, которая является функцией многошагового преобразования кодов машинных слов.*

Доказательство. Искомая п.р.ф. $\mathrm{run}(t, w, y)$ определяется по схеме примитивной рекурсии

$$\begin{cases} \mathrm{run}(t, w, 0) = w, \\ \mathrm{run}(t, w, y + 1) = \mathrm{step}(t, \mathrm{run}(t, w, y)), \end{cases}$$

где $\mathrm{step}(t, x)$ — функция одношагового преобразования кодов машинных слов из предложения 47. □

Определение. Пусть $k \in \omega$ — фиксированное натуральное число. *Функцией кодирования входных машинных слов* назовём всюду определённую функцию

$$\mathrm{in}^k(x_1, \ldots, x_k) = \lfloor q_1 0 1^{x_1} 0 \ldots 0 1^{x_k} 0 \rfloor.$$

Предложение 49. *Функция* $\mathrm{in}^k(x_1, \ldots, x_k)$ *является примитивно рекурсивной.*

Доказательство. Положим $v(x_1, \ldots, x_k) = \lfloor 1^{x_1} 0 \ldots 0 1^{x_k} 0 \rfloor_r$. Если $k \geqslant 1$, то функция

$$v(x_1, \ldots, x_k) = \Big(\prod_{i=1}^{x_1} p_{i \dot{-} 1}^2\Big) \cdot p_{x_1} \cdot \Big(\prod_{i=1}^{x_2} p_{i+x_1}^2\Big) \cdot p_{x_1+x_2+1} \cdot \ldots \cdot \Big(\prod_{i=1}^{x_k} p_{i+x_1+\ldots+x_{k-1}+(k \dot{-} 2)}^2\Big) \cdot p_{x_1+\ldots+x_k+(k \dot{-} 1)}$$

и, следовательно, является примитивно рекурсивной в силу лемм 37, 44 и 46. Если же $k = 0$, то $v = \lfloor 0 \rfloor_r = 2$ является нульместной п.р.ф.

Тогда функция $\mathrm{in}^k(x_1, \ldots, x_k) = 2^1 \cdot 3^1 \cdot 5^0 \cdot 7^{v(x_1, \ldots, x_k)}$ примитивно рекурсивна. □

Определение. *Счётчиком единиц в коде выходного машинного слова* назовём любую всюду определённую функцию $f(w)$, удовлетворяющую следующему условию: если $w = \lfloor q_0 01^y 00^s \rfloor$ для некоторых $y, s \geqslant 0$, то $f(w) = y$.

Предложение 50. *Существует п.р.ф.* $\mathrm{out}(w)$, *которая является счётчиком единиц в коде выходного машинного слова.*

Доказательство. Искомая п.р.ф. определяется по следующей схеме:

$$\mathrm{out}(w) = \sum_{i=0}^{\mathrm{long}(\mathrm{ex}(3,w))} \overline{\mathrm{sg}}|\mathrm{ex}(i, \mathrm{ex}(3, w)) - 2|.$$

□

Определение. *Функцией текущего состояния* назовём любую всюду определённую функцию $f(t, x_1, \ldots, x_k, y)$, удовлетворяющую условию: если $t = \lfloor T \rfloor$, где T — некоторая машина Тьюринга, машинное слово $M = q_1 01^{x_1} 0 \ldots 01^{x_k} 0$ и $M_T^{(y)} = U q_i a_j V$, то $f(t, x_1 \ldots, x_k, y) = i$.

Предложение 51. *Существует п.р.ф.* $q(t, x_1, \ldots, x_k, y)$, *которая является функцией текущего состояния.*

Доказательство. Используя построенные в предложениях 48 и 49 функции, определим искомую п.р.ф. по следующей схеме:

$$q(t, x_1, \ldots, x_k, y) = \mathrm{ex}(1, \mathrm{run}(t, \mathrm{in}^k(x_1, \ldots, x_k), y)).$$

□

Упражнения

1. Доказать, что множество левых (правых) кодов всех слов в счётном алфавите $\{a_0, a_1, a_2, \ldots\}$ примитивно рекурсивно.

2. Доказать, что множество кодов всех машинных слов $U q_i a_j V$, где $i, j \in \omega$, $U, V \in \{a_0, a_1, \ldots\}^*$, примитивно рекурсивно.

3. Доказать, что множество кодов всех команд $q_i a_j \longrightarrow q_k a_l S$, где $i \in \omega \setminus \{0\}$, $j, k, l \in \omega$, $S \in \{\Lambda, L, R\}$, примитивно рекурсивно.

4. Доказать, что множество кодов всех машин Тьюринга примитивно рекурсивно.

5. Доказать, что существует п.р.ф. $f(t, w, y)$, удовлетворяющая условию: если $t = \lfloor T \rfloor$, где T — некоторая машина Тьюринга, $w = \lfloor M \rfloor$, где M — машинное слово в алфавите машины T, то $f(t, w, y)$ равно количеству новых ячеек, достроенных справа в процессе переработки слова M в слово $M_T^{(y)}$.

§ 20. Машины Тьюринга vs частично рекурсивные функции

Мы, наконец, готовы к тому, чтобы привести доказательство теоремы о частичной рекурсивности функций, вычислимых на машинах Тьюринга.

Теорема 52. *Любая функция, вычислимая по Тьюрингу, является частично рекурсивной.*

Доказательство. Пусть $f(x_1, \ldots, x_k)$ — произвольная частичная функция, вычислимая по Тьюрингу. Следовательно, существует машина T, которая вычисляет f. Пусть t — код T, а $\overline{x} = \langle x_1, \ldots, x_k \rangle$ — произвольные значения аргументов f.

Если $f(\overline{x})$ определено, то T перерабатывает слово $M = q_1 01^{x_1} 0 \ldots 01^{x_k} 0$ в слово $M' = q_0 01^{f(\overline{x})} 00^s$ для некоторого $s \geqslant 0$ без достраивания ячеек слева. Пусть при этом n — это количество шагов работы машины T, после исполнения которых она впервые попадает в состояние q_0. Следовательно, n является минимальным с условием $M_T^{(n)} = M'$. В силу предложений 48, 49, 51, заключаем, что $\mathrm{run}(t, \mathrm{in}^k(\overline{x}), n) = \lfloor M' \rfloor$ и $q(t, \overline{x}, n) = 0$. Тогда из минимальности n следует равенство $n = \mu y[q(t, \overline{x}, y) = 0]$. Отсюда, используя предложение 50, заключаем, что

$$f(\overline{x}) = \mathrm{out}(\mathrm{run}(t, \mathrm{in}^k(\overline{x}), n)) = \mathrm{out}(\mathrm{run}(t, \mathrm{in}^k(\overline{x}), \mu y[q(t, \overline{x}, y) = 0])).$$

Если же $f(\overline{x})$ не определено, то T, начав работу с входного машинного слова $M = q_1 01^{x_1} 0 \ldots 01^{x_k} 0$, никогда не остановится, т. е. q_0 не входит в $M_T^{(y)}$ для всех $y \in \omega$. Следовательно, для любого $y \in \omega$ имеет место $q(t, \overline{x}, y) \neq 0$, откуда следует, что значение $\mu y[q(t, \overline{x}, y) = 0]$ не определено. Таким образом, опять выполняется соотношение

$$f(\overline{x}) = \mathrm{out}(\mathrm{run}(t, \mathrm{in}^k(\overline{x}), \mu y[q(t, \overline{x}, y) = 0])).$$

Так как функция $q(t, \overline{x}, y)$ примитивно рекурсивна, то $\mu y[q(t, \overline{x}, y) = 0]$ — частично рекурсивная функция. Поскольку $\mathrm{in}^k(\overline{x})$, $\mathrm{run}(t, w, y)$ и $\mathrm{out}(w)$ примитивно рекурсивны, то функция $f(\overline{x}) = \mathrm{out}(\mathrm{run}(t, \mathrm{in}^k(\overline{x}), \mu y[q(t, \overline{x}, y) = 0]))$ частично рекурсивна. □

Из теоремы 52 можно вывести несколько важных следствий. Следующие три результата можно считать достаточным оправданием трудоёмкой работы, проделанной в предыдущих параграфах.

Следствие 53. *Частичная функция вычислима по Тьюрингу тогда и только тогда, когда она является частично рекурсивной.*

Доказательство. Следует из теорем 35 и 52. □

Следствие 54. *Любая ч.р.ф. может быть получена из простейших функций с помощью конечной последовательности применений операторов S, R или M, в которой общее число применений оператора M не превосходит 1.*

Доказательство. Если $f(\overline{x})$ — п.р.ф., то по определению $f(\overline{x})$ может быть получена из простейших функций с помощью конечной последовательности применений операторов S и R, без использования оператора M.

Если $f(\overline{x})$ — ч.р.ф., то из доказательства теоремы 52 следует, что

$$f(\overline{x}) = \mathrm{out}(\mathrm{run}(t, \mathrm{in}^k(\overline{x}), \mu y[q(t, \overline{x}, y) = 0])), \qquad (*)$$

где t — код машины Тьюринга, вычисляющей $f(\overline{x})$.

Функции $q(t, \overline{x}, y)$, $\mathrm{in}^k(\overline{x})$, $\mathrm{run}(t, w, y)$, $\mathrm{out}(w)$ примитивно рекурсивны и, следовательно, могут быть получены из простейших функций только с помощью операторов S и R. Таким образом, доказываемое утверждение вытекает из тождества $(*)$. □

Определение. Пусть $k \in \omega$, K — некоторое семейство k-местных частичных функций. $(k{+}1)$-местная частичная функция $F(x_0, x_1, \ldots, x_k)$ называется *универсальной для семейства* K, если выполняются следующие условия:

(1) для любого $n \in \omega$ существует $f \in K$ такая, что $F(n, x_1, \ldots, x_k) = f(x_1, \ldots, x_k)$;

(2) для любой $f \in K$ существует $n \in \omega$ такое, что $F(n, x_1, \ldots, x_k) = f(x_1, \ldots, x_k)$.

Другими словами, K совпадает с множеством функций $\{F(0, \overline{x}), F(1, \overline{x}), F(2, \overline{x}), \ldots\}$.

Теорема 55 (об универсальной ч.р.ф.)**.** *Существует $(k{+}1)$-местная ч.р.ф. $F(x_0, x_1, \ldots, x_k)$, универсальная для семейства всех k-местных ч.р.ф.*

Доказательство. В качестве F возьмём $(k{+}1)$-местную частичную функцию

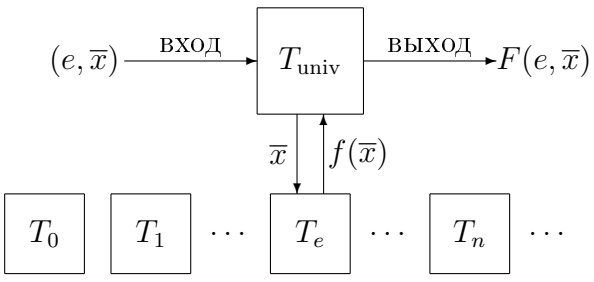
$$F(x_0, x_1, \ldots, x_k) = \mathrm{out}(\mathrm{run}(x_0, \mathrm{in}^k(x_1 \ldots, x_k), \mu y[q(x_0, x_1, \ldots, x_k, y) = 0])).$$

Из предложений 48–51 следует, что $F(x_0, x_1, \ldots, x_k)$ — ч.р.ф.

Ясно, что, фиксируя значение аргумента x_0 в $F(x_0, x_1, \ldots, x_k)$, мы получим некоторую k-местную ч.р.ф. Если же $f(x_1, \ldots, x_k)$ — произвольная k-местная ч.р.ф., то, взяв в качестве t код вычисляющей её машины Тьюринга, по теореме 52 получим, что $f(x_1, \ldots, x_k) = F(t, x_1, \ldots, x_k)$.

Таким образом, F — универсальная для всех k-местных ч.р.ф. □

Из теоремы об универсальной ч.р.ф. следует, что существует *универсальная машина Тьюринга* T_{univ}, которая в определённом смысле способна заменить бесконечное семейство всех машин Тьюринга. Работу T_{univ} можно описать следующим образом (см. рисунок): на вход универсальной машины подаётся кортеж данных $(e, \overline{x})$; универсальная машина по коду e конструирует программу машины T_e, соответствующую этому коду, и запускает её на входных данных $\overline{x}$; результат $f(\overline{x})$ работы машины T_e выдаётся на выход T_{univ} и является окончательным результатом её работы, т. е. $F(e, \overline{x}) = f(\overline{x})$.



Определение. Пусть $k \in \omega$. Для каждого $e \in \omega$ введём обозначение

$$\varphi_e^k(x_1, \ldots, x_k) = \mathrm{out}(\mathrm{run}(e, \mathrm{in}^k(x_1 \ldots, x_k), \mu y[q(e, x_1, \ldots, x_k, y) = 0])).$$

Функцию $\varphi_e^k(x_1, \ldots, x_k)$ будем называть *k-местной частично вычислимой функцией с клиниевским номером e*. В случае одноместных функций будем просто писать $\varphi_e(x)$ без верхнего индекса.

Замечание. Если e — код машины Тьюринга, которая вычисляет функцию f, то e также будет клиниевским номером этой функции. Но обратное, вообще говоря, неверно! Не любой клиниевский номер функции является кодом машины Тьюринга, вычисляющей эту функцию. См. замечание после предложения 47.

Следствие 56. *Существуют частичные функции, не являющиеся частично рекурсивными.*

Доказательство. Рассмотрим семейство $\{\varphi_0(x), \varphi_1(x), \varphi_2(x), \ldots\}$. В силу теоремы об универсальной ч.р.ф. оно совпадает с семейством всех одноместных ч.р.ф. Введём следующую всюду определённую одноместную функцию:

$$f(x) = \begin{cases} \varphi_x(x) + 1, & \text{если } \varphi_x(x) \text{ определено,} \\ 0, & \text{иначе.} \end{cases}$$

Допустим, f — ч.р.ф. Тогда найдётся $e \in \omega$ такое, что $f = \varphi_e$. Следовательно, φ_e тоже всюду определена. В частности, значение $\varphi_e(e)$ определено. Но тогда $\varphi_e(e) = f(e) = \varphi_e(e) + 1$. Противоречие. Следовательно, f не является частично рекурсивной. □

Упражнения

1. Доказать, что частичная функция f рекурсивна тогда и только тогда, когда существует конечный набор *всюду определённых* функций $f_0, \ldots, f_m = f$ такой, что для любого $i \leqslant m$ функция f_i либо простейшая, либо получается из некоторых предыдущих f_j, $j < i$, с помощью одного из операторов S, R или M.

2. Доказать, что предикат $T_k(t, x_1, \ldots, x_k, y, z)$, истинный тогда и только тогда, когда t — код некоторой машины Тьюринга T и машина T перерабатывает машинное слово $q_1 01^{x_1} 0 \ldots 01^{x_k} 0$ не более чем за y шагов в слово $q_0 01^z 00^s$ для некоторого $s \geqslant 0$, является примитивно рекурсивным.

3. Доказать, что следующая функция не является частично рекурсивной:

$$f(x) = \begin{cases} 1, & \text{если } x \text{ есть код некоторой машины Тьюринга } T \text{ и } T \\ & \text{останавливается, начав работу с машинного слова } q_1 01^x 0, \\ 0 & \text{в противном случае.} \end{cases}$$

§ 21. Универсальные функции

Мы уже убедились в том, что для семейства всех k-местных ч.р.ф. существует универсальная $(k+1)$-местная функция, которая сама является ч.р.ф. Однако для семейств всех k-местных п.р.ф. и всех k-местных р.ф. аналогичное свойство не имеет места.

Предложение 57. *Пусть $k \geqslant 1$. Не существует $(k+1)$-местной п.р.ф., универсальной для семейства всех k-местных п.р.ф.*

Доказательство. Допустим, напротив, $F(x_0, x_1, \ldots, x_k)$ — п.р.ф., универсальная для семейства всех k-местных п.р.ф., т. е. $\{F(0, x_1, \ldots, x_k), F(1, x_1, \ldots, x_k), \ldots\}$ — класс всех k-местных п.р.ф. Определим k-местную функцию

$$f(x_1, \ldots, x_k) = F(x_1, x_1, x_2, \ldots, x_k) + 1.$$

Тогда $f(x_1, \ldots, x_k)$ — п.р.ф. Следовательно, в силу универсальности F найдётся $n \in \omega$ такой, что для всех $x_1, \ldots, x_k \in \omega$ выполняется $F(n, x_1, \ldots, x_k) = f(x_1, \ldots, x_k)$. Рассмотрим значения $x_1 = n, x_2 = \ldots = x_k = 0$. Тогда получаем

$$F(n, n, 0, \ldots, 0) = f(n, 0, \ldots, 0) = F(n, n, 0, \ldots, 0) + 1.$$

Противоречие. □

Предложение 58. *Пусть $k \geqslant 1$. Не существует $(k{+}1)$-местной р.ф., универсальной для семейства всех k-местных р.ф.*

Доказательство. Повторяет доказательство предложения 57. □

Замечание. Условие $k \geqslant 1$ в формулировках предложений 57 и 58 присутствует не случайно. Для семейства всех 0-местных п.р.ф. (которое совпадает с семейством всех 0-местных р.ф.) существует универсальная п.р.ф. — это, очевидно, 1-местная функция $F(x) = x$.

Диагональный метод доказательства предложения 57 существенно опирается на предположение о примитивной рекурсивности функции $F(x_0, x_1, \ldots, x_k)$. Если это предположение ослабить и считать, что $F(x_0, x_1, \ldots, x_k)$ — рекурсивная, то диагональные рассуждения уже ни к чему не приводят, т. е. не доказывают отсутствия р.ф., универсальной для семейства всех п.р.ф. Более того, справедливо следующее утверждение, которое мы приводим без доказательства.

Предложение 59. *Существует $(k{+}1)$-местная р.ф., универсальная для семейства всех k-местных п.р.ф.* □

Полное доказательство предложения 59 изложено в [9]. В основе этого доказательства лежат теорема Робинсона о том, что все 1-местные п.р.ф. можно получить из функций $s(x) = x{+}1$ и $q(x) = x \dot{-} [\sqrt{x}]^2$ операциями сложения, суперпозиции и итерирования функций, и теорема о рекурсивности функций, полученных из некоторых п.р.ф. с помощью рекурсии 2-й ступени.

Из предложения 59 вытекает важное следствие, утверждающее, что класс всех п.р.ф. не совпадает с классом всех р.ф. Напомним, что справедливы следующие теоретико-множественные включения:

$$\text{ПРФ} \subseteq \text{РФ} \subseteq \text{ЧРФ}.$$

Каждое из этих включений — строгое. Второе включение является строгим, поскольку, очевидно, существуют не всюду определённые ч.р.ф. Например, нигде не определённая функция $f = \mu x[s(x) = 0]$ является ч.р.ф. Докажем, что первое включение является строгим.

Следствие 60. *Существует рекурсивная функция, не являющаяся примитивно рекурсивной.*

Доказательство. Пусть $F(x,y)$ — р.ф., универсальная для семейства всех 1-местных п.р.ф., которая существует в силу предложения 59. Если бы $F(x,y)$ была примитивно рекурсивной, то она была бы п.р.ф., универсальной для семейства всех 1-местных п.р.ф., что невозможно в силу предложения 57. □

Существует другой известный пример рекурсивной функции, которая не является примитивно рекурсивной, — это так называемая *функция Аккермана*. Определение и свойства функции Аккермана можно найти в [9].

Пример. Вопрос о существовании универсальной функции можно исследовать не только для семейств всех k-местных ч.р.ф., р.ф. или п.р.ф. Докажем, что существует 2-местная п.р.ф., универсальная для семейства всех полиномов от одной переменной с натуральными коэффициентами.

Если $f(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$ — полином, $a_i \in \omega$, $n \in \omega$, то сопоставим ему код$(f) = p_0^{a_0+1} \cdot p_1^{a_1+1} \cdot \ldots \cdot p_n^{a_n+1}$, где $p_0 = 2, p_1 = 3, p_2 = 5, \ldots$ — перечисление простых чисел в порядке возрастания.

Заметим, что если $k =$ код(f), то степень полинома f находится с помощью функции $\mathrm{long}(k)$, а i-й коэффициент f — с помощью функции $\mathrm{ex}(i,k) \dot{-} 1$.

Определим двухместную примитивно рекурсивную функцию

$$F(y,x) = \sum_{i=0}^{\mathrm{long}(y)} (\mathrm{ex}(i,y) \dot{-} 1) \cdot x^i.$$

Тогда, с одной стороны, если у функции $F(y,x)$ зафиксировать значение y, то получится некоторый полином от переменной x с натуральными коэффициентами. С другой стороны, если f — произвольный полином, то $F($код$(f), x) = f(x)$. Таким образом, F — искомая универсальная функция.

В заключение параграфа приведём сводную таблицу результатов о существовании универсальных функций для классов всех k-местных п.р.ф., всех k-местных р.ф. и всех k-местных ч.р.ф ($k \geqslant 1$).

Класс функций	Универсальная п.р.ф.	Универсальная р.ф.	Универсальная ч.р.ф.
Класс всех k-местных п.р.ф.	−	+	+
Класс всех k-местных р.ф.	−	−	−
Класс всех k-местных ч.р.ф.	−	−	+

Для класса всех k-местных п.р.ф. не существует $(k{+}1)$-местной универсальной п.р.ф. в силу предложения 57.

Для класса всех k-местных п.р.ф. существуют $(k{+}1)$-местные универсальные р.ф. и ч.р.ф. в силу предложения 59.

Для класса всех k-местных р.ф. не существует $(k{+}1)$-местных универсальных п.р.ф. и р.ф. в силу предложения 58.

Для класса всех k-местных р.ф. не существует $(k{+}1)$-местной универсальной ч.р.ф., так как в противном случае универсальная ч.р.ф. для данного класса была бы всюду определённой.

Для класса всех k-местных ч.р.ф. не существует $(k{+}1)$-местных универсальных п.р.ф. и р.ф., поскольку в данном классе есть не всюду определённые функции.

Для класса всех k-местных ч.р.ф. существует $(k{+}1)$-местная универсальная ч.р.ф. в силу теоремы 55.

Упражнения

1. Доказать, что не существует двухместной рекурсивной функции $F(y,x)$, универсальной для семейства всех одноместных рекурсивных функций $f(x)$ со свойством $f(x) \leqslant 2^x$ для всех $x \in \omega$.

2. Доказать, что существует двухместная частично рекурсивная функция $F(y,x)$, универсальная для семейства всех одноместных дробно-рациональных функций вида $f(x) = p(x)/q(x)$, где $p(x)$ и $q(x)$ — многочлены от переменной x с натуральными коэффициентами.

3. Доказать, что существует двухместная рекурсивная функция $F(y,x)$, универсальная для семейства всех одноместных ступенчатых функций, т. е. функций вида
$$f(x) = \begin{cases} a_0, & x \leqslant b_0, \\ a_1, & b_0 < x \leqslant b_1, \\ \dots & \dots \\ a_{n-1}, & b_{n-2} < x \leqslant b_{n-1}, \\ a_n, & b_{n-1} < x, \end{cases}$$
где $a_0, \dots, a_n \in \omega$, $b_0, \dots, b_{n-1} \in \omega$, $n \geqslant 1$, $b_0 < b_1 < \dots < b_{n-1}$.

4. Доказать, что существует двухместная частично рекурсивная функция $F(y,x)$, универсальная для семейства всех одноместных функций, определённых лишь в конечном числе точек, т. е. функций вида
$$f(x) = \begin{cases} a_0, & x = b_0, \\ \dots & \dots \\ a_{n-1}, & x = b_{n-1}, \\ \text{не определено}, & \text{иначе}, \end{cases}$$
где $a_0, \dots, a_{n-1} \in \omega$, $b_0, \dots, b_{n-1} \in \omega$, $n \in \omega$, и числа $b_0, b_1, \dots, b_{n-1}$ попарно различны.

Глава V

Теория вычислимости

Данная глава является *введением* в теорию вычислимости и содержит фундаментальные понятия и теоремы из нескольких, ставших уже классическими разделов этой теории. Для дальнейшего её изучения можно порекомендовать книги [2,9,12,13].

Основной результат предыдущей главы позволяет нам ввести следующее определение

Определение. Частичная функция f называется *частично вычислимой (ч.в.ф.)*, если она удовлетворяет любому из следующих двух условий:

(1) f является частично рекурсивной функцией;

(2) f вычислима по Тьюрингу.

Всюду определённую частично вычислимую функцию будем называть *вычислимой функцией (в.ф.)*.

Далее мы будем использовать термин *ч.в.ф.*, поскольку для нас теперь не имеет значения, какая из формализаций понятия вычислимой функции используется в рассуждениях.

В теории вычислимости важную роль играет введённая в предыдущей главе клиниевская нумерация φ_e^k для класса всех k-местных частично вычислимых функций. Напомним, что её определение задаётся тождеством

$$\varphi_e^k(x_1,\ldots,x_k) = \mathrm{out}(\mathrm{run}(e, \mathrm{in}^k(x_1\ldots,x_k), \mu y[q(e,x_1,\ldots,x_k,y)=0])),$$

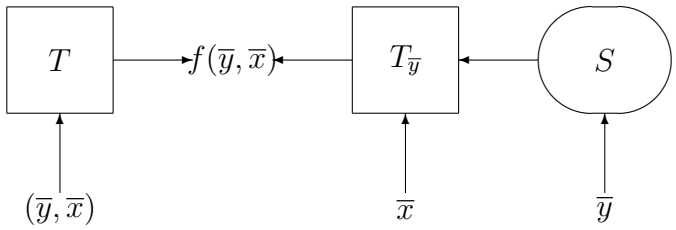
правая часть которого является $(k{+}1)$-местной ч.в.ф., универсальной для семейства всех k-местных ч.в.ф.

§ 22. Теорема о параметризации

В данном параграфе мы докажем фундаментальную теорему теории вычислимости — теорему о параметризации. Интуитивный смысл данной теоремы состоит в следующем. Если задана некоторая ч.в.ф. $f(\overline{y},\overline{x})$, переменные которой условно разделены на две группы, то к вычислению этой функции можно подойти двумя способами.

Первый способ — стандартный: мы целиком подаём на вход машины T, вычисляющей f, весь набор $\langle\overline{y},\overline{x}\rangle$ и получаем на выходе значение $f(\overline{y},\overline{x})$.

Второй способ: мы фиксируем значения переменных $\overline{y}$, считая их параметрами, и преобразуем программу машины T с помощью специального *параметризатора* в программу новой машины $T_{\overline{y}}$, которая уже будет вычислять функцию f как функцию от оставшихся переменных $\overline{x}$. Код машины $T_{\overline{y}}$ зависит от значений $\overline{y}$, но основное свойство заключается в том, что этот код может быть вычислен по $\overline{y}$ с помощью некоторой в.ф. $s(\overline{y})$. Строго говоря, параметризатор — это и есть функция s, которая, используя значения $\overline{y}$, преобразует код e машины T в код $s(\overline{y})$ машины $T_{\overline{y}}$.



Для доказательства теоремы о параметризации нам потребуется следующая техническая лемма.

Лемма 61. *Существует двухместная вычислимая функция $x \circ y$, удовлетворяющая условию: если $x = \lfloor T_1 \rfloor$, $y = \lfloor T_2 \rfloor$, где T_1 и T_2 — некоторые машины Тьюринга, имеющие один и тот же внешний алфавит, то $x \circ y = \lfloor T_1 \circ T_2 \rfloor$.*

Доказательство. Пусть x — код некоторой машины $T_1 = \langle A, \{q_0, \ldots, q_r\}, P_1, q_1, q_0\rangle$, а y — код некоторой машины $T_2 = \langle A, \{q_0, \ldots, q_s\}, P_2, q_1, q_0\rangle$, причём T_1 и T_2 имеют один и тот же внешний алфавит $A = \{a_0, \ldots, a_n\}$. Тогда их композиция $T_1 \circ T_2 = \langle A, \{q_0, \ldots, q_{r+s}\}, P, q_1, q_0\rangle$ имеет программу

$$P = \big(P_1\big)^{q_0}_{q_{r+1}} \cup \big(P_2\big)^{q_1,\ldots,q_s}_{q_{r+1},\ldots,q_{r+s}}.$$

Наибольший индекс состояния машины с кодом t находится с помощью вычислимой функции $m(t) = \mathrm{ex}(0, \mathrm{long}(t))$, а наибольший индекс символа внешнего алфавита — с помощью функции $n(t) = \mathrm{ex}(1, \mathrm{long}(t))$. Заметим, что, в силу наших предположений, $n(x) = n(y)$.

Тогда, используя вычислимые функции

$$\begin{aligned} k(t,i,j) &= \mathrm{ex}(0, \mathrm{ex}(2^i 3^j, t)),\\ l(t,i,j) &= \mathrm{ex}(1, \mathrm{ex}(2^i 3^j, t)),\\ s(t,i,j) &= \mathrm{ex}(2, \mathrm{ex}(2^i 3^j, t)), \end{aligned}$$

можно представить коды x и y в виде

$$x = \prod_{\substack{1 \leqslant i \leqslant m(x)\\ 0 \leqslant j \leqslant n(x)}} p_{2^i 3^j}^{2^{k(x,i,j)} \cdot 3^{l(x,i,j)} \cdot 5^{s(x,i,j)}}, \quad y = \prod_{\substack{1 \leqslant i \leqslant m(y)\\ 0 \leqslant j \leqslant n(y)}} p_{2^i 3^j}^{2^{k(y,i,j)} \cdot 3^{l(y,i,j)} \cdot 5^{s(y,i,j)}}.$$

Следующие две вычислимые функции осуществляют замену индексов состояний в кодах программ P_1 и P_2 в соответствии с определением композиции машин:

$$g(x,i,j) = \begin{cases} k(x,i,j), & \text{если } k(x,i,j) \neq 0,\\ m(x)+1, & \text{если } k(x,i,j) = 0, \end{cases}$$

$$h(x,y,i,j) = \begin{cases} k(y,i,j) + m(x), & \text{если } k(y,i,j) \neq 0,\\ 0, & \text{если } k(y,i,j) = 0. \end{cases}$$

Тогда код композиции $T_1 \circ T_2$ вычисляется с помощью функции

$$x \circ y = \prod_{\substack{1 \leqslant i \leqslant m(x)\\ 0 \leqslant j \leqslant n(x)}} p_{2^i 3^j}^{2^{g(x,i,j)} \cdot 3^{l(x,i,j)} \cdot 5^{s(x,i,j)}} \times \prod_{\substack{1 \leqslant i \leqslant m(y)\\ 0 \leqslant j \leqslant n(y)}} p_{2^{i+m(x)} 3^j}^{2^{h(x,y,i,j)} \cdot 3^{l(y,i,j)} \cdot 5^{s(y,i,j)}}. \qquad (*)$$

Заметим, что если x, y не удовлетворяют условию леммы, то функция в правой части тождества $(*)$ всё равно определена и вычисляет некоторое значение, которое нам на самом деле не важно. Поэтому можно считать, что тождество $(*)$ определяет значения функции $x \circ y$ для произвольных натуральных x, y. Отсюда окончательно заключаем, что построенная вычислимая функция $x \circ y$ является искомой. □

Теорема 62 (о параметризации). *Для любой частично вычислимой функции $f(y_1, \ldots, y_m, x_1, \ldots, x_n)$ существует вычислимая функция $s(y_1, \ldots, y_m)$ такая, что*

$$f(y_1, \ldots, y_m, x_1, \ldots, x_n) = \varphi^n_{s(y_1,\ldots,y_m)}(x_1, \ldots, x_n).$$

Доказательство. Обозначим для краткости $\overline{y} = \langle y_1, \ldots, y_m \rangle$, $\overline{x} = \langle x_1, \ldots, x_n \rangle$ и введём функцию $f'(\overline{x}, \overline{y})$, положив по определению $f'(\overline{x}, \overline{y}) = f(\overline{y}, \overline{x})$. Ясно, что $f'(\overline{x}, \overline{y})$ — ч.в.ф. Следовательно, существует машина Тьюринга F', которая её вычисляет.

Для каждого $y \in \omega$ определим вспомогательную машину H_y, которая осуществляет следующее преобразование:

$$q_1 0 \underset{H_y}{\Longrightarrow} 01^y q_0 0.$$

Программа для H_0 состоит из одной команды $q_1 0 \to q_0 0R$. Если же $y \geqslant 1$, то программа для H_y имеет вид

$$\begin{array}{l} q_1 0 \to q_2 0R \\ q_2 0 \to q_3 1R \\ \ldots \quad \ldots \\ q_y 0 \to q_{y+1} 1R \\ q_{y+1} 0 \to q_0 1R. \end{array}$$

Добавим в эту программу для каждого $1 \leqslant i \leqslant y+1$ фиктивную команду $q_i 1 \to q_0 0$. Тогда код машины H_y вычисляется с помощью функции

$$h(y) = \begin{cases} p_2^{5^2} \cdot p_6, & \text{если } y = 0, \\ p_2^{2^2 \cdot 5^2} \cdot \Big(\prod\limits_{i=2}^{y} p_{2^i}^{2^{i+1} \cdot 3 \cdot 5^2} \Big) \cdot p_{2^{y+1}}^{3 \cdot 5^2} \cdot \Big(\prod\limits_{i=1}^{y+1} p_{2^i 3} \Big), & \text{если } y \geqslant 1. \end{cases}$$

Зафиксируем теперь значения $\overline{y}$, считая их параметрами, и рассмотрим функцию $g(\overline{x}) = f'(\overline{x}, \overline{y})$. Построим машину $T_{\overline{y}}$, вычисляющую $g(\overline{x})$. Схема работы $T_{\overline{y}}$ выглядит следующим образом:

$$q_1 01^{x_1} 0 \ldots 01^{x_n} 0 \underset{(\text{Б}^+)^n}{\Longrightarrow} 01^{x_1} 0 \ldots 01^{x_n} q_\alpha 0 \underset{H_{y_1}}{\Longrightarrow} 01^{x_1} 0 \ldots 01^{x_n} 01^{y_1} q_{\beta_1} 0 \underset{H_{y_2}}{\Longrightarrow}$$
$$\underset{H_{y_2}}{\Longrightarrow} 01^{x_1} 0 \ldots 01^{x_n} 01^{y_1} 01^{y_2} q_{\beta_2} 0 \underset{H_{y_3}}{\Longrightarrow} \ldots \underset{H_{y_m}}{\Longrightarrow} 01^{x_1} 0 \ldots 01^{x_n} 01^{y_1} 0 \ldots 01^{y_m} q_{\beta_m} 0 \underset{(\text{Б}^-)^{m+n}}{\Longrightarrow}$$
$$\underset{(\text{Б}^-)^{m+n}}{\Longrightarrow} q_\gamma 01^{x_1} 0 \ldots 01^{x_n} 01^{y_1} 0 \ldots 01^{y_m} 0 \underset{F'}{\Longrightarrow} q_0 1^{f'(\overline{x},\overline{y})} 0 \ldots 0.$$

(Если $f'(\overline{x}, \overline{y})$ не определено, то выше в схеме машина F' не останавливается.)

Таким образом $T_{\overline{y}} = (\text{Б}^+)^n \circ H_{y_1} \circ \ldots \circ H_{y_m} \circ (\text{Б}^-)^{m+n} \circ F'$. Пусть $e_1 = \lfloor (\text{Б}^+)^n \rfloor$ и $e_2 = \lfloor (\text{Б}^-)^{m+n} \circ F' \rfloor$. Лемма 61 позволяет нам вычислить код машины $T_{\overline{y}}$ с помощью функции

$$s(\overline{y}) = e_1 \circ h(y_1) \circ \ldots \circ h(y_m) \circ e_2.$$

Поскольку функции $x \circ y$ и $h(y)$ вычислимы, заключаем, что $s(\overline{y})$ тоже вычислима. По построению $\varphi^n_{s(\overline{y})}(\overline{x}) = g(\overline{x}) = f'(\overline{x}, \overline{y}) = f(\overline{y}, \overline{x})$. Следовательно, функция $s(\overline{y})$ — искомая. □

В заключение параграфа мы докажем s-m-n-теорему, которая связывает нумерации φ_e^k для различных k.

Следствие 63 (s-m-n-теорема)**.** *Для любых $m, n \in \omega$ существует $(m{+}1)$-местная вычислимая функция $s_n^m(e, y_1, \ldots, y_m)$ такая, что*

$$\varphi_e^{m+n}(y_1, \ldots, y_m, x_1, \ldots, x_n) = \varphi^n_{s_n^m(e, y_1, \ldots, y_m)}(x_1, \ldots, x_n).$$

Доказательство. Рассмотрим $(m{+}n{+}1)$-местную ч.в.ф. $f(e, \overline{y}, \overline{x}) = \varphi_e^{m+n}(\overline{y}, \overline{x})$. По теореме о параметризации существует в.ф. $s(e, \overline{y})$ такая, что $f(e, \overline{y}, \overline{x}) = \varphi^n_{s(e,\overline{y})}(\overline{x})$. Функция s является искомой s-m-n-функцией. □

Упражнения

1. Доказать, что существует двухместная вычислимая функция, которая по клиниевским номерам одноместных функций $f(x)$ и $g(x)$ вычисляет клиниевский номер их суперпозиции $f(g(x))$.

2. Доказать, что существует одноместная вычислимая функция, которая по клиниевскому номеру одноместной функции $f(x)$ вычисляет клиниевский номер функции $g(x)$, полученной из $f(x)$ по следующей схеме:

$$\begin{cases} g(0) = 0, \\ g(x+1) = f(g(x)). \end{cases}$$

3. Доказать, что существует двухместная вычислимая функция $f(x, y)$ такая, что для любых $x, y \in \omega$ справедливо тождество $\mathrm{dom}(\varphi_{f(x,y)}) = \mathrm{dom}(\varphi_x) \cap \mathrm{dom}(\varphi_y)$.

4. Доказать, что существуют одноместные вычислимые функции $f(x)$ и $g(x)$ такие, что $\mathrm{dom}(\varphi_x) = \mathrm{range}(\varphi_{f(x)})$ и $\mathrm{range}(\varphi_x) = \mathrm{dom}(\varphi_{g(x)})$ для всех $x \in \omega$.

§ 23. Теорема о неподвижной точке

Теорема о неподвижной точке (другое название — теорема о рекурсии) была доказана Клини и является одним из наиболее важных результатов теории вычислимости. Её короткое доказательство использует s-m-n-теорему и на первый взгляд кажется несколько мистическим.

Теорема 64 (о неподвижной точке)**.** *Для любой частично вычислимой функции $f(x)$ существует $a \in \omega$ такое, что $\varphi_{f(a)} = \varphi_a$.*

Доказательство. Рассмотрим частично вычислимую функцию $F(e, y, x) = \varphi_e^2(y, x)$, которая является универсальной для семейства всех 2-местных ч.в.ф. По s-m-n-теореме существует вычислимая функция $s(e, y)$ такая, что

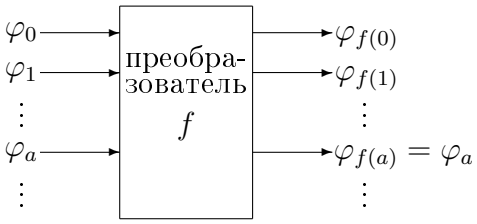
$$\varphi_e^2(y, x) = \varphi_{s(e,y)}(x). \qquad (*)$$

Функция $\varphi_{f(s(y,y))}(x)$ является 2-местной ч.в.ф. от переменных $\langle y, x\rangle$. Следовательно, в силу универсальности найдётся клиниевский номер $n \in \omega$ такой, что

$$\varphi_{f(s(y,y))}(x) = \varphi_n^2(y, x). \qquad (**)$$

Из $(*)$ и $(**)$ при $e = n$ следует, что $\varphi_{f(s(y,y))}(x) = \varphi_{s(n,y)}(x)$. Подставив в данное тождество $y = n$, получим $\varphi_{f(s(n,n))}(x) = \varphi_{s(n,n)}(x)$. Отсюда видно, что натуральное число $a = s(n, n)$ является искомым. □

Функцию $f(x)$ из теоремы о неподвижной точке можно неформально представлять как эффективный преобразователь программ, который любую программу n, реализующую процедуру φ_n, перерабатывает в программу $f(n)$, реализующую, вообще говоря, какую-то другую процедуру $\varphi_{f(n)}$. Интуитивный смысл теоремы о неподвижной точке заключается в том, что для любого такого преобразователя найдётся хотя бы одна процедура φ_a, которая не изменяется под его действием, т. е. $\varphi_{f(a)} = \varphi_a$. Другими словами, у любого такого преобразователя найдётся неподвижная точка-процедура.



Замечание. Теорема о неподвижной точке допускает следующее обобщение с параметрами $\overline{x}$: *для любой частично вычислимой функции $f(\overline{x}, y)$ существует вычислимая функция $g(\overline{x})$ такая, что $\varphi_{f(\overline{x},g(\overline{x}))} = \varphi_{g(\overline{x})}$*. Доказывается данное обобщение так же, как и теорема о неподвижной точке в классической формулировке.

Пример. В программировании известна популярная задача-головоломка о написании программ, воспроизводящих свой собственный текст (такие программы называют словом «quine»).

Оказывается, существование подобных программ близко связано с теоремой о неподвижной точке. В определённом смысле, если для языка программирования справедлива теорема о параметризации, то можно написать программу на данном языке, которая выводит свой собственный код.

Докажем существование подобной программы для языка машин Тьюринга, т. е. необходимо показать, что существует программа P с кодом e, которая после запуска всегда останавливается и выдаёт (в качестве выходного значения на ленте) свой собственный код e. Для этого рассмотрим вычислимую функцию $g(x, y) = x$ и применим к ней теорему о параметризации — получим некоторую вычислимую функцию $f(x)$ такую, что $g(x, y) = \varphi_{f(x)}(y)$. Далее используем теорему о неподвижной точке для функции $f(x)$ — получим число $a \in \omega$ такое, что $\varphi_a = \varphi_{f(a)}$. Из доказательства теоремы о неподвижной точке видно, что $a = s(n, n)$ для некоторой s-m-n-функции s, а из доказательства теоремы о параметризации вытекает, что любая s-m-n-функция всегда в качестве своих значений выдает коды некоторых программ. Отсюда заключаем, что найденное a является кодом программы, и программа с кодом a вычисляет функцию

$$\varphi_a(y) = \varphi_{f(a)}(y) = g(a, y) = a,$$

т. е. выдаёт свой собственный код a. □

Теорема о неподвижной точке имеет очень важное следствие, которое утверждает, что никакое нетривиальное (т. е. присущее некоторым, но не всем функциям) свойство частично вычислимых функций не может быть эффективно распознаваемым

по их клиниевским номерам. Для доказательства данного утверждения нам понадобится следующее определение, которое является переформулировкой определения рекурсивного отношения в терминах вычислимых функций.

Определение. Множество $A \subseteq \omega^n$ называется *вычислимым*, если его характеристическая функция $\chi_A(x_1, \ldots, x_n)$ является вычислимой.

Теорема 65 (теорема Райса)**.** *Пусть S — произвольное семейство одноместных ч.в.ф. такое, что $S \neq \varnothing$ и S не совпадает с семейством всех одноместных ч.в.ф. Тогда множество $\{x \in \omega \mid \varphi_x \in S\}$ всех клиниевских номеров функций, принадлежащих семейству S, невычислимо.*

Доказательство. Допустим, напротив, множество $A = \{x \in \omega \mid \varphi_x \in S\}$ вычислимо, т. е. вычислимой является его характеристическая функция

$$\chi_A(x) = \begin{cases} 1, & \text{если } x \in A, \\ 0, & \text{если } x \notin A. \end{cases}$$

Так как $S \neq \varnothing$ и $S \neq \{f \mid f$ — 1-местная ч.в.ф.$\}$, то найдутся $a, b \in \omega$ такие, что $\varphi_a \in S$ и $\varphi_b \notin S$. Определим функцию

$$f(x) = \begin{cases} b, & \text{если } x \in A, \\ a, & \text{если } x \notin A. \end{cases}$$

В силу предложения 43, функция $f(x)$ вычислима. По теореме о неподвижной точке существует $n \in \omega$ такое, что $\varphi_{f(n)} = \varphi_n$. Тогда имеет место следующая цепочка эквивалентностей (вторая эквивалентность следует из выбора a и b, а третья — из определения функции $f(x)$):

$$\varphi_n \in S \iff \varphi_{f(n)} \in S \iff f(n) = a \iff n \notin A \iff \varphi_n \notin S.$$

Противоречие. Следовательно, сделанное нами предположение о вычислимости множества $\{x \in \omega \mid \varphi_x \in S\}$ неверно. □

Пример. Пусть $f(x)$ — произвольная одноместная ч.в.ф. Рассмотрим одноэлементное семейство функций $S = \{f\}$. Оно удовлетворяет условиям теоремы Райса, следовательно, множество $A = \{e \in \omega \mid \varphi_e = f\}$ не вычислимо. Заметим, что множество A — это в точности множество всех клиниевских номеров функции f. Таким образом, из теоремы Райса следует, что *множество всех клиниевских номеров фиксированной ч.в.ф. не вычислимо*. В частности, любая ч.в.ф. имеет бесконечно много клиниевских номеров (поскольку, очевидно, любое конечное множество вычислимо).

Упражнения

1. Доказать, что существует $a \in \omega$ такое, что $\varphi_a(x) = \chi_{\{a\}}(x)$.
2. Доказать, что существует $a \in \omega$ такое, что $\varphi_a(a^{2019}) = a^{2018}$.
3. Доказать, что существуют $a, b \in \omega$ такие, что b делит a и $\varphi_a(b) = a/b$.
4. Доказать, что существуют $a, b \in \omega$ такие, что $\varphi_a(b) = a + b$ и $\varphi_b(a) = a \cdot b$.

5. Доказать, что существует $a \in \omega$ такое, что $\mathrm{dom}(\varphi_a) = \{0, a, 2a, 3a, \dots\}$.
6. Доказать, что существует $a \in \omega$ такое, что $\mathrm{dom}(\varphi_a) = \{1, a, a^2, a^3, \dots\}$.
7. Доказать, что множество $\{x \in \omega \mid \varphi_x$ — константа$\}$ не является вычислимым.
8. Доказать, что множество $\{\langle x, y\rangle \in \omega^2 \mid \varphi_x = \varphi_y\}$ не является вычислимым.
9. Доказать, что множество $\{\langle x, y\rangle \in \omega^2 \mid \mathrm{dom}(\varphi_x) = \omega \setminus \mathrm{dom}(\varphi_y)\}$ невычислимо.

§ 24. Вычислимо перечислимые множества

В этом параграфе мы введём понятие вычислимо перечислимого множества (сокращённо в.п. множества) и изучим некоторые свойства таких множеств. С интуитивной точки зрения множество является вычислимо перечислимым, если существует алгоритм эффективного перечисления всех его элементов. При этом мы допускаем, что это перечисление может иметь повторения и не обязано быть перечислением в каком-то строго определённом порядке.

Определение. Пусть $k \geqslant 1$. Множество $A \subseteq \omega^k$ называется *вычислимо перечислимым* (*в.п.*), если $A = \varnothing$ или $A = \{\langle f_1(x), \dots, f_k(x)\rangle \mid x \in \omega\}$ для некоторых вычислимых функций $f_1(x), \dots, f_k(x)$.

Другими словами, вычислимые функции $f_1, \dots, f_k$ покоординатно перечисляют множество A. В случае $k = 1$ определение выглядит проще: множество $A \subseteq \omega$ является в.п. тогда и только тогда, когда $A = \varnothing$ или $A = \mathrm{range}(f)$ для некоторой вычислимой функции $f(x)$. Введённое определение является формальным описанием интуитивного понятия перечислимости. Однако существует несколько эквивалентных описаний в.п. множеств, каждое из которых оказывается полезным при изучении тех или иных свойств в.п. множеств.

Теорема 66 (об эквивалентных определениях в.п. множеств)**.**
Для произвольного множества $A \subseteq \omega^k$ следующие условия эквивалентны:

(1) A вычислимо перечислимо.

(2) Существует вычислимое отношение $R \subseteq \omega^{k+1}$ такое, что

$$\langle x_1, \dots, x_k\rangle \in A \iff \exists y R(x_1, \dots, x_k, y).$$

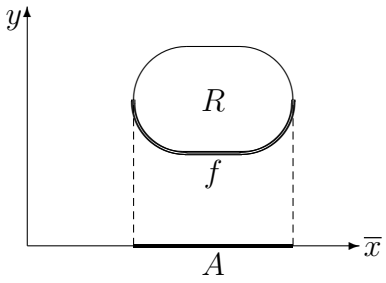
(3) Существует ч.в.ф. $f(x_1, \dots, x_k)$ такая, что $A = \mathrm{dom}(f)$.

Доказательство. Докажем справедливость импликации (1) $\Rightarrow$ (2). Если $A = \varnothing$, то вычислимое множество $R = \varnothing$, очевидно, удовлетворяет условию (2). Если же $A \neq \varnothing$ и $A = \{\langle f_1(x), \dots, f_k(x)\rangle \mid x \in \omega\}$, где $f_1, \dots, f_k$ — вычислимые функции, то имеет место эквивалентность

$$\langle x_1, \dots, x_k\rangle \in A \iff \exists y\big(f_1(y) = x_1 \,\&\, \dots \,\&\, f_k(y) = x_k\big).$$

Тогда множество $R = \{\langle x_1, \dots, x_k, y\rangle \mid f_1(y) = x_1 \,\&\, \dots \,\&\, f_k(y) = x_k\}$ является искомым $(k{+}1)$-местным вычислимым отношением.

Теперь докажем импликацию (2) $\Rightarrow$ (3). Пусть $R \subseteq \omega^{k+1}$ — вычислимое отношение такое, что A является его проекцией (см. рисунок), т. е. имеет место эквивалентность $\overline{x} \in A \iff \exists y R(\overline{x}, y)$.



Определим частичную k-местную функцию $f(\overline{x}) = \mu y[R(\overline{x}, y)]$. Так как R вычислимо, то $f(\overline{x})$ — ч.в.ф. Кроме этого, имеет место

$$f(\overline{x})\downarrow \iff \exists y R(\overline{x}, y) \iff \overline{x} \in A.$$

Другими словами, $\mathrm{dom}(f) = A$.

Докажем импликацию (3) $\Rightarrow$ (1). Если $A = \varnothing$, то доказывать нечего. Пусть $A \neq \varnothing$. Следовательно, найдётся кортеж $\overline{a} = \langle a_1, \ldots, a_k\rangle \in A$. По условию $A = \mathrm{dom}(f)$ для некоторой ч.в.ф. f. Тогда f вычислима на некоторой машине Тьюринга с кодом e и по теореме об универсальной ч.р.ф.

$$f(\overline{x}) = \mathrm{out}(\mathrm{run}(e, \mathrm{in}^k(\overline{x}), \mu y[q(e, \overline{x}, y) = 0])).$$

Для каждого $i \in \{1, \ldots, k\}$ определим 1-местную вычислимую функцию

$$f_i(n) = \begin{cases} \mathrm{ex}(i, n), & \text{если } q(e, \mathrm{ex}(1, n), \ldots, \mathrm{ex}(k, n), \mathrm{ex}(0, n)) = 0, \\ a_i, & \text{если } q(e, \mathrm{ex}(1, n), \ldots, \mathrm{ex}(k, n), \mathrm{ex}(0, n)) \neq 0. \end{cases}$$

В силу предложения 43, функции $f_1, \ldots, f_k$ вычислимы.

Покажем, что набор функций $f_1, \ldots, f_k$ — искомый, т. е. $A = \{\langle f_1(x), \ldots, f_k(x)\rangle \mid x \in \omega\}$. Для этого докажем сначала включение $A \subseteq \{\langle f_1(x), \ldots, f_k(x)\rangle \mid x \in \omega\}$. Пусть $\overline{x} = \langle x_1, \ldots, x_k\rangle \in A$. Тогда $f(\overline{x}) = \mathrm{out}(\mathrm{run}(e, \mathrm{in}^k(\overline{x}), \mu y[q(e, \overline{x}, y) = 0]))$ определено. Таким образом, значение $\mu y[q(e, \overline{x}, y) = 0]$ определено. Следовательно, существует $y \in \omega$ такой, что $q(e, \overline{x}, y) = 0$. Положим $n = p_0^y \cdot p_1^{x_1} \cdot \ldots \cdot p_k^{x_k}$. Тогда $q(e, \mathrm{ex}(1, n), \ldots, \mathrm{ex}(k, n), \mathrm{ex}(0, n)) = 0$, и значит, $f_i(n) = \mathrm{ex}(i, n) = x_i$ для каждого $i \in \{1, \ldots, k\}$. Таким образом, $\overline{x} \in \{\langle f_1(x), \ldots, f_k(x)\rangle \mid x \in \omega\}$.

Докажем обратное включение $\{\langle f_1(x), \ldots, f_k(x)\rangle \mid x \in \omega\} \subseteq A$. Рассмотрим произвольный набор $\langle f_1(n), \ldots, f_k(n)\rangle$, где $n \in \omega$. Если $q(e, \mathrm{ex}(1, n), \ldots, \mathrm{ex}(k, n), \mathrm{ex}(0, n)) \neq 0$, то $\langle f_1(n), \ldots, f_k(n)\rangle = \overline{a} \in A$ и всё доказано. Пусть теперь справедливо равенство $q(e, \mathrm{ex}(1, n), \ldots, \mathrm{ex}(k, n), \mathrm{ex}(0, n)) = 0$. Тогда, в силу предложения 51, машина с кодом e, начав работу на входном машинном слове $q_1 01^{\mathrm{ex}(1,n)} 0 \ldots 01^{\mathrm{ex}(k,n)} 0$ и проделав $\mathrm{ex}(0, n)$ шагов, переходит в состояние q_0. Так как данная машина вычисляет функцию f, заключаем, что значение $f(\mathrm{ex}(1, n), \ldots, \mathrm{ex}(k, n))$ определено. Следовательно, кортеж $\langle f_1(n), \ldots, f_k(n)\rangle = \langle \mathrm{ex}(1, n), \ldots, \mathrm{ex}(k, n)\rangle \in \mathrm{dom}(f) = A$. □

Выясним теперь, как соотносятся между собой семейство всех в.п. множеств и семейство всех вычислимых множеств.

Предложение 67. *Если $A \subseteq \omega^k$ вычислимо, то A вычислимо перечислимо.*

Доказательство. Пусть A вычислимо, следовательно, функция $\chi_A(x_1, \ldots, x_k)$ вычислима. Определим частичную функцию $f(\overline{x}) = \mu y[|\chi_A(\overline{x}) - 1| = 0]$. Тогда f — ч.в.ф. и $\mathrm{dom}(f) = A$. Следовательно, в силу пункта (3) теоремы 66 A является в.п. множеством. □

Утверждение, обратное к предложению 67, вообще говоря, неверно.

Предложение 68. *Множество* $K = \{x \in \omega \mid \varphi_x(x)\downarrow\}$ *является вычислимо перечислимым, но не является вычислимым.*

Доказательство. Докажем, что K вычислимо перечислимо. По теореме об универсальной ч.р.ф. $\varphi_x(y)$ является двухместной ч.в.ф. Тогда функция $f(x) = \varphi_x(x)$ — одноместная ч.в.ф. Ясно, что $K = \mathrm{dom}(f)$. Отсюда по пункту (3) теоремы 66 заключаем, что K вычислимо перечислимо.

Докажем, что K не вычислимо. Допустим, напротив, характеристическая функция $\chi_K(x)$ вычислима. Рассмотрим одноместную функцию

$$f(x) = \begin{cases} 0, & \text{если } \varphi_x(x)\uparrow \\ \text{не определено}, & \text{если } \varphi_x(x)\downarrow . \end{cases}$$

Так как $f(x) = \mu y[\chi_K(x) = 0]$, то $f(x)$ частично вычислима. По теореме об универсальной ч.р.ф. существует клиниевский номер $n \in \omega$ такой, что $f(x) = \varphi_n(x)$. Рассмотрим значение аргумента $x = n$, получим следующую цепочку эквивалентных условий:

$$\varphi_n(n)\downarrow \iff f(n)\downarrow \iff \varphi_n(n)\uparrow .$$

Противоречие. Следовательно, K не вычислимо. □

Определение. Множество $K = \{x \in \omega \mid \varphi_x(x)\downarrow\}$ из предложения 68 называется *креативным.*

Далее мы исследуем свойства замкнутости семейства в.п. множеств относительно теоретико-множественных операций. Напомним, что для вычислимых множеств справедливо следующее предложение.

Предложение 69. *Пусть* $A, B \subseteq \omega^k$ *— вычислимые множества. Тогда множества* $A \cup B$, $A \cap B$ *и* $\omega^k \setminus A$ *тоже вычислимы.*

Доказательство. См. доказательство предложения 40. □

Для семейства в.п. множеств замкнутость относительно объединения и пересечения остаётся справедливой. Однако, в отличие от вычислимых множеств, в.п. множества незамкнуты относительно дополнения.

Предложение 70. *Пусть* $A, B \subseteq \omega^k$ *— вычислимо перечислимые множества. Тогда множества* $A \cup B$ *и* $A \cap B$ *тоже вычислимо перечислимы.*

Доказательство. Пусть $P, R \subseteq \omega^{k+1}$ такие вычислимые множества, что

$$\overline{x} \in A \iff \exists y P(\overline{x}, y),$$
$$\overline{x} \in B \iff \exists y R(\overline{x}, y).$$

Тогда для объединения получаем

$$\overline{x} \in A \cup B \iff \Big(\exists y P(\overline{x}, y) \vee \exists y R(\overline{x}, y)\Big) \iff \exists y\Big(P(\overline{x}, y) \vee R(\overline{x}, y)\Big) \iff \exists y Q(\overline{x}, y),$$

где $Q(\overline{x}, y) = P(\overline{x}, y) \vee R(\overline{x}, y)$ — вычислимый предикат. Следовательно, $A \cup B$ вычислимо перечислимо.

Для пересечения имеем

$$\overline{x} \in A \cap B \iff \Big(\exists y P(\overline{x}, y) \;\&\; \exists z R(\overline{x}, z)\Big) \iff \exists y \exists z \Big(P(\overline{x}, y) \;\&\; R(\overline{x}, z)\Big) \iff$$
$$\iff \exists t \Big(P(\overline{x}, \mathrm{ex}(0,t)) \;\&\; R(\overline{x}, \mathrm{ex}(1,t))\Big) \iff \exists t Q(\overline{x}, t),$$

где $Q(\overline{x}, t) = P(\overline{x}, \mathrm{ex}(0,t)) \;\&\; R(\overline{x}, \mathrm{ex}(1,t))$ — вычислимый предикат. Следовательно, $A \cap B$ вычислимо перечислимо. □

Теорема 71 (теорема Поста). *Множество $A \subseteq \omega^k$ вычислимо тогда и только тогда, когда A и $\omega^k \setminus A$ вычислимо перечислимы.*

Доказательство. ($\Longrightarrow$) Следует из замкнутости вычислимых множеств относительно дополнения и предложения 67.

($\Longleftarrow$) Пусть $P, R \subseteq \omega^{k+1}$ такие вычислимые множества, что

$$\overline{x} \in A \iff \exists y P(\overline{x}, y),$$
$$\overline{x} \notin A \iff \exists y R(\overline{x}, y).$$

Определим вычислимую функцию $f(\overline{x}) = \mu y[P(\overline{x}, y) \vee R(\overline{x}, y)]$. Тогда получаем: $\overline{x} \in A \iff \exists y P(\overline{x}, y) \;\&\; \forall y \neg R(\overline{x}, y) \iff P(\overline{x}, f(\overline{x}))$. Поэтому $\chi_A(\overline{x}) = \chi_P(\overline{x}, f(\overline{x}))$ является вычислимой функцией. Таким образом, A вычислимо. □

Следствие 72. *Существует множество $A \subseteq \omega$ такое, что A вычислимо перечислимо, но $\omega \setminus A$ не является вычислимо перечислимым.*

Доказательство. Рассмотрим креативное множество $K = \{x \in \omega \mid \varphi_x(x) \downarrow\}$. По предложению 68 множество K — в.п. Если бы $\omega \setminus K$ было в.п., то по теореме Поста K было бы вычислимым, что невозможно. □

Теорема 73 (теорема о графике). *Частичная функция $f(x_1, \ldots, x_k)$ является частично вычислимой тогда и только тогда, когда её график*

$$\Gamma_f = \{\langle x_1, \ldots, x_k, y\rangle \mid \langle x_1, \ldots, x_k\rangle \in \mathrm{dom}(f), f(x_1, \ldots, x_k) = y\}$$

является вычислимо перечислимым.

Доказательство. ($\Longrightarrow$) Пусть f — ч.в.ф. По теореме об универсальной ч.р.ф.

$$f(\overline{x}) = \mathrm{out}(\mathrm{run}(e, \mathrm{in}^k(\overline{x}), \mu y[q(e, \overline{x}, y) = 0])),$$

где e — код машины Тьюринга, вычисляющей f. Тогда получаем

$$\langle \overline{x}, y\rangle \in \Gamma_f \iff f(\overline{x}) \downarrow = y \iff$$
$$\iff \exists z \big(q(e, \overline{x}, z) = 0 \;\&\; \mathrm{out}(\mathrm{run}(e, \mathrm{in}^k(\overline{x}), z)) = y\big) \iff \exists z R(\overline{x}, y, z),$$

где $R = \{\langle \overline{x}, y, z\rangle \mid q(e, \overline{x}, z) = 0 \;\&\; \mathrm{out}(\mathrm{run}(e, \mathrm{in}^k(\overline{x}), z)) = y\}$. Поскольку отношение R является вычислимым, то в силу пункта (2) теоремы 66 заключаем, что Γ_f — в.п.

($\Longleftarrow$) Пусть Γ_f — в.п. В соответствии с пунктом (2) теоремы 66 существует вычислимое отношение $R \subseteq \omega^{k+2}$ такое, что справедлива эквивалентность $\langle \overline{x}, y\rangle \in \Gamma_f \iff$ $\iff \exists z R(\overline{x}, y, z)$. Отсюда заключаем:

$$\overline{x} \in \mathrm{dom}(f) \iff \exists y \langle \overline{x}, y\rangle \in \Gamma_f \iff \exists y \exists z R(\overline{x}, y, z) \iff$$
$$\iff \exists t R(\overline{x}, \mathrm{ex}(0,t), \mathrm{ex}(1,t)) \iff \mu t[R(\overline{x}, \mathrm{ex}(0,t), \mathrm{ex}(1,t))] \text{ определено.}$$

Поэтому $f(\overline{x}) = \mathrm{ex}\big(0, \mu t[R(\overline{x}, \mathrm{ex}(0,t), \mathrm{ex}(1,t))]\big)$ является ч.в.ф. □

Упражнения

1. Доказать, что множество $\{x \in \omega \mid 0 \in \mathrm{range}(\varphi_x)\}$ вычислимо перечислимо.

2. Доказать, что множество $\{x \in \omega \mid \varphi_x$ и φ_{x+1} определены в нуле$\}$ является вычислимо перечислимым.

3. Доказать, что множество $\{x \in \omega \mid \mathrm{dom}(\varphi_x) \neq \varnothing\}$ является вычислимо перечислимым, но не является вычислимым.

4. Пусть A — вычислимо перечислимое подмножество ω^2, B — вычислимо перечислимое подмножество ω. Обозначим $A_x = \{y \in \omega \mid \langle x, y\rangle \in A\}$. Доказать, что множество $\bigcup\limits_{x \in B} A_x$ вычислимо перечислимо.

5. Доказать, что множество $A \subseteq \omega^n$ вычислимо перечислимо тогда и только тогда, когда его *частичная характеристическая функция*

$$\chi_A^*(x_1, \ldots, x_n) = \begin{cases} 1, & \langle x_1, \ldots, x_n\rangle \in A, \\ \text{не определено}, & \langle x_1, \ldots, x_n\rangle \notin A, \end{cases}$$

является частично вычислимой.

6. Доказать, что множество $A \subseteq \omega$ вычислимо перечислимо тогда и только тогда, когда существует ч.в.ф. $f(x)$ такая, что $A = \mathrm{range}(f)$.

7. Доказать, что бесконечное множество $A \subseteq \omega$ является вычислимо перечислимым тогда и только тогда, когда $A = \mathrm{range}(f)$ для некоторой вычислимой инъективной функции $f(x)$.

8. Доказать, что любое бесконечное в.п. множество $A \subseteq \omega$ содержит бесконечное вычислимое подмножество.

9. Пусть A — вычислимое подмножество ω, а $f(x)$ — вычислимая функция, такие что $\mathrm{range}(f) = \omega$ и $f(A) \cap f(\omega \setminus A) = \varnothing$. Доказать, что множество $f(A)$ вычислимо.

10. Пусть $A, B \subseteq \omega$ такие, что множество $(A \setminus B) \cup (B \setminus A)$ конечно. Доказать, что A вычислимо перечислимо тогда и только тогда, когда B вычислимо перечислимо.

11. Вычислимо перечислимое множество $A \subseteq \omega$ называется *простым*, если его дополнение $\omega \setminus A$ бесконечно и не содержит бесконечных вычислимо перечислимых подмножеств. Доказать, что если A, B — простые, то $A \cap B$ — простое.

12. Пусть $f_1, \ldots, f_k$ — n-местные ч.в.ф., а $A_1, \ldots, A_k$ — попарно непересекающиеся в.п. подмножества ω^n. Доказать, что следующая функция частично вычислима:

$$g(x_1, \ldots, x_n) = \begin{cases} f_1(x_1, \ldots, x_n), & \text{если } \langle x_1, \ldots, x_n\rangle \in A_1, \\ \ldots & \ldots \\ f_k(x_1, \ldots, x_n), & \text{если } \langle x_1, \ldots, x_n\rangle \in A_k, \\ \text{не определено}, & \text{иначе.} \end{cases}$$

13. Какова мощность семейства всех вычислимо перечислимым, но не вычислимых подмножеств ω?

Список литературы

1. *Дасгупта С., Пападимитриу Х., Вазирани У.* Алгоритмы. М.: МЦНМО, 2014.
2. *Ершов Ю. Л.* Теория нумераций. М.: Наука, 1977.
3. *Ершов Ю. Л., Палютин Е. А.* Математическая логика. СПб.: Лань, 2004.
4. *Йех Т.* Теория множеств и метод форсинга. М.: Мир, 1973.
5. *Касьянов В. Н.* Лекции по теории формальных языков, автоматов и сложности вычислений. Новосибирск: НГУ, 1995.
6. *Когабаев Н. Т.* Лекции по теории алгоритмов. Новосибирск: НГУ, 2009.
7. *Кормен Т. Х., Лейзерсон Ч. И., Ривест Р. Л., Штайн К.* Алгоритмы: построение и анализ. 2-е изд. М.: Вильямс, 2005.
8. *Лавров И. А., Максимова Л. Л.* Задачи по теории множеств, математической логике и теории алгоритмов. М.: Наука, 1984.
9. *Мальцев А. И.* Алгоритмы и рекурсивные функции. М.: Наука, 1986.
10. *Мендельсон Э.* Введение в математическую логику. М.: Наука, 1971.
11. *Морозов А. С.* Машины Шёнфилда: метод. указания. Новосибирск: НГУ, 1996.
12. *Роджерс Х.* Теория рекурсивных функций и эффективная вычислимость. М.: Мир, 1972.
13. *Соар Р. И.* Вычислимо перечислимые множества и степени. Казань: Казанское мат. общ-во, 2000.
14. *Khoussainov B., Nerode A.* Automata Theory and Its Applications. Boston: Birkhauser, 2001.
15. *Lewis H. R., Papadimitriou C. H.* Elements of the Theory of Computation. N. J.: Plentice Hall, 1998.
16. *Shoenfield J. R.* Recursion Theory, Lecture Notes in Logic. Berlin: Springer-Verlag, 1993.
17. *Morphett A.* A Web-based Turing Machine Simulator, `http://morphett.info/turing/`

Учебное издание

Когабаев Нурлан Талгатович

ДИСКРЕТНАЯ МАТЕМАТИКА И ТЕОРИЯ АЛГОРИТМОВ

Учебное пособие

Редактор *Д. И. Ковалева*

Подписано в печать 01.03.2023 г.
Формат 60×84 1/8. Уч.-изд. л. 15,75. Усл. печ. л. 14,6.
Тираж 100 экз. Заказ № 20

Издательско-полиграфический центр НГУ
630090, Новосибирск, ул. Пирогова, 2.