Вычислимые функции

Ю.Ю. Кочетков

1. Машина с неограниченными регистрами

В этом пособии будут рассматриваться вопросы, связанные с понятием вычислимости: вычислимые функции и сложность их вычисления. Понятие вычислимости требует точного определения. Хотя таких определений предложено довольно много, в действительности, они эквивалентны. Мы рассмотрим три из них: определение, использующее машину с неограниченными регистрами, определение, использующее рекурсивные функции и определение, использующее машину Тьюринга. Первое определение самое простое. С него мы и начнем.

Наш основной объект — это функция $f(x_1, \ldots, x_n)$ от переменных $x_1, \ldots, x_n \in \mathbb{Z}_{\geqslant 0}$, где $\mathbb{Z}_{\geqslant 0}$ — это множество целых неотрицательных чисел. Значения функции f — это тоже целые неотрицательные числа. Функцию f мы будем называть *тотальной*, если она определена при всех значениях x_i, $i = 1, \ldots, n$. В противном случае мы будем называть f *частичной*. Функцию от n переменных мы будем называть *n-местной*.

Вычислимая функция — это функция, вычислимая некоторой программой (алгоритмом). Программы написаны для идеализированной вычислительной машины, которую мы будем называть *машиной с неограниченными регистрами* — такая машина имеет бесконечно много регистров памяти R_1, R_2, ..., а в каждом регистре может храниться сколь угодно большое целое неотрицательное число. Программа состоит из команд четырех типов.

- **Обнуление.** Команда Z(n): в регистр R_n заносится число 0, содержимое остальных регистров не меняется.
- **Прибавление единицы.** Команда S(n): содержимое регистра R_n увеличивается на единицу, остальные регистры не меняются.
- **Переадресация.** Команда T(m,n): в регистр R_n заносится содержимое регистра R_m, при этом остальные регистры (включая регистр R_m) не затрагиваются этой командой.
- **Условный переход.** Команда J(m,n,q): если содержимое регистра R_m равно содержимому регистра R_n, то программа переходит к выполнению команды с номером q, в противном случае выполняется следующая по номеру команда.

Команды данной программы перенумерованы. Выполнение программы начинается с первой команды. После выполнения очередной команды, если это не есть команда условного перехода, выполняется следующая по номеру команда. Программа останавливается, если

(1) при переходе к следующей команде выясняется, что следующей команды нет;
(2) при выполнении команды условного перехода J(m,n,q) к команде с номером q выясняется, что команды с номером q нет.

Определение 1.1. Функция $f(x_1, \ldots, x_n)$ называется вычислимой если существует программа P такая, что:

- если набор $(a_1, \ldots, a_n)$ принадлежит области определения функции f, то при начальной конфигурации регистров $(a_1, \ldots, a_n, 0, \ldots)$ программа P завершает работу, и в первом регистре находится число $f(a_1, \ldots, a_n)$;
- если набор $(b_1, \ldots, b_n)$ не принадлежит области определения функции f, то при начальной конфигурации регистров $(b_1, \ldots, b_n, 0, \ldots)$ программа P не останавливается.

Пример. Программа, состоящая из одной команды $J(1, 1, 1)$, задает вычислимую функцию с пустой областью определения.

Пример. Вот программа, задающая вычислимую тотальную двуместную функцию $f(x, y) = x + y$:

1. $J(2, 3, 5)$
2. $S(1)$
3. $S(3)$
4. $J(1, 1, 1)$

Программа работает так: начальная конфигурация — это $(x, y, 0, \ldots)$. Далее мы увеличиваем на единицу регистр R_1 и регистр R_3, до тех пор пока, содержимое регистра R_3 не сравняется с содержимым регистра R_2. В этот момент в R_1 будет лежать сумма $x + y$.

Для дальнейшего нам понадобится программа, вычисляющая умножение $f(x, y) = xy$. Вот эта программа для случая ненулевых чисел x и y.

Пример. Мы будем y раз складывать x с собой.

(1) T(1,5) (копируем первый регистр в пятый);

(2) S(3) (третий регистр — счетчик числа сложений);
(3) J(2,3,9) (третий регистр сравнялся со вторым — останов);
(4) Z(4) (начинается цикл сложения);
(5) J(4,5,2) (выход из цикла сложения);
(6) S(1);
(7) S(4);
(8) J(1,1,5) (возвращение к началу цикла сложения).

2. Пример невычислимой функции

Привести пример невычислимой функции не так-то просто.

Рассмотрим множество A_n всех программ, состоящих из n команд и работающих только с n первыми регистрами, причем условный переход может быть на команду с номером от 1 до $n+1$. Это множество конечно, хотя и очень велико. В самом деле, что можно сказать о размере множества A_2? Вот список команд, работающих с первыми двумя регистрами: $Z(i)$, $i = 1, 2$ (2 команды); $S(i)$, $i = 1, 2$ (2 команды); $T(i, j)$, $i, j = 1, 2$ (4 команды); $J(i, j, k)$, $i, j = 1, 2$, $k = 1, 2, 3$ (12 команд). Итого, 20 команд и 400 программ.

Рассмотрим подмножество $B_n \subset A_n$, состоящее из тех программ, которые останавливаются при начальной нулевой конфигурации (т.е. когда во всех регистрах находятся нули). Пусть $P \in B_n$. Запустим P при начальной нулевой конфигурации и дождемся остановки программы. Через a_P обозначим число в регистре R_1. Теперь мы можем определить функцию

$$f(n) = \max_{P \in B_n} a_P.$$

Замечание. Функция $f(n)$ быстро растет с ростом n. Например, можно предложить такую программу. Сначала $S(1)$ пять раз, потом $T(1, 2)$, потом программа умножения (8 команд). Следовательно, $f(15) \geqslant 25$. Немного усложнив программу, мы можем получить 5^{5^5}.

Теорема 2.1. *Функция $f(n)$ невычислима.*

Доказательство. Во-первых функция f строго монотонна: $f(n) < f(n+1)$. Действительно, пусть $P \in B_n$ — программа, для которой $a_P = f(n)$. Рассмотрим программу $\tilde{P}$ — это программа P плюс еще одна команда $S(1)$. Тогда $\tilde{P} \in B_{n+1}$ и $f(n+1) \geqslant a_{\tilde{P}} = a_P + 1 = f(n) + 1$.

Предположим, что есть программа П из m команд, вычисляющая функцию f, т.е. при начальной конфигурации $(n, 0, \ldots)$ в результате работы программы в регистре R_1 будет $f(n)$. Рассмотрим следующую программу Q: сначала k раз команда $S(1)$, затем команда $T(1, 2)$, затем умножение (8 команд), затем обнуление регистров R_2, R_3, R_4 и R_5, затем программа П. В этой программе $n = m + k + 13$ команд, и в результате ее работы в регистре R_1 будет число $a_Q = f(k^2)$. Следовательно,

$$f(k^2) = a_Q \leqslant \max_{P \in B_n} a_P = f(n) = f(m + k + 13).$$

Но это неравенство противоречит строгой монотонности функции f, потому что $k^2 > m + k + 13$ при достаточно большом k. □

Обсуждение. В программировании есть понятие некорректной программы — это программа, которая зацикливается и не останавливается. Только что доказанная теорема в сущности означает, что не существует алгоритма проверки корректности программ. Действительно, пусть такой алгоритм есть. Тогда можно написать программу, вычисляющую функцию f. Эта программа работает так. Начальная конфигурация — это $(n, 0, \ldots)$. Первый блок программы — это генератор всех программ длины n с указанными выше свойствами. Сгенерированная программа P подается на вход алгоритма проверки корректности. Если она корректна, то запускаем ее на нулевой начальной конфигурации и находим число a_P. Далее берем максимальное из этих чисел. Но функция f невычислима, следовательно, алгоритма проверки корректности не существует.

Упражнения

(1) Написать программы, вычисляющие частичные функции: а) $f(x, y) = x - y$; б) $f(x) = x/2$; в) $f(x) = \log_2 x$.
(2) Написать программу, вычисляющую функцию $f(x) = 2^x$.
(3) Написать программу, вычисляющую функцию $f(x) = x \mod 3$.

3. Нумерация

В этом разделе мы определим нумерацию команде и программ, причем эта нумерация будет эффективной: по программе (команде) нетрудно найти ее номер и по номеру — программу (команду).

Первый шаг — нумерация пар. Рассмотрим множество все пар положительных целых чисел $A = \{(k, l) : k, l \in \mathbb{N}\}$. Каждой такой паре мы сопоставим число – ее номер $N_2(k, l) = 2^{k-1}(2l - 1)$. Убедимся в том, что эта нумерация корректна. Во-первых, номер — это целое положительное число. Во-вторых, если $N_2(k, l) =$

$N_2(m,n)$, то оба этих числа делятся на одну и ту же степень двойки, следовательно, $2^{k-1} = 2^{m-1}$, т.е. $k = m$. Теперь поделив на эту степень двойки, получаем, что $2l-1 = 2n-1$, т.е. $l = n$. Если q — некоторое натуральное число, то его можно единственным образом представить в виде $q = 2^s r$, где r — нечетно. Тогда $q = N_2(s+1,(r+1)/2)$.

Пример. $N_2(6,3) = 2^5 \cdot 5 = 180$. Пусть $q = 100 = 2^2 \cdot 25$. Тогда $q = N_2(3,13)$. Полезно отметить, что номер 1 имеет пара $(1,1)$.

Второй шаг — нумерация троек. Рассмотрим множество всех троек положительных целых чисел $B = \{(k,l,m) : k,l,m \in \mathbb{N}\}$. Каждой такой тройке сопоставим число — ее номер $N_3(k,l,m) = N_2(k,N_2(l,m)) = 2^{k-1}(2N_2(l,m)-1) = 2^{k-1}(2^l(2m-1)-1))$. Нетрудно убедиться, что эта нумерация корректна. При этом номер 1 имеет тройка $(1,1,1)$.

Третий шаг — нумерация команд. Команды вида $Z(n)$ будут нумероваться числами, делящимися на 4, команды вида $S(n)$ — числами, имеющими остаток 1 от деления на 4, команды вида $T(m,n)$ — числами, имеющими остаток 2 от деления на 4 и команды вида $J(k,l,m)$ — числами, имеющими остаток 3 от деления на 4. А именно,

$$\begin{aligned} N_c(Z(n)) &= 4n, \\ N_c(S(n)) &= 4(n-1)+1, \\ N_c(T(m,n)) &= 4(N_2(m,n)-1)+2, \\ N_c(J(k,l,m)) &= 4(N_3(k,l,m)-1)+3. \end{aligned}$$

Пример. $N_c(T(3,5)) = 4(N_2(3,5)-1)+2 = 4 \cdot (36-1)+2 = 142$. Какая команда имеет номер 51? Так как $51 = 4 \cdot 12 + 3$, то это команда условного перехода и нам нужно найти тройку с номером 13. Имеем $13 = N_2(1,7) = N_2(1,N_2(1,4)) = N_3(1,1,4)$. Следовательно, 51 — это номер команды $J(1,1,4)$.

Четвертый шаг — нумерация конечных последовательностей. Пусть $(a_1,\ldots,a_n)$ — конечная последовательность натуральных чисел. Нумеровать последовательности так, как мы нумеровали пары или тройки чисел, нельзя, потому что число членов последовательности — переменная величина. Мы поступим по другому. Сначала, мы сопоставим последовательности $(a_1,\ldots,a_n)$ строго возрастающую последовательность $(b_1,\ldots,b_n)$, где $b_1 = a_1$, $b_2 = a_1 + a_2$, ..., $b_n = a_1 + \cdots + a_n$. А последовательности $(b_1,\ldots,b_n)$ мы сопоставим число $2^{b_1-1} + 2^{b_2-1} + \cdots + 2^{b_n-1}$, которое и является номером $N_s(a_1,\ldots,a_n)$ последовательности $(a_1,\ldots,a_n)$. Корректность нумерации следует из однозначности представления натурального числа в двоичной системе счисления.

Замечание. Если последовательность $(a_1,\ldots,a_n)$ является возрастающей, то все равно сначала нужно перейти к последовательности $(b_1,\ldots,b_n)$.

Пример. Рассмотрим последовательность $c = (3,1,4,2,1)$. Превратим ее в строго возрастающую $(3,4,8,10,11)$. Тогда

$$N_s(c) = 2^2 + 2^3 + 2^7 + 2^9 + 2^{10} = 4 + 8 + 128 + 512 + 1024 = 1676.$$

Найдем последовательность с номером 75. Имеем,

$$75 = 64 + 8 + 2 + 1 = 2^0 + 2^1 + 2^3 + 2^6.$$

Следовательно, строго возрастающая последовательность — это $(1,2,4,7)$. Значит, $75 = N_s(1,1,2,3)$.

Пятый шаг — нумерация программ. Теперь уже просто. Программа P — это конечная последовательность команд, у каждой команды есть номер, т.е. программе мы сопоставляем конечную последовательность c номеров команд и полагаем $N_p(P) = N_s(c)$.

Пример. Найдем программу с номером $19 = 2^0 + 2^1 + 2^4$. Этот номер имеет последовательность $(1,1,3)$. Соответствующая программа — это

$$\begin{aligned} &1.S(1) \\ &2.S(1) \\ &3.J(1,1,1) \end{aligned}$$

Следует иметь в виду, что программы с маленькими номерами довольно бессмысленные. Номер же самой короткой осмысленной программы — программы сложения двух чисел, огромен.

$$\begin{aligned} &1.J(2,3,5) \\ &2.S(1) \\ &3.S(3) \\ &4.J(1,1,1) \end{aligned}$$

Так как $N_3(2,3,5) = N_2(2,N_2(3,5)) = N_2(2,36) = 142$, то $N_c(J(2,3,5)) = 4 \cdot 141 + 3 = 567$. Программе сопоставлена последовательность номеров команд $c = (567,1,9,3)$. Номер программы равен числу

$$N_s(c) = 2^{566} + 2^{567} + 2^{576} + 2^{579} \approx 2.22 \cdot 10^{174}.$$

Упражнения.

(1) Какой номер имеет программа S(1), S(1),..., S(1) (n раз)?
(2) Найти программу с номером 13056.

4. Универсальная функция

Напомним, что функция и программа, которая ее вычисляет, — это разные вещи. Очевидно, что существует бесконечно много программ, вычисляющих данную функцию. Мы будем говорить о функциях, хотя правильнее было говорить об программах. Пока ограничимся одноместными функциями. Через $\varphi_n(x)$ мы будем обозначать функцию (программу) с номером n. На самом деле каждая программа вычисляет функцию от любого числа переменных: помещаем значения аргументов в первые регистры и запускаем программу.

Определение 4.1. Универсальной функцией для одноместных вычислимых функций называется двуместная функция Φ, для которой $\Phi(k,l) = \varphi_k(l)$.

Предложение 4.1. *Функция Φ вычислима.*

Объяснение. Для доказательства вычислимости функции Φ нужно написать программу, которая ее вычисляет. Но такая программа очень велика, хотя было бы любопытно ее написать. Я постараюсь объяснить, почему такая программа есть.

Эта программа должна работать так. Она состоит из двух модулей — модуля построения программы по ее номеру и модуля, который моделирует работу машины. Будем считать, что второй модуль работает с регистрами $R_4, \ldots, R_s$. Первый модуль строит двоичное представление числа k, перемещает число l в первый регистр, в регистрах, начиная с $(s+1)$-го записывает номера команд программы, во второй регистр помещает число команд, а в третий — номер текущей команды. После этого начинает работать второй модуль. Он рассматривает команду с очередным номером, определяет ее тип, выполняет ее, если это команда Z, S или T, и переходит к следующей по номеру команде. Если же это команда J, то модуль должен определить номер следующей команды.

Отдельные части универсальной программы можно написать — они обозримы. Все вместе можно написать на языке высокого уровня, например, на Паскале.

Начнем с примера.

Пример. Рассмотрим функцию $g(n) = \Phi(n,n) + 1$. Эта функция вычислима, потому что вычислима сама функция Φ (в R_1 помещаем n, затем выполняем команду $T(1,2)$, а потом запускаем программу, вычисляющую Φ). Таким образом g — вычислимая одноместная функция, и пусть программа, которая ее вычисляет имеет номер N, т.е. $g(n) = \varphi_N(n)$. Но тогда для $n = N$ получаем, что $\varphi_N(N)+1 = \Phi(N,N)+1 = g(N) = \varphi_N(N)$. На первый взгляд кажется, что здесь противоречие, и мы допустили серьезную ошибку в наших рассуждениях. Но на самом деле все просто — число N не входит в область определения функции φ_N. Обратите внимание на полученный результат: не изучая структуру программы, мы доказали, что программа φ_N не останавливается на входе $(N, 0, \ldots)$.

Наш следующий пример более серьезный.

Пример. Обозначим через $W(f)$ область определения функции f. Рассмотрим функцию

$$h(n) = \begin{cases} \varphi_n(n) + 1, & \text{если } n \in W(\varphi_n), \\ 0, & \text{если } n \notin W(\varphi_n). \end{cases}$$

Эта функция кажется похожей на функцию g из предыдущего примера, но на самом деле они совершено различны: функция h невычислима (обратите внимание, мы превратили g в h, сделав функцию тотальной).

Предположим, что h вычислима, тогда $h(n) = \varphi_N(n)$ для некоторого N и $h(N) = \varphi_N(N)$. Если $N \in W(\varphi_N)$, то мы получаем равенство двух чисел, одно из которых на 1 больше другого. Противоречие. Если $N \notin W(\varphi_N)$, то значение h равно 0, а функция справа не определена. Противоречие.

Из этого рассуждения опять-таки следует вывод о невозможности алгоритмической проверки корректности программ.

Упражнения.

(1) Найти $\varphi_1(1)$, $\varphi_2(1)$, $\varphi_3(1)$, $\varphi_4(1)$, $\varphi_5(1)$.

5. Невычислимые функции

В этом разделе мы построим целую серию невычислимых функций.

Мы будем работать с универсальной функцией $\Phi(k,l) = \varphi_k(l)$, которая является вычислимой двуместной функцией. Сначала докажем простое, но полезное утверждение.

Лемма 5.1. *Пусть $f(x,y)$ — вычислимая двуместная функция. Тогда существует такая вычислимая функция $k(x)$, что $f(x,y) = \varphi_{k(x)}(y)$.*

Доказательство. Фиксируем число a и рассмотрим *одноместную* функцию $f(a,y)$. Пусть F — программа, вычисляющая функцию f. Тогда программа Q_a, вычисляющая функцию $f(a,y)$ выглядит так: 1. T(1,2); 2. Z(1) (исходная конфигурация $(y,0,\ldots)$ преобразуется в конфигурацию $(0,y,0,\ldots)$); 3. S(1) ... (команда S(1) повторена a раз). В регистрах теперь стоят числа $(a,y,0\ldots)$). Далее идет программа F. Пусть $N_p(F) = m$. Найдем номер $k(a)$ программы Q_a. Последовательность номеров команд программы Q_a имеет вид:

$$10, 4, \underbrace{1, \ldots, 1}_{\text{a раз}}, \text{номера команд программы } F.$$

Соответствующая возрастающая последовательность имеет вид:

$$10, 14, 15, \ldots, 14+a, n_1+14+a, n_1+n_2+14+a, \ldots,$$

где $n_1, n_2, \ldots$ — последовательность номеров команд программы F. Следовательно,

$$N_p(Q_a) = 2^9 + 2^{13} + 2^{14} + \cdots + 2^{13+a} + 2^{14+a}m,$$

и мы видим, что функция $k(a) = N_p(Q_a)$ является вычислимой функцией от a. По построению $\varphi_{k(a)}(y) = f(a,y)$. □

Пусть нас интересует некоторое свойство вычислимой функции (например, тотальность, или строгая положительность, или монотонность). Мы будем говорить, что задача об этом свойстве разрешима, если функция

$$s(n) = \begin{cases} 1, & \varphi_n \text{ обладает этим свойством,} \\ 0, & \varphi_n \text{ не обладает этим свойством.} \end{cases}$$

вычислима.

Пример. Задача $\varphi_n \equiv 0$ неразрешима. Рассмотрим двуместную функцию f, заданную формулой

$$f(x,y) = \begin{cases} 0, & \text{если } x \in W_x, \\ \text{не определена}, & \text{если } x \notin W_x. \end{cases}$$

Легко видеть, что функция f вычислима (к программе, вычисляющей универсальную функцию Φ, мы приписываем в конце команду $Z(1)$, а в начале $T(1,2)$). Поэтому существует вычислимая функция $k(x)$, для которой $f(x,y) = \varphi_{k(x)}(y)$, т.е.

$$x \in W_x \Leftrightarrow \varphi_{k(x)} \equiv 0.$$

Пусть наша задача разрешима, т.е. существует программа, которая для начальной конфигурации $(x,0,\ldots)$ выдает ответ 1, если $\varphi_x \equiv 0$, и выдает ответ 0, если $\varphi_x \not\equiv 0$. Тогда, вычислив по данному x значение $k(x)$ и подставив его в программу, мы получим ответ на вопрос: верно ли, что $x \in W_x$? Но мы знаем, что эта задача неразрешима. Следовательно, наше предположение о разрешимости задачи $\varphi_x \equiv 0$ неверно.

Докажем общий результат о неразрешимости задач такого рода.

Теорема Райса. *Пусть $\mathcal{A}$ — множество вычислимых функций, а $\mathcal{B} \subset \mathcal{A}$ — собственное подмножество (т.е. $\mathcal{B} \neq \varnothing$ и $\mathcal{B} \neq \mathcal{A}$). Тогда задача $\varphi_x \in \mathcal{B}$ неразрешима.*

Замечание. В теореме идет речь о таких программах, что функции, вычисляемые ими, принадлежат множеству $\mathcal{B}$.

Доказательство. Так как задачи $\varphi_x \in \mathcal{B}$ и $\varphi_x \in \mathcal{A} \setminus \mathcal{B}$ или обе разрешимы, или обе неразрешимы, то мы будем считать, что нигде не определенная функция $f_\varnothing$ не принадлежит $\mathcal{B}$ (напомним, что такая функция "вычисляется", например, программой из одной команды $J(1,1,1)$). Пусть g — некоторая функция из $\mathcal{B}$. Рассмотрим функцию $f(x,y)$, заданную формулой:

$$f(x,y) = \begin{cases} g(y), & \text{если } x \in W_x, \\ \text{не определена}, & \text{если } x \notin W_x. \end{cases}$$

Легко видеть, что f вычислима. Тогда существует вычислимая функция $k(x)$, такая, что $f(x,y) = \varphi_{k(x)}(y)$. Таким образом,

$$x \in W_x \Rightarrow \varphi_{k(x)} = g \Rightarrow \varphi_{k(x)} \in \mathcal{B},$$
$$x \notin W_x \Rightarrow \varphi_{k(x)} = f_\varnothing \Rightarrow \varphi_{k(x)} \notin \mathcal{B}.$$

Это означает, что с помощью вычислимой функции $k(x)$ мы свели задачу $x \in W_x$ к задаче $\varphi_x \in \mathcal{B}$. Теперь, как и в предыдущем примере, приходим к выводу о неразрешимости задачи $\varphi_x \in \mathcal{B}$. □

Упражнения.

(1) Пусть $f(x, y) = x + y$. Что из себя представляет функция $k(x)$ из Леммы?
(2) Вычислима ли функция

$$g(n) = \begin{cases} 1, & n \in W_{n+1}; \\ 0, & n \notin W_{n+1}. \end{cases}$$

6. Рекурсивные функции

Рассмотрим другой подход к определению вычислимых функций, который позволяет классифицировать их по сложности. Этот подход использует понятие *рекурсивной функции*. Позже мы докажем, что вычислимые и рекурсивные функции — это одно и то же.

Определение 6.1. Следующие три функции называются *основными*:

(1) нуль-функция $0 : 0(x) = 0$;
(2) функция следования $n : n(x) = x + 1$;
(3) проекция $U_i^n : U_i^n(x_1, \ldots, x_n) = x_i$.

Замечание. Функции 0 и n — одноместны и тотальны. Каждая из них вычислима программой из одной команды, для нуль функции — это $Z(1)$, а для функции следования — это $S(1)$. Проекция — многоместная и тотальная функция. Она также вычислима программой из одной команды — $T(i, 1)$.

Определим три операции над функциями.

Определение 6.2. Рассмотрим функцию $f(y_1, \ldots, y_k)$ и функции $g_1(x_1, \ldots, x_n), \ldots, g_k(x_1, \ldots, x_n)$. Тогда о функции $h(x_1, \ldots, x_n) = f(g_1(x_1, \ldots, x_n), \ldots, g_k(x_1, \ldots, x_n))$ мы говорим, что она получена *подстановкой* функций $g_1, \ldots, q_k$ в функцию f.

Замечание. Если функции f и g_i тотальны то и функция h тотальна. Особенно важным является следующий факт: если функции f и g_i вычислимы, то и функция h тоже вычислима. Действительно, пусть функция вычисляется программой F, а функции $g_1, \ldots, g_k$ вычисляются программами $G_1, \ldots, G_k$, соответственно. Пусть программа F работает с l регистрами, а программа G_i, $i = 1, \ldots, k$, с m_i регистрами $i = 1, \ldots, k$. Пусть $N = \max\{k, l, n\}$. Модифицируем программы G_i следующим образом: программа G_1 будет теперь работать с регистрами $R_{N+1}, \ldots, R_{N+m_1}$, а результат будет помещать в регистр R_{N+1}; программа G_2 будет теперь работать с регистрами $R_{N+m_1+1}, \ldots, R_{N+m_1+m_2}$, а результат будет помещать в регистр R_{N+m_1+1}, и так далее. Исходная конфигурация $(x_1, \ldots, x_n, 0, \ldots)$. Программа работает так: содержимое регистров $R_1, \ldots, R_n$ переносится в регистры $R_{N+1}, \ldots, R_{N+n}$ и исполняется модуль G_1. Затем содержимое регистров $R_1, \ldots, R_n$ переносится в регистры $R_{N+m_1+1}, \ldots, R_{N+m_1+n}$ и исполняется модуль G_2. И так далее. В результате в регистрах $R_{N+1}, R_{N+m_1+1}, \ldots$ будут находиться числа $g_1(x_1, \ldots, x_n), g_2(x_1, \ldots, x_n), \ldots$. Переместим эти числа в регистры $R_1, \ldots, R_k$, поместим нули в регистры $R_{k+1}, \ldots, R_N$ и выполним программу F.

Определение 6.3. Рассмотрим функции $f(x_1, \ldots, x_n)$ и $g(x_1, \ldots, x_n, y, z)$, первая из которых n-местная, а вторая — $(n + 2)$-местная. Про $(n + 1)$-местную функцию $h(x_1, \ldots, x_n, y)$ мы говорим, что она получена рекурсией из функций f и g, если

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

Замечание.

(1) Если $n = 0$, то „функция“ f — это число.
(2) Если функции f и g тотальны, то функция h тоже тотальна.
(3) Если функции f и g вычислимы, то функция h тоже вычислима. Действительно, пусть f вычисляется программой F, а g — программой G. Программа, вычисляющая h, работает так: сначала с помощью программы F мы находим $h(x_1, \ldots, x_n, 0)$. Далее организуем цикл: программа G вычисляет $h(x_1, \ldots, x_n, 1)$ (начальная конфигурация $(x_1, \ldots, x_n, 0, h(x_1, \ldots, x_n, 0), 0 \ldots)$). Далее программа G вычисляет $h(x_1, \ldots, x_n, 2)$ (начальная конфигурация $(x_1, \ldots, x_n, 1, h(x_1, \ldots, x_n, 1), 0 \ldots)$) и т.д., пока не дойдем до нужного значения y.

Определение 6.4. Рассмотрим $(n+1)$-местную функцию $f(x_1, \ldots, x_n, y)$. Про n-местную функцию $g(x_1, \ldots, x_n)$ мы говорим, что она получена минимизацией функции f по переменной y, если $g(x_1, \ldots, x_n)$ равно минимальному y такому, что а) функция $f(x_1, \ldots, x_n, z)$ определена при всех $z \leqslant y$ и б) $f(x_1, \ldots, x_n, y) = 0$. Если такого y не существует, то набор $x_1, \ldots, x_n$ не принадлежит области определения функции g.

Замечание. Функция g может не быть тотальной, даже если f тотальна. Если функция f вычислима, то и функция g вычислима. Действительно, пусть f вычисляется программой F. Мы повторяем цикл с

программой F, каждый раз увеличивая значение регистра R_{n+1} на единицу, до тех пор, пока не получим нуль в регистре R_1.

Теперь мы можем дать определение.

Определение 6.5. Функция называется *рекурсивной*, если ее можно построить из основных функций с помощью конечного числа операций подстановки, рекурсии и минимизации.

Упражнения.

(1) Доказать рекурсивность функции

$$h(n) = \begin{cases} 1, & n = 1; \\ 2, & n = 2; \\ 0, & n > 2. \end{cases}$$

7. Рекурсивные функции, продолжение

В этом разделе мы убедимся, что класс рекурсивных функций достаточно обширен. Вот примеры рекурсивных функций.

- Сложение $h(x, y) = x + y$. Определяется рекурсией: $f(x) = x$ — проекция U_1^1, $g(x, y, z) = z + 1$ — проекция U_3^3 и следование. Тогда $h(x, y + 1) = (x + y) + 1 = g(x, y, (x + y))$.
- Умножение $h(x, y) = xy$. Определяется рекурсией: $f(x) = 0$ — нуль-функция, $g(x, y, z) = x + z$ — сложение проекций U_1^3 и U_3^3. Тогда $h(x, y + 1) = xy + x = g(x, y, xy)$.
- Возведение в степень $h(x, y) = x^y$. Определяется рекурсией: $f(x) = 1$ — нуль-функция и следование, $g(x, y, z) = xz$ — умножение проекций. Тогда $h(x, y + 1) = x^y \cdot x = g(x, y, x^y)$.
- Неполное уменьшение $d(x)$. Здесь $d(x) = x - 1$, если $x > 0$, и $d(0) = 0$. Определяется рекурсией: $g(y, z) = y$. Тогда $d(y + 1) = y = g(y, d(y))$.
- Неполное вычитание

  $$r(x, y) = \begin{cases} x - y, & \text{если } x \geqslant y, \\ 0, & \text{если } x < y. \end{cases}$$

  Определяется рекурсией: $g(x, y, z) = d(z)$. Тогда $r(x, y + 1) = d(r(x, y))$.
- Знак

  $$\operatorname{sg}(x) = \begin{cases} 0, & \text{если } x = 0, \\ 1, & \text{если } x > 0. \end{cases}$$

  Определяется рекурсией: $g(y, z) = 1$.
- Противоположный знак

  $$\overline{\operatorname{sg}}(x) = \begin{cases} 1, & \text{если } x = 0, \\ 0, & \text{если } x > 0. \end{cases}$$

  Подстановка: $\overline{\operatorname{sg}}(x) = r(1, \operatorname{sg}(x))$.
- Модуль разности $|x - y|$. Подстановка: $|x - y| = r(x, y) + r(y, x)$.
- Факториал $x!$. Рекурсия: $g(y, z) = (y + 1) \cdot z$. Тогда $(y + 1)! = g(y, y!)$.
- Минимум $\min(x, y)$. Подстановка: $\min(x, y) = r(x, r(x, y))$.
- Максимум $\max(x, y)$. Подстановка: $\max(x, y) = x + r(y, x)$.
- Остаток от деления y на x — функция $\operatorname{rm}(x, y)$ (условимся считать, что $\operatorname{rm}(0, y) = y$). Так как

  $$\operatorname{rm}(x, y + 1) = \begin{cases} \operatorname{rm}(x, y) + 1, & \text{если } \operatorname{rm}(x, y) + 1 \neq x, \\ 0, & \text{если } \operatorname{rm}(x, y) + 1 = x, \end{cases}$$

  то получаем рекурсию: $\operatorname{rm}(x, y + 1) = g(x, y, \operatorname{rm}(x, y))$, где $g(x, y, z) = (z + 1)\operatorname{sg}(|x - (z + 1)|)$.
- Частное от деления y на x — функция $\operatorname{qt}(x, y)$ (условимся считать, что $\operatorname{qt}(0, y) = 0$). Так как

  $$\operatorname{qt}(x, y + 1) = \begin{cases} \operatorname{qt}(x, y) + 1, & \text{если } \operatorname{rm}(x, y) + 1 = x, \\ \operatorname{qt}(x, y), & \text{если } \operatorname{rm}(x, y) + 1 \neq x, \end{cases}$$

  то получаем рекурсию $\operatorname{qt}(x, y + 1) = g(x, y, \operatorname{qt}(x, y))$, где $g(x, y, z) = z + \overline{\operatorname{sg}}(|x - (z + 1)|)$.

Мы видим, что обычные арифметические функции рекурсивны, более того, минимизация *не* используется для их построения. Дадим определение.

Определение 7.1. Рекурсивная функция называется *примитивно рекурсивной*, если для ее построения операция минимизации не нужна.

Рассмотрим три операции над примитивно рекурсивными функциями.

- Ограниченное суммирование. Пусть $f(x_1,\dots,x_n,y)$ и $s(y)$ — тотальные примитивно рекурсивные функции. Определим функцию
$$h(x_1,\dots,x_n,y)=\sum_{z<s(y)} f(x_1,\dots,x_n,z).$$
Тогда функция h примитивно рекурсивна. Действительно, рассмотрим функцию h', заданную рекурсией: $h'(x_1,\dots,x_n,0)=0$, $h'(x_1,\dots,x_n,z+1)=h'(x_1,\dots,x_n,z)+f(x_1,\dots,x_n,y)$. Здесь $g(x_1,\dots,x_n,y,z)=f(x_1,\dots,x_n,y)+z$. Тогда $h(y)=h'(s(y))$ — подстановка.
- Ограниченное произведение. Пусть $f(x_1,\dots,x_n,y)$ и $s(y)$— тотальные примитивно рекурсивные функции. Определим функцию
$$h(x_1,\dots,x_n,y)=\prod_{z<s(y)} f(x_1,\dots,x_n,z).$$
Тогда функция h примитивно рекурсивна. Доказательство этого утверждения аналогично доказательству предыдущего.
- Ограниченная минимизация. Пусть $f(x_1,\dots,x_n,y)$ и $s(y)$ — тотальные примитивно рекурсивные функции. Определим функцию $h(x_1,\dots,x_n,y)$ равную минимальному $z<s(y)$, для которого $f(x_1,\dots,x_n,z)=0$, если такое z существует, и равную $s(y)$, если такого z нет. Тогда функция h примитивно рекурсивна. Действительно,
$$h(x_1,\dots,x_n,y)=\sum_{v<s(y)}\left(\prod_{u\leqslant v}\operatorname{sg}(f(x_1,\dots,x_n,u))\right).$$

Нам далее понадобятся функции, связанные с двоичным представлением натуральных чисел.

- Функция $\alpha(i,x)$ — коэффициент при 2^i в двоичном представлении числа x: $x=\sum_{i=0}^{\infty}\alpha(i,x)2^i$. Эта функция принимает значения 0 и 1. Так как $\operatorname{qt}(2^i,x)=\alpha(i,x)+\alpha(i+1,x)\cdot 2+\cdots$, то $\alpha(i,x)=\operatorname{rm}(2,\operatorname{qt}(2^i,x))$ и является примитивно рекурсивной функцией.
- Функция $l(x)$ — число таких i, что $\alpha(i,x)=1$: $l(x)=\sum_{i\leqslant x}\alpha(i,x)$. Так как эта функция задается операцией ограниченного суммирования, то она примитивно рекурсивна.
- Рассмотрим двоичное разложение числа $x>0$: $x=2^{b(1,x)}+2^{b(2,x)}+\cdots+2^{b(l(x),x)}$. Значение функции $b(i,x)$ — это показатель степени, отвечающий i-му ненулевому знаку в двоичном представлении числа x (количество ненулевых знаков в двоичном представлении равно $l(x)$). Тогда
$$b(i,x)=\min_{y<x}\{c(y,x):=\sum_{z\leqslant y}\alpha(z,x)=i\}.$$
Так как функцию b можно определить с помощью операций ограниченного суммирования и ограниченной минимизации, то она примитивно рекурсивна.

Кроме того, нам будет необходима функция $p(n)$, равная n-му простому числу. Эта функция примитивно рекурсивна. Действительно, число $dd(n)$ делителей числа n равно $dd(n)=\sum_{k=1}^{n}\overline{\operatorname{sg}}(\operatorname{rm}(k,n))$ (ограниченное суммирование). Функция $\pi(n)=\overline{\operatorname{sg}}(dd(n)-2)$ равна 1, если n простое, и 0 — если нет. Функция $\Pi(n)=\sum_{i=1}^{n}\pi(n)$ равна количеству простых чисел $\leqslant n$ (ограниченное суммирование). Тогда $p(n)=\min_m r(n,\Pi(m))$ (минимизация). Эти рассуждения доказывают рекурсивность функции $p(n)$. Чтобы доказать ее примитивную рекурсивность, нужно показать, что минимизацию можно заменить на ограниченную минимизацию. Покажем, что $p(n)<2^{2^n}=s(n)$. Имеем, $2=p(1)<2^{2^1}$. Пусть $p(k)<2^{2^k}$ при $k=1,\dots,n$. Число $p(1)\times\cdots\times p(n)+1$ не делится на простые числа $p(1),\dots,p(n)$, следовательно,
$$p(n+1)\leqslant p(1)\times\cdots\times p(n)+1<2^{2^1}\times\cdots\times 2^{2^n}=2^{2^1+\cdots+2^n}<2^{2^{n+1}}.$$
Теперь функцию можно определить с помощью ограниченной минимизации $p(n)=\min_{m<s(n)} r(n,\Pi(m))$.

Мы знаем, что частичные функции не могут быть примитивно рекурсивными. Но, как мы только что видели, арифметические тотальные функции примитивно рекурсивны. Возникает вопрос, существуют ли тотальные рекурсивные функции, не являющиеся примитивно рекурсивными? Пример такой функции будет рассмотрен в следующем разделе.

Упражнения.

(1) Показать, что частичные функции $x-y$, $\sqrt{x}$, $\log_2(x)$ рекурсивны.
(2) Является ли примитивно рекурсивной фунуция
$$g(n)=\begin{cases}1, & \text{если } n \text{ полный квадрат;}\\ 0, & \text{в противном случае.}\end{cases}$$
(3) Рекурсивна ли функция $p(x_1,\dots,x_n)$, где $p\in\mathbb{Z}[x_1,\dots,x_n]$ — многочлен с целыми коэффициентами?

8. Функция Аккермана

Определение 8.1. Функцией Аккермана называется тотальная двуместная функция $\psi(x, y)$, заданная следующими условиями:

- $\psi(0, y) = y + 1$;
- $\psi(x + 1, 0) = \psi(x, 1)$;
- $\psi(x + 1, y + 1) = \psi(x, \psi(x + 1, y))$.

Замечание. Рекурсивность функции Аккермана вытекает из эквивалентности понятий вычислимости и рекурсивности, которая будет обсуждаться в следующем разделе.Эта функция определена с помощью "двойной рекурсии", и, на первый взгляд, кажется примитивно рекурсивной. Попробуем, однако, найти ее значения.

Мы знаем, что $\psi(0, y) = y + 1$. А что можно сказать о $\psi(1, y)$? Это просто: $\psi(1, 0) = \psi(0, 1) = 2$, а $\psi(1, y + 1) = \psi(0, \psi(1, y)) = \psi(1, y) + 1$. Из чего следует, что $\psi(1, y) = y + 2$. Чему равно $\psi(2, y)$? Так как $\psi(2, 0) = \psi(1, 1) = 3$ и $\psi(2, y + 1) = \psi(1, \psi(2, y)) = \psi(2, y) + 2$, то $\psi(2, y) = 2y + 3$. Найдем теперь $\psi(3, y)$. Так как $\psi(3, 0) = \psi(2, 1) = 5$ и $\psi(3, y + 1) = \psi(2, \psi(3, y)) = 2\psi(3, y) + 3$, то $\psi(3, 1) = 2 \cdot 5 + 3 = 13$, $\psi(3, 2) = 2 \cdot 13 + 3 = 29$ и т.д. Нетрудно получить формулу: $\psi(3, y) = 2^{y+3} - 3$. Следующий шаг еще интереснее: чему равно $\psi(4, y)$? Так как $\psi(4, 0) = \psi(3, 1) = 13$ и $\psi(4, y + 1) = \psi(3, \psi(4, y)) = 2^{\psi(4,y)+3} - 3$, то $\psi(4, 1) = 2^{16} - 3$, $\psi(4, 2) = 2^{2^{16}} - 3$ и т.д. Мы видим, что функция Аккермана растет очень быстро с ростом y. Это наблюдение и поможет нам доказать, что ψ не является примитивно рекурсивной. Точнее мы докажем, что функция Аккермана растет быстрее любой примитивно рекурсивной функции.

Теорема 8.1. *Функция Аккермана $\psi(x, y)$ не является примитивно рекурсивной.*

Мы будем последовательно изучать свойства функции ψ, что и приведет нас, в конце концов, к доказательству теоремы.

Лемма 8.1. $\psi(x, y) > y$.

Доказательство. Будем доказывать это утверждение индукцией по x. При $x = 0$ это утверждение справедливо. Предположим, что $\psi(x, y) > y$, т.е. $\psi(x, y) \geqslant y + 1$. Докажем, что $\psi(x + 1, y) > y$. Зафиксируем $x + 1$ и будем доказывать последнее утверждение индукцией по y. Так как $\psi(x + 1, 0) = \psi(x, 1) \geqslant 2$, то утверждение верно для $y = 0$. Предположим, что $\psi(x + 1, y) > y$. Тогда

$$\psi(x + 1, y + 1) = \psi(x, \psi(x + 1, y)) \geqslant \psi(x + 1, y) + 1 \geqslant y + 1 + 1 = y + 2.$$

Лемма доказана. □

Лемма 8.2. *Функция $\psi(x, y)$ монотонна по y.*

Доказательство. Будем доказывать это утверждение индукцией по x. Так как

$$\psi(0, y) = y + 1 < y + 2 = \psi(0, y + 1),$$

то при $x = 0$ лемма справедлива. Пусть $\psi(x, y) < \psi(x, y + 1)$ для всех $y \geqslant 0$. Тогда $\psi(x + 1, y + 1) = \psi(x, \psi(x + 1, y)) > \psi(x + 1, y)$. Лемма доказана. □

Лемма 8.3. *Функция $\psi(x, y)$ монотонна по x.*

Доказательство. Сначала докажем, что $\psi(x + 1, y) \geqslant \psi(x, y + 1)$. Будем доказывать это утверждение индукцией по y. Для $y = 0$ это верно: $\psi(x + 1, 0) = \psi(x, 1)$. Пусть это утверждение верно для некоторого y, докажем его для $y + 1$. Имеем,

$$\psi(x + 1, y + 1) = \psi(x, \psi(x + 1, y)) \geqslant \psi(x, \psi(x, y + 1))$$

(здесь использовано предположение индукции и монотонность функции по второму аргументу). Так как $\psi(x, y) > y$, то $\psi(x, y + 1) \geqslant y + 2$, следовательно, $\psi(x, \psi(x, y + 1)) \geqslant \psi(x, y + 2)$. Утверждение доказано. Теперь докажем лемму:

$$\psi(x + 1, y) \geqslant \psi(x, y + 1) > \psi(x, y).$$

□

Доказательство теоремы. Сначала докажем, что для любой примитивно рекурсивной функции $f(x_1, \ldots, x_n)$ найдется такое m, что $f(x_1, \ldots, x_n) < \psi(m, \max\{x_1, \ldots, x_n\})$. Доказывать это утверждение мы будем так: мы покажем, что это верно для основных функций. Далее покажем, что если это верно для функций $f(y_1, \ldots, y_k)$, $g_1(x_1, \ldots, x_n)$, ..., $g_k(x_1, \ldots, x_n)$, то это верно и для подстановки. А потом покажем, что если это верно для функций $f(x_1, \ldots, x_n)$ и $g(x_1, \ldots, x_n, y, z)$, то это верно и для функции $h(x_1, \ldots, x_n, y)$, рекурсивно определяемой с помощью функций f и g.

Для основных функций это утверждение очевидно: для 0-функции $m = 0$, для проекции $m = 0$ и для функции следования $m = 1$.

Подстановка. Пусть

$$\begin{aligned} f(y_1,\ldots,y_k) &< \psi(m_0,\max(y_1,\ldots,y_k)),\\ g_1(x_1,\ldots,x_n) &< \psi(m_1,\max(x_1,\ldots,x_n)),\\ &\vdots\\ g_k(x_1,\ldots,x_n) &< \psi(m_k,\max(x_1,\ldots,x_n)). \end{aligned}$$

Пусть $s=\max(m_1,\ldots,m_k)$, $v=\max(s,m_0)$, зафиксируем набор $(x_1,\ldots,x_n)$ и пусть $t=\max(x_1,\ldots,x_n)$. Имеем,

$$\begin{aligned} f(g_1,\ldots,g_k) < \psi(m_0,\max(g_1,\ldots,g_k)) &< \psi(m_0,\psi(s,\max(x_1,\ldots,x_n))) = \psi(m_0,\psi(s,t))\\ &< \psi(v,\psi(v+1,t)) = \psi(v+1,t+1) \leqslant \psi(v+2,t). \end{aligned}$$

Следовательно, для числа m для функции $f(g_1,\ldots,g_k)$ равно $v+2$.

Рекурсия. Мы проведем доказательство для одноместной функции h. Доказательство в общем случае технически довольно трудное. Итак, пусть $h(y+1)=g(y,h(y))$ и $g(y,z)<\psi(s,\max(y,z))$. Будем считать, что s выбрано настолько большим, что $h(0)\leqslant\psi(s+1,0)$. Покажем, что $h(y)\leqslant\psi(s+1,y)$. Предположим, что это утверждение верно для некоторого y. Тогда

$$h(y+1)=g(y,h(y))<\psi(s,\max(y,h(y))\leqslant\psi(s,\psi(s+1,y))=\psi(s+1,y+1).$$

Здесь мы воспользовались тем, что $y\leqslant\psi(s+1,y)$. Следовательно, число m для функции h равно $s+1$.

Предположим теперь, что функция Аккермана примитивно рекурсивна. Тогда существует такое N, что $\psi(x,y)\leqslant\psi(N,\max(x,y))$. Но тогда $\psi(N,N)<\psi(N,\max(N,N))=\psi(N,N)$. Противоречие. □

Упражнения.

(1) Написать на Паскале программу для вычисления функции Аккермана (компьютер, разумеется, не сможет вычислить, например, $\psi(4,4)$).
(2) Найти $\psi(5,0)$ и $\psi(5,1)$.

9. Рекурсивность и вычислимость

Каждая рекурсивная функция вычислима. Но верно и обратное: каждая вычислимая функция рекурсивна. Доказательство этого утверждения требует усилий.

Пусть f — n-местная функция, которая вычисляется программой P из s команд и пусть $(\bar{x},0,0,\ldots)$, где $\bar{x}=(x_1,\ldots,x_n)$, — начальное состояние регистров. Состояние регистров мы будем называть *конфигурацией*. Если программа использует только регистры $R_1,\ldots,R_m$, то остальные регистры всегда будут нулевыми. Обозначим через p_k — k-е простое число. Конфигурацию $(r_1,r_2,\ldots)$ мы будем описывать числом $c=\prod_{k=1}^{m}p_k^{r_k}$. Очевидно, что по числу конфигурация восстанавливается однозначно.

Состояние вычисления полностью описывается двумя числами c и j — номером следующей команды. Состояние мы будем описывать одним числом $\sigma=N_2(c,j)$.

Числа c,j,σ меняются в ходе вычисления и зависят от начальной конфигурации $\bar{x}$ и t — количества выполненных шагов программы P. Определим функции:

(1) $c(\bar{x},t)$ — конфигурация после выполнения t шагов программы или заключительная конфигурация, если программа остановилась за t или меньше шагов ($c(\bar{x},0)=2^{x_1}\cdots p_n^{x_n}$);
(2) $j(\bar{x},t)$ — номер следующей команды после выполнения t шагов программы или 0, если программа остановилась за t шагов или меньше ($j(\bar{x},0)=1$);
(3) $\sigma(\bar{x},t)=N_2(c(\bar{x},t),j(\bar{x},t))$.

Пусть $g(\bar{x})$ является минимизацией функции $j(\bar{x},t)$ по t. Тогда $c(\bar{x},g(\bar{x}))$ — заключительная конфигурация (если функция $g(\bar{x})$ определена) и степень двойки, на которую делится это число, и есть $f(\bar{x})$. Таким образом, если функции c и j рекурсивны, то f тоже рекурсивна.

Чтобы описать изменения функций и в процессе вычисления рассмотрим еще две функции: $\mathrm{config}(\sigma)$ и $\mathrm{nxt}(\sigma)$. Эти функции зависят от числа состояния $\sigma=N_2(c,j)$. Функция config — это новая конфигурация после выполнения команды с номером j, $1\leqslant j\leqslant s$, примененной к конфигурации c (или заключительная конфигурация); функция nxt — это номер следующей команды после выполнения j-й команды (или 0). Обратите внимание, функции nxt и config *не* зависят от шага программы, а лишь от текущей конфигурации и номера команды в программе. Функцию σ можно задать с помощью рекурсии

$$\begin{aligned} \sigma(\bar{x},0) &= N_2((\bar{x},1),\\ \sigma(\bar{x},t+1) &= N_2(\mathrm{config}(\sigma(\bar{x},t)),\mathrm{nxt}(\sigma(\bar{x},t))). \end{aligned}$$

Мы видим, что нам достаточно доказать примитивную рекурсивность функций config и nxt.

Далее через $\beta(I)$ мы будем обозначать код команды I, т.е. ее номер в нашей нумерации. Рассмотрим функции:

- $\mathrm{gn}(j)$ — код команды с номером j, если $1\leqslant j\leqslant s$, и 0 в противном случае. Примитивная рекурсивность этой функции следует из того, что она тождественно равна нулю при $j>s$.

- $\mathrm{ch}(c, z)$ — конфигурация, которая получается в результате применения команды с кодом z к конфигурации c.
- $\nu(c, j, z)$ — номер следующей команды, когда к конфигурации c применена команда с кодом z и эта команда имеет номер j в программе (или 0).

(Функции ch и ν не имеют отношения к нашей программе — они применимы к произвольной конфигурации и коду.) Тогда

$$\mathrm{config}(\sigma) = \mathrm{ch}(c, \mathrm{gn}(j)),$$
$$\mathrm{nxt}(\sigma) = \nu(c, j, \mathrm{gn}(j)).$$

Осталось доказать примитивную рекурсивность функций ch и ν. Это уже очевидно. Как работает функция ch? Распознается тип команды и происходит изменение в конфигурации в зависимости от типа. Что касается функции ν, то распознается тип команды, если это команда Z, S или T, то $j := j + 1$, если это команда J, то требуется дополнительная работа.

„Распознавание типа команды“можно организовать, например, так. Пусть функция $f(n)$ задана следующим образом:

$$f(n) = \begin{cases} g_0(n), & n \equiv 0 \mod 4, \\ g_1(n), & n \equiv 1 \mod 4, \\ g_2(n), & n \equiv 2 \mod 4, \\ g_3(n), & n \equiv 3 \mod 4. \end{cases}$$

Тогда функция f примитивно рекурсивна, если примитивно рекурсивны функции g_i. Действительно,

$$\begin{aligned} f(n) =& |\mathrm{rm}(4,n) - 1| \cdot |\mathrm{rm}(4,n) - 2| \cdot |\mathrm{rm}(4,n) - 3| \cdot g_0(n)/6 + \mathrm{rm}(4,n) \cdot |\mathrm{rm}(4,n) - 2| \cdot |\mathrm{rm}(4,n) - 3| \cdot g_1(n)/2 \\ &+ \mathrm{rm}(4,n) \cdot |\mathrm{rm}(4,n) - 1| \cdot |\mathrm{rm}(4,n) - 3| \cdot g_2(n)/2 + \mathrm{rm}(4,n) \cdot |\mathrm{rm}(4,n) - 1| \cdot |\mathrm{rm}(4,n) - 2| \cdot g_3(n)/6. \end{aligned}$$

В случае функции ch функции g_i таковы:

- функция $g_0(n)$ описывает обнуление регистра $R_{n/4}$: мы находим простое число p с номером $n/4$; находим степень k, с которой p входит в разложение числа c, и выполняем деление c/p^k;
- функция $g_1(n)$ описывает увеличение регистра $R_{(n+3)/4}$ на единицу: мы находим простое число p с номером $(n + 3)/4$ и умножаем c на p;
- функция $g_2(n)$ описывает пересылку содержимого регистра R_k в регистр R_l, $N_2(k, l) = (n + 2)/4$;
- функция $g_3(n)$ конфигурацию не меняет.

Упражнения.

(1) Доказать, что функция $s(m, n)$, равная максимальному k такому, что n делится на m^k, примитивно рекурсивна.
(2) Доказать примитивную рекурсивность функции ν.

10. Перечислимые множества

В этом разделе мы дадим определение перечислимого множества и расскажем о теореме Матиясевича.

Определение 10.1. Подмножество $A \subset \mathbb{N}$ ($\mathbb{N}$ — множество натуральных чисел) называется *разрешимым*, если вычислима тотальная функция

$$f_A(x) = \begin{cases} 1, & x \in A, \\ 0, & x \notin A. \end{cases}$$

Разрешимыми являются такие подмножества, как подмножество четных чисел, подмножество полных квадратов, подмножество простых чисел и т.д. Более обширным классом является класс перечислимых подмножеств.

Определение 10.2. Подмножество $A \subset \mathbb{N}$ называется *перечислимым*, если вычислима функция

$$g_A(x) = \begin{cases} 1, & x \in A, \\ \text{не определена}, & x \notin A. \end{cases}$$

Мы доказывали, что функция

$$f(x) = \begin{cases} 1, & x \in W_x \\ \text{не определена}, & x \notin W_x \end{cases}$$

вычислима, а функция

$$g(x) = \begin{cases} 1, & x \in W_x \\ 0, & x \notin W_x \end{cases}$$

не вычислима. Следовательно, множество

$$B = \{x \in \mathbb{N} : x \in W_x\}$$

перечислимо, но не разрешимо.

Есть любопытный критерий разрешимости перечислимого множества.

Теорема 10.1. *Если множество A и его дополнение CA оба перечислимы, то A разрешимо.*

Доказательство. Пусть программа G_1 вычисляет функцию g_A, а программа G_2 — функцию g_{CA}. Поместим в первый регистр число n и запустим обе программы одновременно (например, сначала G_1 выполняет одну команду, потом G_2, потом снова G_1, и так далее). В какой-то момент одна из программ завершит работу и остановится. Если остановилась программа G_1, то $n \in A$ и тогда поместим в первый регистр единицу, Если остановилась программа G_2, то $n \in CA$ и тогда поместим в первый регистр нуль. □

Следствие. Множество

$$CB = \{x \in \mathbb{N} : x \notin W_x\}$$

не перечислимо.

Смысл понятия перечислимости иллюстрируется следующей теоремой.

Теорема 10.2. *Множество $A \neq \varnothing$ перечислимо тогда и только тогда, когда A является множеством значений тотальной одноместной функции.*

Доказательство. Пусть функция g_A вычисляется программой F и пусть $a \in A$ — некоторый элемент. Легко видеть, что A является множеством значений следующей тотальной *двуместной* функции:

$$f(m,n) = \begin{cases} m, & \text{если } F \text{ вычисляет } g_A(m) \text{ за } n \text{ или менее шагов,} \\ a, & \text{в остальных случаях.} \end{cases}$$

Очевидно, что функция f вычислима. Теперь нам нужно построить *одноместную* функцию h, множество значений которой совпадает с множеством значений функции f. Положим $h(n) = f(k,l)$, где $n = N_2(k,l)$. □

Так как любое подмножество $A \subset \mathbb{N}$ можно считать последовательностью натуральных чисел $A = (a_1, a_2, \ldots)$, $a_1 < a_2 < \ldots$, то естественно спросить: а можно ли перечислить A тотальной *возрастающей* функцией h, т.е. такой, что $h(n) = a_n$ для всех n?

Теорема 10.3. *Если множество A перечисляется возрастающей функцией h, то A разрешимо.*

Доказательство. Рассмотрим программу F, работающую следующим образом: F последовательно вычисляет $h(1)$, $h(2)$, и так далее, до тех пор, пока $h(i)$ не станет больше (или равным) данному n. Если $h(i) = n$, то в первый регистр помещаем 1, а если $h(i) > n$, то — 0. Тогда F вычисляет характеристическую функцию множества A. Следовательно, множество A разрешимо. □

Существует совершенно неожиданное описание перечислимых множеств.

Определение 10.3. Подмножество $A \subset \mathbb{N}$ называется *диофантовым*, если существует многочлен $p(x, y_1, \ldots, y_n)$ с целыми коэффициентами, такой, что

$$x \in A \Leftrightarrow \exists y_1 \ldots \exists y_n : p(x, y_1, \ldots, y_n) = 0.$$

Теорема (Ю.Матиясевич). *Перечислимые множества — диофантовы и наоборот.*

Это трудная теорема. Ее доказательство выходит далеко за пределы нашего курса.

Теорема 10.4. *Множество перечислимо тогда и только тогда, когда оно является множеством неотрицательных значений некоторого многочлена $p(x_1, \ldots, x_n)$ с целыми коэффициентами, причем переменные $x_1, \ldots, x_n$ пробегают $\mathbb{Z}_{\geqslant 0}$.*

Доказательство. Если A — множество неотрицательных значений некоторого многочлена, то оно конечно перечислимо.

Обратно, пусть A — перечислимое множество. Тогда существует многочлен $q(x, y_1, \ldots, y_m)$ такой, что

$$x \in A \Leftrightarrow \exists y_1 \ldots \exists y_m : q(x, y_1, \ldots, y_m) = 0.$$

Рассмотрим многочлен

$$p(x, y_1, \ldots, y_m) = x - (x+1)(q(x, y_1, \ldots, y_m))^2.$$

Число $p(x, y_1, \ldots, y_m)$ неотрицательно, только в том случае, когда $q(x, y_1, \ldots, y_m) = 0$. Но в этом случае $x \in A$, а $p(x, y_1, \ldots, y_m) = x$. □

Пример. Множество чисел Фибоначчи очевидно перечислимо, значит оно перечисляется многочленом. Вот он:

$$2x^4y + x^3y^2 - 2x^2y^3 - x^5 - xy^4 + 2x.$$

Пример. Множество простых чисел разрешимо и, значит, перечислимо. В существование многочлена, перечисляющего множество множество простых чисел, трудно поверить, но его существование вытекает из теоремы Матиясевича. Самый простой такой многочлен имеет степень 25 и зависит от 26 переменных $a, \ldots, z$. Вот он:

$$\begin{aligned}(k+2)\{1-[wz+h+j-q]^2-[(gk+2g+k+1)(h+j)+h-z]^2-[2n+p+q+z-e]^2-\\ -[16(k+1)^3(k+2)(n+1)^2+1-f^2]^2-[e^3(e+2)(a+1)^2+1-o^2]^2-[(a^2-1)y^2+1-x^2]^2-\\ -[16r^2y^4(a^2-1)+1-u^2]^2-[((a+u^2(u^2-a))^2-1)(n+4dy)^2+1-(x+cu)^2]^2-[n+l+v-y]^2-\\ -[(a^2-1)l^2+1-m^2]^2-[ai+k+1-l-i]^2-[p+l(a-n-1)+b(2an+2a-n^2-2n-2)-m]^2-\\ -[q+y(a-p-1)+s(2ap+2a-p^2-2p-2)-x]^2-[z+pl(a-p)+t(2ap-p^2-1)-pm]^2\}.\end{aligned}$$

Пример. Нетрудно построить диофантово множество с экспоненциальным ростом используя теорему Пелля.

Теорема Пелля. *Рассмотрим уравнение (уравнение Пелля)*

$$x^2 - d\,y^2 = 1,$$

где целое положительное число d не делится на квадрат простого числа (d „свободно от квадратов“). Тогда существует „наименьшее“целое положительное решение x_1, y_1 этого уравнения такое, что числа x_n, y_n, где $x_n+\sqrt{d}\,y_n = (x_1+\sqrt{d}\,y_1)^n$, являются решениями этого уравнения при любом $n > 1$ и все целые положительные решения этого уравнения описываются этой формулой.

Рассмотрим уравнение

$$x^2 - 3y = 1.$$

Очевидно, что $x_1 = 2, y_1 = 1$ — наименьшее решение. Из рекуррентного соотношения

$$x_{n+1} + \sqrt{3}\,y_{n+1} = (2+\sqrt{3})(x_n + \sqrt{3}\,y_n)$$

получаем, что

$$x_n = \frac{1}{2}(2+\sqrt{3})^n + \frac{1}{2}(2-\sqrt{3})^n.$$

Так как число $2-\sqrt{3} < 1$, то x_n растет как $\frac{1}{2}(2+\sqrt{3})^n$.

Упражнения.

(1) Доказать вычислимость функции $f(m,n)$ из доказательства Теоремы 2.
(2) Докажите перечислимость множества неотрицательных значений многочлена $p(x_1, \ldots, x_n)$ с целыми коэффициентами.

11. Элементарные функции

Элементарные функции — естественный и широкий подкласс класса примитивно рекурсивных функций, просто характеризуемый в терминах сложности вычисления.

Определение 11.1. Класс $\mathcal{E}$ *элементарных функций* — это наименьший класс, который

- содержит основные функции, а также сложение, умножение и ограниченное вычитание;
- замкнут относительно подстановки;
- замкнут относительно операций ограниченного суммирования и ограниченного произведения.

Грубо говоря, $\mathcal{E}$ — это класс функций, которые можно получить итерацией операций обычной арифметики. Элементарные функции примитивно рекурсивны, потому что ограниченное суммирование и ограниченное произведение — это примитивно рекурсивные операции. Так как операцию ограниченной минимизации можно определить в терминах ограниченного суммирования, ограниченного произведения и в терминах функции знак, то функция $g(x,y) = \min_{z<y}[f(x,z) = 0]$ элементарна, если элементарна функция f.

Многие функции из нашего списка примитивно рекурсивных функций определяются через рекурсию, которая *не* является элементарной операцией. Но с эти можно справиться.

(1) $x^y = \prod_{i<y} x = \prod_{i<y} U_1^2(x,i)$.
(2) $\mathrm{qt}(x,y) = \min_{z\leqslant y}[r(y,(z+1)x]$, где $r(a,b)$ — неполное вычитание.
(3) $\mathrm{rm}(x,y) = r(y - x\,\mathrm{qt}(x,y))$.
(4) p_n. Для вычисления этой функции используются операции ограниченного суммирования и ограниченной минимизации.

Что касается рекурсии, то эта операция сохраняет элементарные функции, если ее немного модифицировать. Далее через $\bar{x}$ мы будем обозначать набор $(x_1, \ldots, x_n)$. Пусть $|\bar{x}| = \max_{1 \leqslant i \leqslant n} x_i$.

Теорема 11.1. *Пусть $f(\bar{x})$ и $g(\bar{x}, y, z)$ — элементарные функции. Рекурсивно определяемая функция $h(\bar{x}, y)$:*

$$h(\bar{x}, 0) = f(\bar{x})$$
$$h(\bar{x}, y+1) = g(\bar{x}, y, h(\bar{x}, y))$$

элементарна, если существует элементарная функция $b(\bar{x}, y)$, для которой $h(\bar{x}, y) \leqslant b(\bar{x}, y)$ для всех $\bar{x}, y$. (Такую рекурсию мы будем называть ограниченной рекурсией.)

Доказательство. Пусть p_n — n-е простое число. Рассмотрим функцию

$$s(\bar{x}, y) = 2^{h(\bar{x},0)}\, 3^{h(\bar{x},1)} \cdots p_{y+1}^{h(\bar{x},y)} = \prod_{z \leqslant y} p_{z+1}^{h(\bar{x},z)}$$

и функцию

$$c(\bar{x}, y) = \prod_{z \leqslant y} p_{z+1}^{b(\bar{x},z)}.$$

Функция $c(\bar{x}, y)$ элементарна и $s(\bar{x}, y) \leqslant c(\bar{x}, y)$. Теперь функцию $h(\bar{x}, y)$ можно определить с помощью ограниченной минимизации:

$$h(\bar{x}, y) = \min_{s \leqslant c(\bar{x},y)} (\forall z < y : s_{z+2} = g(\bar{x}, z, s_{z+1})),$$

где s_k — это показатель степени p_k в разложении s на простые множители. □

Наша цель — охарактеризовать элементарные функции в терминах сложности вычисления. Первым шагом будет следующая теорема.

Теорема 11.2. *Пусть $f(\bar{x})$ — элементарная функция, тогда найдется такое число k, что $f(\bar{x}) \leqslant b_k(|\bar{x}|)$, где $b_0(z) = z$, $b_1(z) = 2^{b_0(z)}, \ldots, b_{t+1}(z) = 2^{b_t(z)}$.*

Доказательство. Мы будем использовать тот факт, что b_k — возрастающая функция и что $z^2 < 2^{2^z}$ для всех z.

Мы докажем справедливость теоремы для основных элементарных функций и покажем, что ее справедливость сохраняется при подстановке, ограниченном суммировании и ограниченном произведении.

1. $x + 1 \leqslant 2^x$.

2. $U_i^n(\bar{x}) \leqslant |\bar{x}|$.

3. $r(x, y) \leqslant \max(x, y)$.

4. $x + y \leqslant 2\max(x, y) \leqslant 2^{\max(x,y)}$.

5. $xy \leqslant (\max(x, y))^2 \leqslant 2^{2^{\max(x,y)}}$.

6. Пусть $h(\bar{x}) = f(g_1(\bar{x}), \ldots, g_m(\bar{x}))$ и $g_i(\bar{x}) \leqslant b_{k_i}(|\bar{x}|)$, $1 \leqslant i \leqslant m$, $f(\bar{y}) \leqslant b_l(|\bar{y}|)$. Пусть $k = \max(k_1, \ldots, k_m)$. Тогда

$$\begin{aligned} h(\bar{x}) &\leqslant b_l(\max(g_1(\bar{x}), \ldots, g_m(\bar{x}))) \\ &\leqslant b_l(\max(b_{k_1}(|\bar{x}|), \ldots, b_{k_m}(|\bar{x}|))) \leqslant b_l(b_k(|\bar{x}|)) = b_{k+l}(|\bar{x}|). \end{aligned}$$

7. Пусть $g(\bar{x}, y) = \sum_{z<y} f(\bar{x}, y)$ и $f(\bar{x}, y) \leqslant b_k(\max(|\bar{x}|, y))$. Положим $u = \max(|\bar{x}|, y)$. Тогда

$$\begin{aligned} g(\bar{x}, y) &\leqslant \sum_{z<y} b_k(\max(|\bar{x}|, z)) \leqslant y b_k(\max(|\bar{x}|, y)) \\ &\leqslant u b_k(u) \leqslant (b_k(u))^2 \leqslant 2^{2^{b_k(u)}} = b_{k+2}(\max(|\bar{x}|, y)). \end{aligned}$$

Случай ограниченного произведения рассматривается аналогично. □

Следствие. *Функция $f(n) = b_n(1)$ примитивно рекурсивна, но не элементарна.*

Доказательство. Функция $g(y) = 2^y$ примитивно рекурсивна. Тогда $f(y+1) = g(f(y))$, где $f(0) = 1$. Неэлементарность функции f очевидна: она растет быстрее любой функции $b_k(n)$. □

Рассмотрим теперь задачу о скорости вычисления элементарной функции. Пусть $f(\bar{x})$ — функция, которая вычисляется программой P. Определим функцию $t_P(\bar{x})$ равную числу шагов, за которое программа P вычисляет $f(\bar{x})$ (если значение $f(\bar{x})$ определено). (Функция $t_P(\bar{x})$ не определена, если значение $f(\bar{x})$ не определено.) Функция $t_P(\bar{x})$ вычислима (см. Упражнение 1).

Определение 11.2. Если для функции $f(\bar{x})$ существует вычисляющая ее программа P, такая, что $t_P(\bar{x}) \leqslant b_k(|\bar{x}|)$ для некоторого k, то мы будем говорить, что f вычисляется за „элементарное время“.

Теорема 11.3. *Если функция f элементарна, то она вычисляется за элементарное время.*

Сначала докажем вспомогательное утверждение.

Лемма 11.1. *Пусть элементарная функция $h(\bar{x}, y)$ определена рекурсией из функций $f(\bar{x})$ и $g(\bar{x}, y, z)$, которые можно вычислить за элементарное время. Тогда и h можно вычислить за элементарное время.*

Доказательство. Пусть программы F и G вычисляют функции f и g за элементарное время. Как работает программа H, вычисляющая функцию h? Пусть нам нужно найти $h(\bar{x}, y)$. Сначала вычисляем $f(\bar{x})$ за $t_F(\bar{x})$ шагов. Далее мы вычисляем $g(\bar{x}, 0, h(\bar{x}, 0))$ за $t_G(\bar{x}, 0, h(\bar{x}, 0))$ шагов, $g(\bar{x}, 1, h(\bar{x}, 1))$ за $t_G(\bar{x}, 1, h(\bar{x}, 1))$ шагов, и так далее, пока не вычислим $g(\bar{x}, y-1, h(\bar{x}, y-1))$ за $t_G(\bar{x}, y-1, h(\bar{x}, y-1))$ шагов. Так как h элементарная функция, то $h(\bar{x}, y) \leqslant b_r(\max(|\bar{x}|, y))$ для некоторого r. Так как функция g вычисляется за элементарное время, то $t_G(\bar{x}, y, z) \leqslant b_s(\max(|\bar{x}|, y, z))$ для некоторого s. Поэтому,

$$t_G(\bar{x}, y, h(\bar{x}, y)) \leqslant b_s(\max(|\bar{x}|, y, h(x, y))) \leqslant$$
$$\leqslant b_s(\max(|\bar{x}|, y, b_r(\max(|\bar{x}|, y)))) \leqslant b_s(b_r(\max(|\bar{x}|, y))) = b_{r+s}(\max(|\bar{x}|, y)).$$

Теперь

$$t_H(\bar{x}, y) \leqslant t_F(\bar{x}) + \sum_{i=0}^{y-1} b_{r+s}(\max(|\bar{x}, i)) \leqslant t_F(\bar{x}) + y b_{r+s}(\max(|\bar{x}|, y)).$$

Пусть $m = \max(|\bar{x}|, y)$ и $t_F(\bar{x}) \leqslant b_q(|\bar{x}|)$ для некоторого q. Тогда

$$t_H(\bar{x}, y) \leqslant b_q(m) + m b_{r+s}(m).$$

Далее рассуждаем как в доказательстве Теоремы 11.2 пункт 7. □

Доказательство Теоремы 11.3. Основные функции вычисляются программами за 1 шаг. Сложение, умножение и неполное вычитание определяются рекурсивно из более простых функций, так что здесь работает Лемма 11.1.

Подстановка. Пусть $h(\bar{x}) = f(g_1(\bar{x}), \ldots, g_m(\bar{x}))$, где функции $f, g_1, \ldots, g_m$ вычислимы за элементарное время программами $F, G_1, \ldots, G_m$. Обозначим через H программу, вычисляющую функцию h. Тогда

$$t_H(\bar{x}) = \sum_{i=1}^{m} T_{G_i}(\bar{x}) + t_F(g_1(\bar{x}), \ldots, g_m(\bar{x})).$$

Далее рассуждаем как в доказательстве Теоремы 11.2 пункт 6.

Ограниченные суммы и произведения имеют рекурсивное определение, так что здесь опять-таки работает Лемма 11.1. □

Докажем обратную теорему: если функция может быть вычислена за элементарное время, то она элементарна.

Лемма 11.2. *Пусть f — функция, вычисляемая программой P. Тогда функция состояния σ, а также функции c и j элементарны.*

Доказательство. Мы знаем, что функция состояния определяется рекурсией с использованием функций config и nxt. Достаточно доказать элементарность этих двух функций и ограниченность используемой рекурсии. Действительно, пусть N — количество регистров, используемых в программе P, $(a_1, \ldots, a_N)$ — исходная конфигурация, $(x_1(t), \ldots, x_N(t))$ — конфигурация на шаге t. Тогда $\max\{x_1(t), \ldots, x_N(t)\} \leqslant \max\{a_1, \ldots, a_N\} + t$. Поэтому,

$$c(\bar{x}, t) \leqslant \prod_{i=1}^{N} p_i^{X+t},$$

где $X = \max\{a_1, \ldots, a_N\}$. Функция в правой части неравенства элементарна. Так как $\sigma = N_2(c, j)$, где функция j просто ограничена, а c ограничена сверху элементарной функцией, то и функция состояния ограничена сверху элементарной функцией. Теперь лемма следует из Теоремы 1 и элементарности функций config и nxt (Упражнение 2). □

Следствие. *Пусть $f(\bar{x})$ — элементарная функция, а $g(\bar{x})$ — тотальная функция, вычисляемая программой G. Тогда, если $t_G(\bar{x}) \leqslant f(\bar{x})$, то функция g элементарна.*

Доказательство. Это просто. Мы знаем, что функция

$$k(\bar{x}) = \min_{t \leqslant f(\bar{x})} (j(\bar{x}, t))$$

элементарна (ограниченная минимизация элементарной функции). Теперь $g(\bar{x})$ определяется так: $g(\bar{x}) =$ показатель степени двойки в разложении числа $c(\bar{x}, k(\bar{x}))$ на простые множители. □

Теперь мы можем доказать теорему.

Теорема 11.4. *Пусть тотальная функция $f(\bar{x})$ вычисляется программой F и $t_F(\bar{x}) \leqslant b_k(|\bar{x}|)$ для некоторого k. Тогда f — элементарная функция.*

Доказательство. Так как функция $b_k(|\bar{x}|)$ элементарна, то осталось воспользоваться Следствием. □

Упражнения.

(1) Пусть программа P вычисляет функцию $f(\bar{x})$. Доказать вычислимость функции $t_P(\bar{x})$, равной числу шагов, за которое P вычисляет $f(\bar{x})$ (область определения функции t_P совпадает с областью определения функции f).
(2) Доказать элементарность функций config и nxt.

12. Сложность вычислений

Прежде чем говорить о „труднорешаемых задачах“ и о „эквивалентных по сложности задачах“, сначала нужно договориться о значении основных терминов.

Мы будем рассматривать *массовые задачи*, т.е. задачи с *параметрами* — свободными переменными, конкретные значения которых не определены. Индивидуальная задача I получается из массовой задачи П, если всем параметрам присвоить конкретные значения.

Пример. Сложение $x + y$ — массовая задача с параметрами x и y, сложение $2 + 2$ — индивидуальная задача.

Мы будем говорить, что алгоритм решает массовую задачу П, если он применим к любой индивидуальной задаче $I \in \Pi$ и обязательно дает ее решение. Время работы алгоритма удобно выражать в виде функции от одной переменной. Эта переменная характеризует „размер“ индивидуальной задачи, т.е. объем входных данных. В задаче о сложении двух чисел размером может быть, например, количество знаков в двоичной записи чисел x и y, т.е. $\log_2(x)+\log_2(y)$. *Полиномиальным алгоритмом* или *алгоритмом полиномиальной временной сложности* называется алгоритм, временная сложность которого равна $O(p(n))$, где $p(n)$ — многочлен, а n — входная длина. Алгоритмы, временная сложность которых не поддается такой оценке, называются *экспоненциальными*.

Большинство экспоненциальных алгоритмов — это варианты полного перебора. Нетривиальный полиномиальный алгоритм удается построить лишь тогда, когда удается проникнуть в суть решаемой задачи.

Замечание. Многие стандартные вычислительные задачи полиномиальны по определению. К таким задачам относится, например, задача о приведении матрицы к главному ступенчатому виду или задача об умножении матриц. Но вычисление собственных чисел на первый взгляд не кажется полиномиальной задачей.

Замечание. Хотя симплекс метод — это экспоненциальный алгоритм, но лишь в „малом числе“ задач он демонстрирует экспоненциальную временную сложность. Как правило, симплекс метод дает ответ в удовлетворительное время.

Мы будем называть массовую задачу *труднорешаемой*, если для ее решения не существует полиномиального алгоритма.

Первые результаты о труднорешаемости были получены А. Тьюрингом в 1936г. Он доказал, что некоторые задачи неразрешимы, в том смысле, что вообще не существует алгоритма их решения. Например, именно он доказал, что задача об остановке работы программы неразрешима. С этой задачей мы уже встречались. Примеры разрешимых, но принципиально труднорешаемых задач были построены в 1960-х гг. Однако эти примеры включали только „искусственные“ задачи, специально построенные, чтобы обладать нужными свойствами. Эти задачи не являются даже NP-задачами, т.е. вообще нереалистичны — для такой задачи объем выходных данных (ответ) экспоненциально растет с ростом объема входных данных. (Понятие NP-задачи является центральным в теории и мы будем его обсуждать ниже.)

Из соображений удобства мы будем рассматривать только задачи о распознавании свойств. Ответом на такую задачу является „да“ или „нет“. На самом деле практически любую задачу можно переформулировать в таком виде.

Пример. Изоморфизм подграфу. Дано два графа G_1 и G_2. Верно ли, что граф G_1 содержит подграф, изоморфный G_2?

Пример. Коммивояжер. Дано конечное множество „городов“ $C = \{c_1, \ldots, c_n\}$. Для каждой пары городов дано расстояние $d(c_i, c_j)$ между ними. Кроме того, задано положительное число B. Существует ли маршрут, проходящий через все города, длина которого не превосходит B?

13. Машина Тьюринга

Сложность алгоритмов мы будем описывать на языке *машин Тьюринга*. Машиной Тьюринга (МТ) называется устройство, состоящее из

- бесконечной ленты, разделенной на ячейки (регистры), пронумерованные целыми числами от $-\infty$ до $+\infty$, причем в каждой ячейке хранится 0 или 1;
- считывающего/печатающего устройства;

- собственно управляющей машины, которая может находиться в конечном числе состояний и занимать любое положение относительно ленты.

В данный момент МТ считывает лишь одну ячейку, (ту где находится машина). Такт работы МТ состоит из следующих процедур:

(1) МТ считывает содержимое ячейки. В зависимости от содержимого ячейки и состояния, в котором машина находится, она
(2) печатает в этой ячейке 0 или 1;
(3) сдвигается вправо, влево или прекращает работу;
(4) переходит в другое состояние.

О МТ мы говорим, что она вычисляет одноместную функцию $f(n)$, когда МТ работает так: на вход подается лента, во всех ячейках которой стоят нули, кроме блока из n единиц, причем в начальном состоянии МТ считывает ячейку слева от блока единиц; если n принадлежит области определения функции $f(n)$, то МТ останавливается и во всех ячейках ленты стоят нули, кроме блока из $f(n)$ единиц; если n не принадлежит области определения функции $f(n)$, то МТ не останавливается.

Пример. Следующая задача хорошо известна. Пусть в начальном состоянии во всех регистрах ленты находятся нули. Обозначим через A_n множество всех МТ с n состояниями, которые останавливаются для этого начального состояния. В множестве A_n выделим подмножество B_n, состоящее из тех МТ, результатом работы которых будет напечатанный на ленте сплошной блок из единиц. Если $M \in B_n$, то через s_M мы обозначим длину этого блока. Определим функцию

$$g(n) = \max_{M \in B_n} s_M.$$

Аналогичную функцию для машины с неограниченными регистрами мы уже рассматривали. Как и в случае машины с неограниченными регистрами, функция g невычислима. Задача состоит в определении значений $g(n)$ при малых n.

Известно, что $g(4) = 6$, а $g(5) = 11$. Вот пример МТ с четырьмя состояниями, для которой $s(M) = 6$:

1	2	3	4
1	1	1	1
→	←	←	←
2	3	1	2
0	1	1	1
←	←	←	→
1	4	4	стоп

В таблице указано, что делает МТ в каждом состоянии (номер состояния указан в первой строке), когда она считывает 0 (вторая строчка) или 1 (третья строчка): например, в третьей клетке второй строки указано, что МТ в третьем состоянии при считывании нуля печатает единицу, сдвигается влево и переходит в первое состояние.

Упражнения.

(1) Построить машину Тьюринга $M \in B_5$, для которой $s_M = 11$. Это не простая задача.
(2) Построить МТ, вычисляющие функции $f(n) = 2n$, $f(n) = n \mod 2$ (остаток от деления на 2), $f(n) = n/2$ (это частичная функция).

14. Недетерминированное вычисление и класс NP

Задачу, которая может быть решена полиномиальным алгоритмом, мы будем называть P-задачей. Определим другой класс — класс NP-задач. Сначала дадим пояснения.

Рассмотрим задачу о коммивояжере. Полиномиальный алгоритм решения этой задачи неизвестен. Однако, любое гипотетическое решение легко проверяется. Для этого нужно проверить, что маршрут проходит через все города, и найти его длину. Очевидно, что процедуру проверки можно реализовать в виде алгоритма, временная сложность которого полиномиально зависит от объема входных данных.

Рассмотрим задачу об изоморфизме подграфу. Пусть даны два графа $G_1 = (V_1, E_1)$ и $G_2 = (V_2, E_2)$ (здесь V_i и E_i, $i = 1, 2$, — множества вершин и ребер). Мы ищем подграф $G = (V, E)$ в G_1, $V \subseteq V_1$, $E \subseteq E_1$, изоморфный графу G_2. Если ответом является "да", то предъявляются множества V, E и изоморфизм $f : G_2 \to G$. Алгоритм проверки также имеет полиномиальную временную сложность.

Понятие полиномиальной проверяемости позволяет определить класс NP. Этот класс задач определяется с помощью *недетерминированного алгоритма*. Такой алгоритм состоит из двух блоков: *блока угадывания* и *блока проверки*. По данной индивидуальной задаче первый блок "угадывает" ответ. После чего ответ поступает в блок проверки, который представляет собой обычный детерминированный алгоритм.

Недетерминированный алгоритм, решающий задачу распознавания Π, полиномиален, если найдется многочлен p, такой, что для любой индивидульной задачи $I \in \Pi$ догадка S проверяется за время $p(|I|)$. Отсюда, в частности, следует, что "объем" догадки ограничен некоторым многочленом от $|I|$.

Класс NP — это класс всех задач распознавания П, которые при разумном кодировании могут быть решены недетерминированными (N — nondeterministic) алгоритмами за полиномиальное (P — polynomial) время.

Мы будем считать, что недетерминированные алгоритмы реализуются недетерминированными машинами Тьюринга (НДТМ). НДТМ состоит из двух модулей: угадывающий модуль записывает ответ на ленту в виде входного слова, начиная с первой ячейки, проверяющий модуль — обычная МТ, выполняет проверку и останавливается в состоянии q_Y ("да") или q_N ("нет").

15. Связь между классами P и NP

Легко видеть, что $\mathrm{P} \subseteq \mathrm{NP}$. Действительно, полиномиальный алгоритм предъявляет ответ (т.е. проверка включена в ответ) за полиномиальное время. Справедлива следующая теорема.

Теорема 15.1. *Пусть* $\Pi \in NP$*, тогда существует многочлен* p*, такой, что задача* Π *может быть решена детерминированным алгоритмом за время* $O(2^{p(n)})$.

Доказательство. По определению недетерминированного алгоритма для каждой индивидуальной задачи I объема n найдется догадка объема $q(n)$, причем стадия проверки требует $s(n)$ шагов. Таким образом, всего имеется $2^{q(n)}$ ответов на индивидуальную задачу I, а общее время проверки всех ответов не превышает $s(n)2^{q(n)}$ шагов. Так как $x < 2^x$, то $s(n)2^{q(n)} < 2^{s(n)+q(n)} = 2^{p(n)}$. □

Предполагается, что $\mathrm{P} \neq \mathrm{NP}$. Хотя это утверждение не доказано, но можно указать задачи, *не являющиеся* P-задачами, если $\mathrm{P} \neq \mathrm{NP}$. Такого рода условные утверждения основаны на понятии *полиномиальной сводимости.*

Мы говорим, что задача Π_1 полиномиально сводима к задаче Π_2, если любую индивидуальную задачу $I_1 \in \Pi_1$, $|I_1| = n$, за время $O(p(n))$ можно преобразовать в индивидуальную задачу $I_2 \in \Pi_2$, причем ответом на задачу I_1 будет "да" в том и только том случае, когда ответом на задачу I_2 также будет "да". Полиномиальную сводимость мы будем обозначать так: $\Pi_1 \vdash \Pi_2$.

Лемма 15.1. *Если* $\Pi_1 \vdash \Pi_2$ *и* $\Pi_2 \vdash \Pi_3$*, то* $\Pi_1 \vdash \Pi_3$.

Доказательство. Очевидно. □

Лемма 15.2. *Пусть* $\Pi_1 \vdash \Pi_2$*. Тогда, если* $\Pi_2 \in P$*, то и* $\Pi_1 \in P$*, а если* $\Pi_1 \notin P$*, то и* $\Pi_2 \notin P$.

Доказательство. Очевидно. □

Рассмотрим важный пример сводимости.

Гамильтоновым циклом в графе G называется простой цикл, содержащий все вершины G. Задача „гамильтонов цикл" формулируется так: задан граф $G = (V, E)$. Содержит ли G гамильтонов цикл?

Предложение 15.1. *„Гамильтонов цикл" ⊢ „коммивояжер".*

Доказательство. Пусть $G = (V, E)$, $|V| = m$ — индивидуальная задача из ГЦ. Соответствующая задача из К строится так: множество городов совпадает с V. Для любых двух городов $v_i, v_j \in V$ мы полагаем

$$d(v_i, v_j) = \begin{cases} 1, & \text{если } \{v_i, v_j\} \in E \\ 2, & \text{в противном случае.} \end{cases}$$

Верхнюю границу B длины искомого маршрута мы полагаем равной m. Тогда замкнутый маршрут длины $\leqslant m$, проходящий через все города, и будет гамильтоновым циклом. □

Дадим важное определение.

Определение 15.1. Задача $\Pi \in \mathrm{NP}$ называется *NP-полной*, если к ней сводится любая задача $\Pi' \in \mathrm{NP}$.

Лемма 15.3. *Если NP-полная задача полиномиальна, то любая NP-задача полиномиальна.*

Существование NP-полных задач — удивительный факт. Мы сначала докажем, что, так называемая, *задача о выполнимости* NP-полна (это утверждение называется теоремой Кука), а потом докажем NP-полноту ряда других задач (в том числе и задачи ГЦ).

Задача о выполнимости. Пусть дан набор булевских переменных $x_1, \ldots, x_n$. Дизъюнкцией назовем выражение $y_{i_1} \vee \ldots \vee y_{i_m}$, где $y_i = x_i$ или $y_i = \overline{x_i}$. Формулировка задачи такова: пусть задан набор дизъюнкций C. Существует ли набор значений истинности переменных x_i, для которого все дизъюнкции из C истинны?

Замечание. Если класс NP строго больше класса P, то можно доказать, что существуют неполиномиальные задачи, не являющиеся NP-полными. К числу таких задач предположительно относятся задача об изоморфизме графов и задача о разложении натурального числа на простые множители.

Упражнения.

(1) Как можно организовать с помощью МТ проверку догадки для задачи „выполнимость"?
(2) Как можно организовать с помощью МТ проверку догадки для задачи „гамильтонов цикл"?

16. Теорема Кука

Этот раздел посвящен доказательству теоремы Кука.

Теорема Кука. *Задача „выполнимость“ NP-полна.*

Доказательство. Принадлежность задачи „выполнимость“ классу NP очевидна. Докажем NP-полноту.

Пусть у нас есть некоторая задача $\Pi \in NP$ и машина Тьюринга с полиномиальным временем работы, проверяющая „догадку“. Мы будем считать, что эта МТ имеет Q состояний $q_1, \ldots, q_r$ и еще 3 выделенных состояния q_0, q_Y, q_N — начальное и два конечных. Работа МТ описывается функцией перехода δ, зависящей от состояния и содержимого ячейки. МТ переходит в состояние q_Y и останавливается, если догадка верна, и переходит в состояние q_N и останавливается, если нет. Мы будем рассматривать случай „принимающего“ вычисления, когда догадка верна.

Функция f, реализующая сводимость, будет описана в терминах состояний, номеров ячеек и номера шага вычисления. Если индивидуальная задача $I \in \Pi$ имеет размер $|I|$, то МТ осуществляет проверку за $n = p(|I|)$ шагов (где p — некоторый многочлен). Из этого следует, что МТ работает не более, чем $2n+1$ ячейками ленты с номерами $-n, \ldots, n$. Кроме того, количество состояний r не превышает числа n.

Перечислим булевские переменные:

- $Q[i,k]$, $0 \leqslant i \leqslant n$, $0 \leqslant k \leqslant r$: в момент времени i МТ находиться в состоянии q_k;
- $H[i,j]$, $0 \leqslant i \leqslant n$, $-n \leqslant j \leqslant n$: в момент времени i МТ читает ячейку с номером j;
- $S[i,j,0]$, $0 \leqslant i \leqslant n$, $-n \leqslant j \leqslant n$: в момент времени i в ячейке с номером j находится 0.
- $S[i,j,1]$, $0 \leqslant i \leqslant n$, $-n \leqslant j \leqslant n$: в момент времени i в ячейке с номером j находится 1.

Наша задача состоит в описании множества дизъюнкций для этих переменных, для которого множество значений переменных будет выполняющим в том и только том случае, когда этот набор значений генерируется принимающим вычислением.

Множество дизъюнкций состоит из 6 групп:

G_1 : в любой момент времени i МТ находится ровно в одном состоянии;
G_2 : в любой момент времени i МТ просматривает ровно одну ячейку;
G_3 : в любой момент времени i каждая ячейка содержит ровно один символ (0 или 1);
G_4 : в момент времени 0 МТ находится в состоянии q_0;
G_5 : не позднее чем через n шагов МТ переходит в состояние q_Y;
G_6 : для любого момента времени i, $0 \leqslant i \leqslant n$, конфигурация МТ в момент времени $i+1$ получается из конфигурации в момент времени i одноразовым применением функции перехода δ.

Легко видеть, что эти шесть групп дизъюнкций действительно выполняются, если вычисление принимает догадку. Теперь опишем эти дизъюнкции.

Группа G_1 состоит из дизъюнкций вида $Q[i,0] \vee \ldots \vee Q[i,r]$, $0 \leqslant i \leqslant n$, и дизъюнкций вида $\overline{Q[i,j]} \vee \overline{Q[i,j']}$, $0 \leqslant i \leqslant n$, $0 \leqslant j < j' < r$. Первые $n+1$ дизъюнкций выполняются, если в каждый момент времени МТ находится в каком-то состоянии, остальные дизъюнкции (их $r(r+1)(n+1)/2$) выполняются, если МТ *не* находится одновременно в двух состояниях.

Группы G_2 и G_3 строятся аналогично. Группы G_4 и G_5 очень просты. Несколько сложнее выглядит описание группы G_6, которая гарантирует, что каждая следующая конфигурация получается из предыдущей в результате применения одной команды МТ. Эта группа дизъюнкций состоит из двух подгрупп.

Первая подгруппа гарантирует, что если МТ в момент i *не* просматривает ячейку с номером j, то число в этой ячейке в момент $i+1$ будет тем же, что и в момент i. Это дизъюнкции вида $\overline{S[i,j,0]} \vee H[i,j] \vee S[i+1,j,0]$, $\overline{S[i,j,1]} \vee H[i,j] \vee S[i+1,j,1]$, $0 \leqslant i \leqslant n$, $-n \leqslant j \leqslant n$. Подробнее: первая дизъюнкция не выполняется, если в момент i в ячейке j находится 0, МТ не просматривает ячейку j в момент i и в момент $i+1$ в ячейке j находится 1.

Вторая подгруппа дизъюнкций гарантирует, что переход в другую конфигурацию определен функцией перехода δ. Для каждой тройки

$$(i,j,k),\ 0 \leqslant i \leqslant n,\ -n \leqslant j \leqslant n,\ 0 \leqslant k \leqslant r,$$

эта подгруппа содержит дизъюнкции

$$\overline{H[i,j]} \vee \overline{Q[i,k]} \vee \overline{S[i,j,0]} \vee H[i+1, j+\Delta_0],$$
$$\overline{H[i,j]} \vee \overline{Q[i,k]} \vee \overline{S[i,j,1]} \vee H[i+1, j+\Delta_1],$$
$$\overline{H[i,j]} \vee \overline{Q[i,k]} \vee \overline{S[i,j,0]} \vee Q[i+1, k_0],$$
$$\overline{H[i,j]} \vee \overline{Q[i,k]} \vee \overline{S[i,j,1]} \vee Q[i+1, k_1],$$
$$\overline{H[i,j]} \vee \overline{Q[i,k]} \vee \overline{S[i,j,0]} \vee S[i+1, j, s_0],$$
$$\overline{H[i,j]} \vee \overline{Q[i,k]} \vee \overline{S[i,j,1]} \vee S[i+1, j, s_1],$$

где $\delta(k,0) = (k_0, s_0, \Delta_0)$, $\delta(k,1) = (k_1, s_1, \Delta_1)$,а $\Delta_{0/1} = \pm 1$. Дизъюнкция $\overline{H[i,j]} \vee \overline{Q[i,k]} \vee \overline{S[i,j,0]} \vee H[i+1, j+\Delta_0]$, например, не выполняется, если в момент i МТ считывает ячейку j в состоянии k, причем в ячейке находится 0, а в момент $i+1$ МТ считывает содержимое некоторой ячейки, номер которой отличен от числа $j + \Delta_0$, заданного функцией перехода.

Все это и дает нам требуемую сводимость. □

Упражнения.

(1) Оцените количество булевских переменных и дизъюнкций в процедуре сведения произвольной задачи к задаче „выполнимость“. Покажите, что сведение действительно полиномиально.

17. Шесть основных NP-полных задач

В этом разделе мы приведем шесть задач, NP-полноту которых мы будем доказывать.

(1) „3-выполнимость“. Дан набор булевских переменных $x_1, \ldots, x_n$ и множество дизъюнкций вида $y_{i_1} \vee y_{i_2} \vee y_{i_3}$, где y_j — это x_j или $\overline{x_j}$. Существует ли набор значений истинности, при котором выполняются все эти дизъюнкции?
(2) „Трехмерное сочетание“. Дано множество $M \subseteq X \times Y \times Z$, где X, Y, Z — непересекающиеся множества, причем $|X| = |Y| = |Z| = q$. Верно ли, что M содержит трехмерное сочетание? То-есть подмножество $M' \subseteq M$ такое, что $|M'| = q$, и если (x_1, y_1, z_1), (x_2, y_2, z_2) — различные элементы из M', то $x_1 \neq x_2$, $y_1 \neq y_2$, $z_1 \neq z_2$.
(3) „Вершинное покрытие“. Дан граф $G = (V, E)$ и положительное целое число k, $k \leqslant |V|$. Существует ли вершинное покрытие не более чем из k элементов, т.е. подмножество $V' \subseteq V$, $|V'| \leqslant k$, такое, что для каждого ребра из E хотя бы один из его концов принадлежит V'?
(4) „Клика“. Дан граф $G = (V, E)$ и положительное число целое k, $k \leqslant |V|$. Верно ли, что G содержит клику мощности $\geqslant k$, т.е. подмножество $V' \subseteq V$, $|V'| \geqslant k$, такое, что любые две вершины из V' соединены ребром.
(5) „Гамильтонов цикл“.
(6) „Разбиение“. Даны конечное множество A и „вес“ $s(a)$ каждого элемента $a \in A$. Существует ли подмножество $A' \subset A$, такое, что

$$\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a)?$$

18. Задача „3-выполнимость“ NP-полна

Доказывать NP-полноту основных задач начнем с задачи „3-выполнимость“. Докажем сведение „выполнимость“ $\vdash$ „3-выполнимость“.

Пусть $X = \{x_1, \ldots, x_n\}$ — множество булевских переменных, а $C = \{c_1, \ldots, c_m\}$ — некоторый набор дизъюнкций, которые определяют индивидуальную задачу „выполнимость“. Мы построим набор C' 3-дизъюнкций на множестве переменных $X' = X \cup X'_1 \cup, \ldots, \cup X'_m$. Каждую дизъюнкцию $c_i \in C$, $c = y_1 \vee \ldots \vee y_k$, мы заменим набором 3-дизъюнкций C'_i от переменных $\{x_1, \ldots, x_k\} \cup X'_i$. А именно:

- $k = 1$, $c_i = y_1$. $X'_i = \{z_i^1, z_i^2\}$,
$$C'_i = \{y_1 \vee z_i^1 \vee z_i^2,\ y_1 \vee z_i^1 \vee \overline{z_i^2},\ y_1 \vee \overline{z_i^1} \vee z_i^2,\ y_1 \vee \overline{z_i^1} \vee \overline{z_i^2}\}.$$
- $k = 2$, $c_i = y_1 \vee y_2$. $X'_i = \{z_i\}$,
$$C'_i = \{y_1 \vee y_2 \vee z_i,\ y_1 \vee y_2 \vee \overline{z_i}\}.$$
- $k = 3$, $c_i = y_1 \vee y_2 \vee y_3$. $C'_i = \{c_i\}$.
- $k > 3$. $X'_i = \{z_i^1, \ldots, z_i^{k-3}\}$,
$$C'_i = \{y_1 \vee y_2 \vee z_i^1,\ \overline{z_i^1} \vee y_3 \vee z_i^2,\ \ldots,\ \overline{z_i^{k-4}} \vee y_{k-2} \vee z_i^{k-3},\ \overline{z_i^{k-3}} \vee y_{k-1} \vee y_k\}.$$

Тогда $|X'| \leqslant n + mn$, а $|C'| \leqslant mn$, т.е. количество переменных и дизъюнкций в 3-задаче полиномиально зависит от m и n.

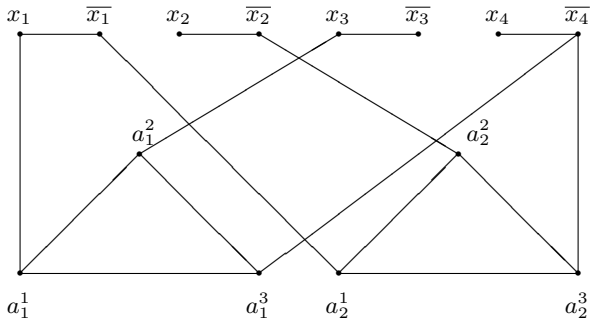
Пусть исходный набор дизъюнкций выполним для некоторого набора значений переменных x_i. Рассмотрим дизъюнкцию $c_i = y_1 \vee \ldots \vee y_k \in C$. И пусть, например, $y_k = 1$. Тогда при $z_i^1 = \cdots = z_i^{k-3} = 1$ выполним и набор дизъюнкций C'_i. Обратно, пусть выполнимы все дизъюнкции из множества C' при некоторых значениях переменных x_i, z_r^j. Покажем, что при тех же значениях переменных x_i выполнимы и все дизъюнкции из множества C. Предположим, что дизъюнкция $c_i = y_1 \vee \ldots \vee y_k \in C$ не выполнима. Это означает, что $y_1 = \cdots = y_k = 0$. Но тогда из выполнимости C'_i следует, что $z_i^1 = \cdots = z_i^{k-3} = 1$, что, в конечном счете, дает невыполнимость дизъюнкции $\overline{z_i^{k-3}} \vee y_{k-1} \vee y_k$. Противоречие.

19. Задача „вершинное покрытие“ NP-полна

Докажем, что имеет место сведение „3-выполнимость“ $\vdash$ „вершинное покрытие“.

Пусть $X = \{x_1, \ldots, x_n\}$ — множество булевских переменных, а $C = \{c_1, \ldots, c_m\}$ — некоторый набор 3-дизъюнкций, которые определяют индивидуальную задачу „3-выполнимость“. Нам нужно найти граф $G = (V, E)$ и число $K \leqslant |V|$ такие, что G имеет вершинное покрытие с числом элементов не более K в том и только том случае, когда выполним набор дизъюнкций C. Каждой переменной x_i мы сопоставим две вершины (обозначим их x_i и $\overline{x_i}$) и соединяющее их ребро. Каждой дизъюнкции $c_i = y_1 \vee y_2 \vee y_3 \in C$ мы сопоставим три вершины a_i^1, a_i^2, a_i^3, ребра $\{a_i^1, a_i^2\}$, $\{a_i^1, a_i^3\}$, $\{a_i^2, a_i^3\}$ и ребра $\{a_i^1, y_1\}$, $\{a_i^2, y_2\}$, $\{a_i^3, y_3\}$ (напоминаем, что y_i — это x_i или $\overline{x_i}$). И, наконец, положим $K = n + 2m$.

Пример. Пусть $X = \{x_1, x_2, x_3, x_4\}$, $C = \{x_1 \vee x_3 \vee \overline{x_4},\ \overline{x_1} \vee \overline{x_2} \vee \overline{x_4}\}$. Тогда граф G имеет вид:



Пусть $V' \subseteq V$, $|V'| \leqslant K$ — вершинное покрытие графа G. Тогда V' содержит хотя бы одну вершину из каждой пары $x_i, \overline{x_i}$ и хотя бы две вершины из каждого треугольника a_i^1, a_i^2, a_i^3. Но $K = n+2m$. Это означает, что V' содержит *ровно одну* вершину из каждой пары и *ровно две* вершины из каждого треугольника. Теперь положим $x_i = 1$, если $x_i \in V'$, и $x_i = 0$, если $\overline{x_i} \in V'$. Рассмотрим некоторую дизъюнкцию c_i и соответствующий треугольник a_i^1, a_i^2, a_i^3. В этом треугольнике *лишь две* вершины принадлежат V'. У ребра, соединяющего третью вершину с некоторой вершиной y_j, одна из концевых вершин должна принадлежать V'. Так как этой вершиной не является вершина треугольника, то $y_j \in V'$. То-есть $y_j = 1$ и дизъюнкция c_i выполняется.

Обратно. Пусть некоторый набор значений переменных x_i выполняет все дизъюнкции из множества C. Мы включаем в покрытие V' вершину x_i, если $x_i = 1$, и вершину $\overline{x_i}$, если $x_i = 0$. Так как все дизъюнкции выполнены, то это обеспечивает покрытие одного ребра, соединяющего каждый треугольник с вершинами $x_i, \overline{x_i}$. Осталось включить в V' две другие вершины каждого треугольника.

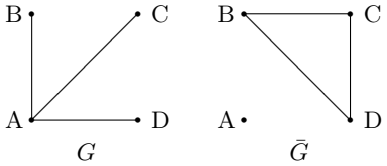
Упражнения.

(1) Покажите, что сведение „3-выполнимость“ $\vdash$ „вершинное покрытие“ действительно полиномиально.

20. Задача „клика“ NP-полна

Докажем, что имеет место сведение „вершинное покрытие“ $\vdash$ „клика“.

Пусть $G = (V, E)$ — некоторый граф и пусть число $k \leqslant |V|$ задает индивидуальную задачу „вершинное покрытие“. Рассмотрим граф $\bar{G} = (V, \bar{E})$, где $\bar{E} = \{\{u, v\} : u, v \in V, \{u, v\} \notin E\}$ (граф $\bar{G}$ представляет собой „дополнение“ графа G):



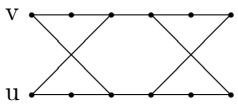
Соответствующая индивидуальная задача „клика“ задается графом $\bar{G}$ и числом $l = |V| - k$. Действительно, пусть множество $V' \subseteq V$, $|V'| \geqslant l$ образует клику в графе $\bar{G}$. Покажем, что множество $V'' = V \setminus V'$, $|V''| = |V| - |V'| \leqslant k$, образует вершинное покрытие графа G. Преположим противное: найдется ребро $\{u, v\} \in E$, причем $u, v \notin V''$. Но тогда $u, v \in V'$, а так как вершины u, v соединены ребром в графе $\bar{G}$, то $\{u, v\} \notin E$. Противоречие. **Упражнения.**

(1) Покажите, что сведение „вершинное покрытие“ $\vdash$ „клика“ действительно полиномиально.

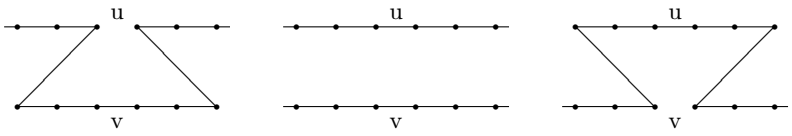
21. Задача „гамильтонов цикл“ NP-полна

Докажем, что имеет место сведение „вершинное покрытие“ $\vdash$ „гамильтонов цикл“.

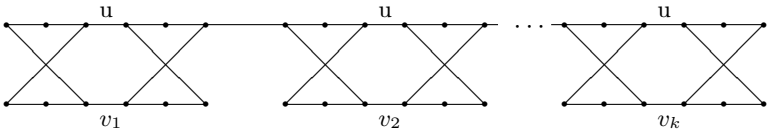
Пусть $G = (V, E)$ — граф, в котором нужно найти вершинное покрытие с числом вершин K. Построим вспомогательный граф $G' = (V', E')$, в котором мы будем искать гамильтонов цикл. В графе G' имеется K выделенных вершин $a_1, \ldots, a_K$, а каждому ребру $e = (v, u) \in E$ сопоставляется „решетка“ — конструкция из 12 вершин и 14 ребер:



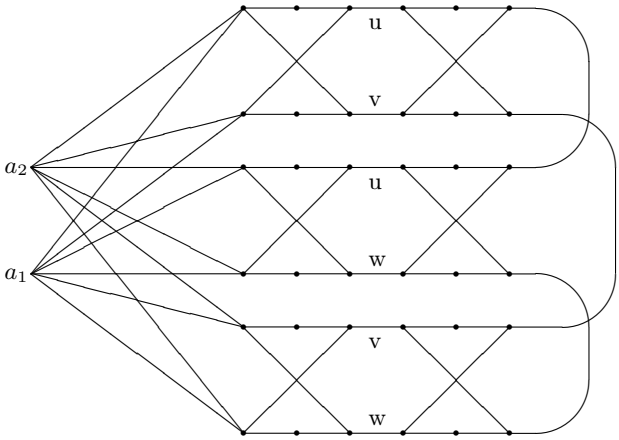
Здесь нижний ряд вершин отвечает вершине u, а верхний — вершине v. Гамильтонов цикл в графе G' должен проходить через все вершины каждой решетки, причем один раз. Это можно сделать только тремя способами.



Рассмотрим все ребра (u, v_1),...,(u, v_k) выходящие из вершины u. Этим ребрам отвечают решетки со сторонами u и v_i, $i = 1, \ldots, k$. Соединим в любом порядке стороны, отмеченные буквой „u“, чтобы решетки, отвечающие ребрам (u, v_i), образовали цепочку:



Проделаем такую операцию со всеми вершинами графа G. Теперь у каждой цепочки есть две „свободные“ концевые вершины. Соединим их со всеми вершинами $a_1, \ldots, a_K$. Вот как выглядит граф G' в случае графа $G = (V, E)$, где $V = \{u, v, w\}$, $E = \{(u, v), (u, w), (v, w)\}$ и $K = 2$:



Докажем, что граф G имеет вершинное K-покрытие в том и только том случае, когда граф G' имеет гамильтонов цикл. Пусть в графе G' есть гамильтонов цикл. Тогда он проходит через все вершины $a_1, \ldots, a_K$ по одному разу. Рассмотрим фрагмент цикла между двумя вершинами a_i и a_j, который не содержит вершин из множества $a_1, \ldots, a_K$. Тогда по свойству решеток здесь цикл проходит по цепочке, отвечающей некоторой вершине v графа G. У нас имеется K фрагментов цикла и, значит, K вершин $v_1, \ldots, v_K$. Эти вершины и образуют K-вершинное покрытие. Действительно, ребру e графа G отвечает решетка. Через эту решетку проходит цикл. Значит эта решетка входит в цепочку, отвечающую некоторой вершине v_i. Но по определению решетки эта вершина является концевой для ребра e.

Обратно. Пусть $V' = \{v_1, \ldots, v_K\}$ — K-вершинное покрытие графа G (если количество вершин покрытия меньше K, то просто добавим вершины). Теперь проведем цикл через цепочки, отвечающие вершинам v_i и замкнем их через вершины $a_1, \ldots, a_K$. Так V' — вершинное покрытие, то цикл пройдет через все решетки.

Упражнения.

(1) Покажите, что сведение „вершинное покрытие“ $\vdash$ „гамильтонов цикл“ действительно полиномиально.

22. Задача „трехмерное сочетание“ NP-полна

Докажем, что имеет место сведение „3 выполнимость“ $\vdash$ „трехмерное сочетание“.

Пусть $X = \{x_1, \ldots, x_n\}$ — множество булевских переменных, а $C = \{c_1, \ldots, c_m\}$ — множество 3-дизъюнкций. Мы построим три множества P, Q и R и подмножество $M \subset P \times Q \times R$, в котором мы будем искать трехмерное сочетание.

Множество P состоит из элементов x_i^j, $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant m$, и элементов $\bar{x}_i^j$, $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant m$. Таким образом, $|P| = 2mn$.

Множество Q состоит из элементов a_i^j, $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant m$, элементов s_1^j, $1 \leqslant j \leqslant m$, и элементов g_1^k, $1 \leqslant k \leqslant m(n-1)$. Таким образом, $|Q| = mn + m + m(n-1) = 2mn$.

Множество R состоит из элементов b_i^j, $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant m$, элементов s_2^j, $1 \leqslant j \leqslant m$, и элементов g_2^k, $1 \leqslant k \leqslant m(n-1)$. Таким образом, $|R| = mn + m + m(n-1) = 2mn$.

Элементом множества M является тройка (m_1, m_2, m_3), где $m_1 \in P$, $m_2 \in Q$ и $m_3 \in R$. В M входят следующие тройки:

(1) $(\bar{x}_i^j, a_i^j, b_i^j)$, $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant m$;
(2) $(x_i^j, a_i^{j+1}, b_i^j$, $1 \leqslant i \leqslant n$, $1 \leqslant j < m$, и тройки (x_i^m, a_i^1, b_i^m), $1 \leqslant i \leqslant n$;
(3) для каждой дизъюнкции $c_j \in C$, $c_j = y_1 \vee y_2 \vee y_3$, тройки (y_1^j, s_1^j, s_2^j), (y_2^j, s_1^j, s_2^j), (y_3^j, s_1^j, s_2^j);
(4) (x_i^j, g_1^k, g_2^k), $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant m$, $1 \leqslant k \leqslant m(n-1)$;
(5) $(\bar{x}_i^j, g_1^k, g_2^k)$, $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant m$, $1 \leqslant k \leqslant m(n-1)$.

Предположим, что набор 3-дизъюнкций C выполним. Это означает, что существуют значения булевских переменных, при которых каждая дизъюнкция равна 1. Трехмерное сочетание M' будет состоять из следующих троек:

- троек из первой группы, для которых $x_i = 1$;
- троек из второй группы, для которых $x_i = 0$;
- в каждой дизъюнкции $c_j = y_1 \vee y_2 \vee y_3$ есть переменная равна 1 (например, y_1): в M' мы включаем тройку (y_1^j, s_1^j, s_2^j);
- тройки вида (x_i^j, g_1^k, g_2^k) или $(\bar{x}_i^j, g_1^k, g_2^k)$ с теми элементами x_i^j или $\bar{x}_i^j$, которые не попали в предыдущие группы.

Найдем $|M'|$. Троек из первой группы, для которых $x_i = 1$, и троек из второй группы, для которых $x_i = 0$, всего mn. Пусть $c_j = y_1 \vee y_2 \vee y_3$ — некоторая дизъюнкция, в которой $y_1 = 1$. Если $y_1 = x_1$, то первая координата тройки (y_1^j, s_1^j, s_2^j) не может быть равна первой координате любой тройки из второй группы. Если $y_1 = \bar{x}_1$, то первая координата тройки (y_1^j, s_1^j, s_2^j) не может быть равна первой координате любой тройки из первой группы. Всего таких троек m. Таким образом, не использованных элементов из множества P осталось $2mn - mn - m = m(n-1)$, т.е. столько, сколько элементов g_1^k или g_2^k. Перенумеруем их $z_1, \ldots, z_{m(n-1)}$. Теперь включим в M' тройки вида (z_k, g_1^k, g_2^k), $1 \leqslant k \leqslant m(n-1)$. Трехмерное сочетание построено.

Обратное утверждение доказывается аналогично.

Упражнения.

(1) Покажите, что сведение „3-выполнимость“ $\vdash$ „трехмерное сочетание“ действительно полиномиально.

23. Задача „разбиение“ NP-полна

Докажем, что имеет место сведение „трехмерное сочетание“ $\vdash$ „разбиение“.

Пусть $|P| = |Q| = |R| = n$, $P = \{p_1, \ldots, p_n\}$, $Q = \{q_1, \ldots, q_n\}$, $R = \{r_1, \ldots, r_n\}$, и пусть задано подмножество $M \subset P \times Q \times R$, $|M| = k$. Нам нужно построить множество A и задать веса $s(a)$ всех элементов из A так, чтобы A содержало подмножество $A' \subset A$, для которого

$$\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a)$$

в том и только том случае, когда M содержит трехмерное сочетание.

Множество A будет содержать $k+2$ элемента. Элементы $a_1, \ldots, a_k$ из A отвечают тройкам $m_1, \ldots, m_k$ из M. Пусть $m_i = (p_{f(i)}, q_{q(i)}, r_{h(i)})$ — некоторая тройка и пусть $c = [\log_2(k+1)] + 1$. Положим

$$s(a_i) = 2^{c(3n-f(i))} + 2^{c(2n-g(i))} + 2^{c(n-h(i))}.$$

Здесь важно отметить, что $k \cdot 2^{c(ln-j)} < 2^{c(ln-j+1)}$, $l = 1, 2, 3$. Из этого следует, что подмножество $A' \subset \{a_1, \ldots, a_k\}$ однозначно восстанавливается по сумме весов его элементов. Поэтому, если

$$\sum_{a \in A'} s(a) = B = \sum_{j=0}^{3n-1} 2^{cj},$$

где $A' \subset \{a_1, \ldots, a_k\}$, то тогда $M' = \{m_i : a_i \in A'\}$ является трехмерным сочетанием (обратное, очевидно, тоже верно).

Оставшиеся два элемента из A обозначим b_1 и b_2. Положим

$$s(b_1) = 2\sum_{i=1}^{k} s(a_i) - B,\ s(b_2) = \sum_{i=1}^{k} s(a_i) + B.$$

Пусть A содержит подмножество $A' \subset A$, такое, что

$$\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a).$$

Тогда каждая из этих сумм равна $2\sum_{i=1}^{k} s(a_i)$. Причем одно из подмножеств A' или $A \setminus A'$ содержит b_1 и не содержит b_2. Тогда равенство возможно лишь тогда, когда сумма весов элементов $a_i \in A'$ равна B, т.е. когда соответствующее подмножество M' образует трехмерное сочетание.

Обратное. Если задано трехмерное сочетание M', то множество $A' = \{b_1\} \cup \{a_i : m_i \in M'\}$ и есть искомое.

Упражнения.

(1) Покажите, что сведение „трехмерное сочетание“ $\vdash$ „разбиение“ действительно полиномиально.

Литература

(1) Гэри М., Джонсон Д., Вычислительные машины и труднорешаемые задачи. М. Мир, 1982.
(2) Катленд Н., Вычислимость. Введение в теорию рекурсивных функций. М. Мир, 1983.
(3) Китаев А., Шень А., Вялый М., Классические и квантовые вычисления. М. МЦМНО, 1999.