2.1. Вычислительные задачи

Пусть, как обычно, $\{0,1\}^*$ — множество всех слов конечной длины в двоичном алфавите. Для слова $x \in \{0,1\}^*$ через $|x|$ обозначаем его длину, $\{0,1\}^l$ — множество слов длины l, $\perp$ — пустое слово (длины 0), α^l — слово, составленное из l экземпляров символа $\alpha \in \{0,1\}$.

Под *вычислительной задачей* понимают задачу вычисления значений функции $f\colon \{0,1\}^* \to \{0,1\}^*$. Аргумент x называют *входными данными* или просто *входом* задачи, значение $f(x)$ — *выходными данными*, *выходом* или *ответом*.

Будем считать, что входы и выходы отличаются от пустого слова $\perp$. Это слово зарезервируем для описания различных ошибочных ситуаций при вычислениях.

Входы и выходы задачи могут представлять различные объекты: числа, многочлены, векторы, графы, функции и др. Входными данными интересующих нас задач, как правило, являются наборы натуральных чисел. Эти числа будем представлять их двоичной записью: слово $a_1 a_2 \ldots a_l$ кодирует число $a = \sum_{i=1}^{l} a_i 2^{l-i}$. Длину l кодового представления обычно будем выбирать минимально возможной: $l = \lfloor \log_2 a \rfloor + 1$.

Если x представляет только a, то длина $l = |x|$ известна и a легко восстанавливается по x. Если же a является только частью x, то для восстановления a можно включить в x кодовое представление l. Например, можно закодировать число l серией из $l-1$ нулей, и тогда получится гамма-код Элиаса. Это двоичное слово из не более чем $2\lfloor \log_2 a \rfloor + 1$ символов, которое однозначно представляет a. Существуют и другие способы кодирования натуральных чисел. Будем использовать те, которые дают кодовые слова длины $O(\log a)$.

Код целого числа b — это код его знака, дополненный в случае $b \neq 0$ кодом натурального $|b|$. Код рационального числа a/b — это объединение кодов a и b. Понятно, что можно составить разумные правила кодирования произвольных структурированных наборов данных. Интересно, что одни из таких правил — так называемая абстрактно-синтаксическая нотация версии 1 (АСН.1) — часто используются в криптографии для описания ключей, долговременных параметров, сообщений протоколов и пр.

Область определения функции f может включать не все двоичные слова. Другими словами, f может быть *частичной* (а не полной) функцией на $\{0,1\}^*$. Исключение определенных входов связано со спецификой задачи и особенностями кодирования входных данных. Будем по возможности расширять область определения f до $\{0,1\}^*$, устанавливая, например, что все недопустимые входы представляют один и тот же фиксированный допустимый вход. Тем не менее, в необходимых случаях будем подчеркивать, что f является частичной функцией.

2.3. Машина Тьюринга

Для решения задач служат алгоритмы. Неформально говоря, алгоритм — это однозначно определенная последовательность инструкций по преобразованию входных данных в выходные.

Приведенное определение не является математически строгим (что такое инструкция? что значит однозначно определенная?). Для полной формализации используются модели вычислительных устройств, реализующих алгоритмы. Наиболее известная модель — *машина Тьюринга* (МТ), предложенная английским математиком А. Тьюрингом

в 1936 г. Согласно общепризнанному тезису Чёрча — Тьюринга любой алгоритм в интуитивном смысле этого слова может быть реализован некоторой машиной Тьюринга. Другими словами, принято отождествлять алгоритмы и МТ.

Машина Тьюринга M характеризуется следующими элементами:

1. *Лента.* Лента представляет собой набор ячеек, пронумерованных $1, 2, \ldots$. Число ячеек не ограничено. В каждой ячейке хранится символ алфавита $\Sigma = \{0, 1, \textvisiblespace\}$. Символы ячеек ленты образуют бесконечное вправо слово $a = a_1 a_2 \ldots$.
2. *Управляющее устройство.* Управляющее устройство выполняет манипуляции над символами ленты в зависимости от своего состояния $q \in Q$, где Q — конечное множество. Состояния меняются по правилам, заданным *функцией переходов* $\varphi\colon Q \times \Sigma \to Q \times \Sigma \times \{-1, 0, 1\}$. Определено начальное состояние q_0.
3. *Головка управляющего устройства.* Головка передвигается по ленте и в конкретный момент времени находится над ячейкой с номером $d \in \mathbb{N}$. Управляющее устройство может читать символ ячейки под головкой и менять только его. Все организовано так, что вычисления меняют физическую среду (ленту) локально. Это основной тезис интуитувных представлений о вычислениях.

Состояние всей машины задается тройкой (a, q, d). Управляющее устройство меняет состояния, выполняя следующие такты вычислений:

Состояние всей машины задается тройкой (a, q, d). Управляющее устройство меняет состояния, выполняя следующие такты вычислений:

1) прочитывается символ a_d, находящийся под головкой;
2) вычисляется тройка $(q', a'_d, \Delta d) \leftarrow \varphi(q, a_d)$;
3) символ a'_d записывается в ячейку с номером d (вместо a_d);
4) состояние управляющего устройства меняется на q';
5) головка сдвигается на Δd позиций;
6) если $d + \Delta d = 0$, то машина останавливается.

В начале работы M в первые ячейки ленты записываются входные данные $x \in \{0,1\}^*$, а остальные ячейки заполняются символом $\llcorner\!\lrcorner$, т. е. первоначально $a = x\llcorner\!\lrcorner\llcorner\!\lrcorner\ldots$. Управляющее устройство начинает работу в состоянии q_0, а головка устанавливается над первой ячейкой. После остановки машины слово, записанное на ленте, имеет вид $y\llcorner\!\lrcorner\ldots$, где $y \in \{0,1\}^*$. Слово-префикс y объявляется выходными данными M на входе x: $y = M(x)$.

Для работы машины M может потребоваться бесконечно много ячеек на ленте. С другой стороны, для исчерпывающего описания M достаточно указать конечные объекты: множество Q, функцию переходов φ и начальное состояние s_0. Таким образом, описание M можно закодировать двоичным словом $[M]$ конечной длины.

К сожалению, МТ является примитивным устройством и код его описания весьма неудобен. Далее мы будем использовать высокоуровневый способ кодирования, включающий стандартные алгоритмические конструкции типа `if – then – else`, `while`, `for`. При некоторых ограничениях программы на таких языках могут быть преобразованы к описанию $[M]$ с помощью специальной МТ (компилятора).

Будем задавать алгоритмические конструкции на русском языке, не придерживаясь жестких синтаксических ограничений. В наших описаниях алгоритмов обязательно будут рубрики «Вход», «Выход», «Шаги». Например, алгоритм Евклида определяется следующим образом.

Алгоритм Евклида

Вход: натуральные a и b, $a \geqslant b$.

Выход: $\gcd(a, b)$.

Шаги:

1. Пока $b \neq 0$ выполнить:
   (1) $a \leftarrow a \mod b$;

2.4. Разрешимые и неразрешимые задачи

Пусть M останавливается на входе x с результатом y. Результат определен однозначно и, таким образом, M определяет функцию $f_M\colon x \mapsto y$. Функция f_M является частичной, ее значения не определены для тех входов x, на которых M не останавливается.

Определение 2.1. *Частичная функция $f\colon \{0,1\}^* \to \{0,1\}^*$ вычислима (задача f разрешима), если найдется машина Тьюринга M такая, что $f = f_M$ (совпадают и области определения, и значения функций). Говорят, что M вычисляет функцию f (решает задачу f).*

Для задач поиска последняя часть определения уточняется: машина M решает задачу поиска R, если f_M решает R.

Теорема 2.1. *Существуют неразрешимые задачи (невычислимые функции).*

Доказательство. Функций $f\colon \{0,1\}^* \to \{0,1\}^*$ больше, чем описаний машин Тьюринга $[M] \in \{0,1\}^*$. $\square$

Возможно самый известный пример неразрешимой задачи — это *задача об остановке*, предложенная самим Тьюрингом. Входными данными задачи является описание $[M]$ машины M и слово $x \in \{0,1\}^*$. Выход: 1, если M остановится на входе x, и 0 в противном случае.

Теорема 2.2 (Тьюринг). *Задача об остановке неразрешима.*

Доказательство. Пусть h — предикат задачи об остановке: $h([M], x) = 1$ тогда и только тогда, когда M остановится на входе x. Предположим, от противного, что предикат h вычислим на некоторой машине A.

Построим машину B, которая обрабатывает вход $[M]$ следующим образом: если $A([M],[M]) = 1$, то B переходит в бесконечный цикл; в противном случае B возвращает 0. Описание B незначительно расширяет описание A: добавляется проверка выхода и, при необходимости, выполняется переход в бесконечный цикл.

Подадим на вход B ее описание $[B]$. Если B остановится, то $A([B],[B]) = 0$ и остановки быть не должно по определению h. Если B не остановится, то $A([B],[B]) = 1$ и, наоборот, остановка должна быть. Найденные противоречия доказывают требуемый результат. $\square$

Еще одна знаменитая неразрешимая задача — это десятая проблема Гильберта. Входными данными здесь является многочлен от нескольких переменных с целыми коэффициентами (например, $x^n + y^n - z^n$, $n \in \mathbb{N}$). Требуется определить, имеет многочлен целочисленные корни или нет. Неразрешимость десятой проблемы Гильберта доказал в 1971 г. советский математик Ю. Матиясевич.

2.5. Ресурсы

Для решения задач требуются ресурсы. Будем рассматривать два типа ресурсов: *время* и *память*. Нас будет интересовать их пиковое потребление. Пусть x — допустимый вход машины M в том смысле, что M останавливается на этом входе. Через $t_M(x)$ обозначим число тактов, которое выполнит M при вычислениях на входе x, а через $s_M(x)$ — крайнюю правую позицию головки при вычислениях.

Определение 2.2. *Машина Тьюринга M работает за время $T_M(l)$, $l \in \mathbb{N}$, если $T_M(l) = \max_x t_M(x)$, где максимум берется по допустимым $x \in \{0,1\}^l$. Аналогично, M работает на памяти $S_M(l)$, если $S_M(l) = \max_x s_M(x)$.*

Ясно, что $S_M(l) \leqslant T_M(l)$ для любой M. Поэтому время работы является более универсальной характеристикой сложности, чем память.

Будем рассматривать характеристики сложности в асимптотике $l \to \infty$. Напомним обозначения, которые касаются сравнения скорости роста функций:
- $u(l) = o(v(l))$, если $u(l)/f(l) \to 0$;
- $u(l) = O(v(l))$, если $|u(l)| \leqslant Cv(l)$ для некоторой константы C и всех достаточно больших l;
- $u(l) = \Omega(v(l))$, если $v(l) = O(u(l))$;
- $u(l) = \Theta(v(l))$, если $u(l) = O(v(l))$ и $v(l) = O(u(l))$.

Говорят, что M работает за *полиномиальное* время или является *полиномиальной* машиной Тьюринга (ПМТ), если найдется $c \in \mathbb{N}$ такое, что $T_M(l) \leqslant l^c$ для всех достаточно больших l. Последнее условие можно записать по-другому: $T_M(l) = l^{O(1)}$. Среди ПМТ выделяют машины, которые работают за *линейное* время: $T_M(l) = \Theta(l)$, за *квазилинейное*: $T_M(l) = l(\log l)^{O(1)}$, за *квадратичное*: $T_M(l) = \Theta(l^2)$, за *кубическое*: $T_M(l) = \Theta(l^3)$ и т. д. Согласно распространенному тезису Кобхэма, полиномиальные алгоритмы и только они являются *эффективными*, т. е. имеющими практическое значение.

Если время работы M нельзя ограничить многочленом, т. е. $T_M(l) = \Omega(l^c)$ для всех $c \in \mathbb{N}$, то говорят, что M работает за *суперполиномиальное* время. Среди суперполиномиальных машин выделяют *квазиполиномиальные*: $T_M(l) = 2^{(\log l)^{O(1)}}$, *субэкспоненциальные*: $T_M(l) = 2^{o(l)}$ и *экспоненциальные*: $T_M(l) = 2^{l^{O(1)}}$. Есть и *суперэкспоненциальные* машины, но мы их рассматривать не будем.

Таблица 2.1. Базовые арифметические задачи

Задача ($a \geqslant b$)	Время
Сложение $(a, b) \mapsto a + b$	$O(\log a + \log b)$
Вычитание $(a, b) \mapsto a - b$	$O(\log a + \log b)$
Умножение $(a, b) \mapsto ab$	$O(\log a \log b)$
Деление $(a, b) \mapsto (q, r)$: $a = qb + r$, $0 \leqslant r < b$	$O(\log q \log b)$
Наибольший общий делитель $(a, b) \mapsto \gcd(a, b)$	$O(\log^2 a)$
Обращение по модулю $(a, b) \mapsto b^{-1} \bmod a$ (a и b взаимно просты)	$O(\log^2 a)$
Возведение в степень по модулю $(a, b, e) \mapsto b^e \bmod a$	$O(\log e \log^2 a)$
Китайская система сравнений $(a_1, \ldots, a_k, b_1, \ldots, b_k) \mapsto b$: $b \equiv b_i \pmod{a_i}$ (a_i попарно взаимно просты)	$O\left(\log^2 a\right)$ $(a = a_1 \ldots a_k)$
Совершенная степень $a \mapsto (b, e)$: $a = b^e$, где e максимально	$O\left((\log a)^{1+o(1)}\right)$

В таблице 2.1 приводятся оценки времени работы алгоритмов, решающих базовые арифметические задачи с натуральными числами. Оценки могут уточняться. Например, умножение может быть выполнено за время $O((\log a)^{\log_2 3})$ с помощью алгоритма Карацубы — Офмана или даже за время $O(\log a \cdot \log\log a \cdot \log\log\log a)$ с помощью алгоритма

Шёнхаге — Штрассена. Тем не менее, даже приведенные оценки означают эффективность базовых арифметических операций.

2.6. Вероятностные машины

Пусть p — нечетное простое, $Q_p = \{b^2 \colon b \in \mathbb{F}_p^*\}$ — множество квадратичных вычетов по модулю p, $\bar{Q}_p = \mathbb{F}_p^* \setminus Q_p$ — множество квадратичных невычетов.

Рассмотрим задачу поиска квадратичного невычета по заданному модулю p. Для решения этой задачи можно использовать следующий алгоритм.

Алгоритм Поиск невычета (детерминированный)

Вход: p — нечетное простое.

Выход: $a \in \bar{Q}_p$.

Шаги:

1. Для $a = 1, 2, \ldots, p-1$:
   (1) если $a^{(p-1)/2} \equiv -1 \pmod p$, то возвратить a.

Алгоритм обязательно остановится с верным результатом в силу следующих фактов:

1) $a \in \bar{Q}_p$ тогда и только тогда, когда $a^{(p-1)/2} \equiv -1 \pmod p$ (критерий Эйлера);
2) $|Q_p| = |\bar{Q}_p| = (p-1)/2$.

При попытке обосновать эффективность алгоритма возникают трудности. С одной стороны, проверка на шаге 1.1 выполняется эффективно (за кубическое от $\log p$ время). С другой стороны, нельзя гарантировать, что потребуется полиномиальное число таких проверок. Пусть $n_2(p)$ — минимальный квадратичный невычет по модулю p. Доказано (теорема Бургесса), что $n_2(p) \leqslant p^{1/(4\sqrt{e})+\epsilon}$, $\epsilon > 0$. С такой оценкой время работы алгоритма будет экспоненциальным. Более сильная оценка $n_2(p) \leqslant 2\ln^2 p$, справедливая при выполнении расширенной гипотезы Римана, дает полиномиальное время. К сожалению, гипотеза на сегодня не доказана. Можно организовать поиск a не в начале натурального ряда, можно учитывать при поиске вид p (например, $p-1$ будет невычетом, если $p \equiv 3 \pmod 4$), но в целом трудности обоснования эффективности алгоритма сохранятся.

Выходом является переход к вероятностному поиску.

Алгоритм Поиск невычета (вероятностный)

Вход: p — нечетное простое.

Выход: $a \in \bar{Q}_p$.

Шаги:

1. Для $i = 1, 2, \ldots$:
   (1) $a \xleftarrow{R} \{1, 2, \ldots, p-1\}$;
   (2) если $a^{(p-1)/2} \equiv -1 \pmod p$, то возвратить a.

В новом алгоритме запись $u \xleftarrow{R} U$ означает случайный равновероятный выбор u из множества U. Выбранное на шаге 1.1 число a окажется невычетом с вероятностью $1/2$ и в среднем потребуется всего 2 попытки генерации.

Алгоритмы, в которых используется случайный выбор, называются *вероятностными*. Моделью таких алгоритмов является *вероятностная машина Тьюринга* (ВМТ). ВМТ снабжается дополнительной лентой, в ячейках которой записаны символы слова $\omega = \omega_1\omega_2\ldots$. Эти символы являются реализациями независимых в совокупности случайных величин с равномерным распределением на $\{0,1\}$. Ячейки случайной ленты

читаются слева направо, без повторений. В функцию переходов φ, которая определяет функционирование управляющего устройства машины, добавляется еще один аргумент — очередной символ ω.

Число тактов и сам результат работы M на фиксированном входе x являются случайными величинами. В частности, машина может допускать ошибки в определении правильного ответа, т. е. решать задачу не наверняка, а только с какой-то вероятностью. Мы будем рассматривать три типа ошибок.

1. *Нульсторонние ошибки*. Машина M возвращает либо правильный ответ, либо пустое слово $\perp$, которое означает «ответ не найден».
2. *Односторонние ошибки*. Машина M решает задачу распознавания языка L, не ошибается на входах $x \in L$ и может ошибаться на входах $x \notin L$. Возможно альтернативное определение: M не ошибается на входах $x \notin L$ и может ошибаться на входах $x \in L$.
3. *Двусторонние ошибки*. Машина M решает задачу распознавания языка L и может ошибаться как на входах $x \in L$, так и на входах $x \notin L$.

Введем характеристики трудоемкости работы вероятностных машин.

Определение 2.3. *ВМТ M работает за среднее время $ET_M(l)$, $l \in \mathbb{N}$, если $ET_M(l) = \max_x \mathsf{E}\, t_M(x)$. Здесь максимум берется по допустимым $x \in \{0,1\}^l$, а $\mathsf{E}\, t_M(x)$ — среднее число тактов, которое выполнит M при вычислениях на входе x. Усреднение выполняется по всевозможным заполнениям случайной ленты.*

Определение 2.4. *ВМТ M решает задачу f с вероятностью $P_M(l)$, $l \in \mathbb{N}$, если $P_M(l) = \min_x \mathsf{P}\{M(x) = f(x)\}$, где минимум берется по допустимым $x \in \{0,1\}^l$.*

Если для $T_M(l)$, $ET_M(l)$ важна полиномиальная ограниченность, то для $P_M(l)$ важно быть достаточно большой. Будем говорить, что M работает с *преобладающей* вероятностью успеха, если $P_M(l) \geqslant 1 - \nu(l)$, где ν — пренебрежимо малая функция.

Определение 2.5. *Функция $\nu\colon \mathbb{N} \to [0,1]$ пренебрежимо мала, если для любого $c \in \mathbb{N}$ неравенство $\nu(l) < l^{-c}$ выполняется при всех достаточно больших $l \in \mathbb{N}$.*

Будем также говорить, что M работает с *обратно полиномиальной* вероятностью успеха, если найдется $c \in \mathbb{N}$ такое, что $1/P_M(l) \leqslant l^c$ при всех достаточно больших l.

2.7. Алгоритмы Лас-Вегас и Монте-Карло

Вероятностный алгоритм, который мы рассмотрели в предыдущем параграфе, всегда возвращает правильный ответ и работает за полиномиальное в среднем время. Такие алгоритмы принято называть *алгоритмами Лас-Вегас*.

Даже если среднее время работы вероятностной машины M полиномиально, вычисления на определенных входах при определенных ω могут выполняться суперполиномиально долго. В некоторых случаях время работы необходимо ограничивать. *Полиномиальная вероятностная машина Тьюринга* (ПВМТ) — это ВМТ, которая всегда работает за полиномиальное время, независимо от заполнения случайной ленты. Платой за ограничение по времени является возможность появления ошибок в ответах.

Полиномиальным ВМТ соответствуют *алгоритмы Монте-Карло*. Вот пример такого алгоритма.

Алгоритм Поиск невычета (полиномиальный вероятностный)

Вход: p — нечетное простое.

Выход: $a \in \bar{Q}_p$ или $\perp$.

Шаги:

1. Для $i = 1, \ldots, k$:
   (1) $a \xleftarrow{R} \{1, 2, \ldots, p-1\}$;
   (2) если $a^{(p-1)/2} \equiv -1 \pmod{p}$, то возвратить a.

2. Возвратить $\perp$.

Здесь $k = (\log p)^{O(1)}$ — параметр, который определяет время работы $(O(k \log^3 p))$ и вероятность получения правильного ответа $(1 - 2^{-k})$. Ошибки алгоритма могут быть только нульсторонними. Рассмотрим такие алгоритмы подробнее.

Пусть A — ВМТ, которая допускает только нульсторонние ошибки. Тогда A может быть преобразована в машину B, которая последовательно обращается к A до тех пор, пока не будет получен ответ, отличный от $\perp$. Машина B не ошибается. Если $\beta > 0$ — вероятность успеха A на некотором входе, то B на том же входе потребуется выполнить $1/\beta$ обращений к A в среднем. Действительно, среднее число обращений

$$\sum_{t=1}^{\infty} t\mathsf{P}\left\{\text{потребуется } t \text{ обращений}\right\} = \sum_{t=0}^{\infty} \mathsf{P}\left\{\text{потребуется } > t \text{ обращений}\right\} =$$

$$= \sum_{t=0}^{\infty} (1-\beta)^t = \frac{1}{\beta}.$$

Отсюда получаем оценку для среднего времени работы B:

$$ET_B(l) \leqslant \frac{T_A(l)}{P_A(l)}.$$

Если A работает за полиномиальное время с обратно полиномиальной вероятностью успеха, то B работает за полиномиальное в среднем время. Таким образом, переход от A к B есть преобразование алгоритма Монте-Карло в алгоритм Лас-Вегас. Возможно обратное преобразование, которое состоит в принудительной остановке B после определенного числа тактов работы с возвратом $\perp$ после остановки. Фактически этот прием применен в нашем последнем алгоритме поиска невычета.

Машина B может запустить A только k раз, снова ожидая ответ, отличный от $\perp$. Искомый ответ будет получен с вероятностью $\beta_B = 1 - (1-\beta)^k$ за β_B/β обращений к A в среднем. Машина B, как и A, допускает нульсторонние ошибки, но их вероятность быстро уменьшается с ростом k.

Предположим теперь, что A решает задачу распознавания языка L и допускает односторонние ошибки:

$$\mathsf{P}\left\{A(x) = 1\right\} = \begin{cases} 1, & x \in L, \\ \epsilon, & x \notin L. \end{cases}$$

Здесь $0 < \epsilon < 1$. Построим машину B, который получает на вход x, запускает A на этом входе k раз и фиксирует ответы $y_1, \ldots, y_k$. Если все $y_i = 1$, то B возвращает 1. Если хотя бы один ответ $y_i = 0$, то B возвращает 0. Машина B также распознает язык L, также допускает односторонние ошибки, но уже с меньшими вероятностями:

$$\mathsf{P}\left\{B(x) = 1\right\} = \begin{cases} 1, & x \in L, \\ \epsilon^k, & x \notin L. \end{cases}$$

С ростом k время работы B увеличивается линейно, а вероятность ошибки уменьшается экспоненциально. Изменяя k, можно контролировать качество распознавания, сохраняя приемлемым время работы. Такой подход применяется в вероятностных алгоритмах проверки простоты.

Снизить можно и вероятности двусторонних ошибок. Пусть A допускает двусторонние ошибки и вероятность ошибки на входе x не превосходит $\epsilon < 1/2$. Перестроим алгоритм B так, чтобы он определял свой ответ по $y_1, \ldots, y_k$, руководствуясь *правилом*

большинства: если среди y_i единиц больше, чем нулей, то B возвращает 1, в противном случае B возвращает 0. Тогда вероятность ошибки B на входе x не превышает

$$\sum_{i=\lceil k/2 \rceil}^{k} \binom{k}{i} \epsilon^i (1-\epsilon)^{k-i} \leqslant \exp\left(-k\frac{(1/2-\epsilon)^2}{(1/2+\epsilon)}\right).$$

Последняя оценка следует из неравенства Чернова (см. задание 2.5).

2.9. Классы сложности

В этом параграфе мы обсудим ряд важных аспектов сложности вычислений. Эти аспекты связаны только с задачами распознавания. Нас не должно это смущать, поскольку задачи поиска, которые встречаются в криптографии, как правило, легко сводятся к задачам распознавания. Например, `Factor` сводится к распознаванию языка

$$L_{\texttt{Factor}} = \{(n,m)\colon n,m \in \mathbb{N}, \text{существует } d \in \{2,3,\ldots,m\}, \text{ которое делит } n\}.$$

Действительно, нетривиальный делитель d числа n можно найти, обращаясь $O(\log n)$ раз к машине, которая распознает $L_{\texttt{Factor}}$, т. е. проверяет, что $2 \leqslant d \leqslant m$. Граница m задается при этих обращениях так, чтобы реализовать дихотомию множества $\{2,3,\ldots,n-1\}$.

Будем рассматривать далее задачу распознавания с предикатом f и языком L.

Определение 2.8. *Язык L (предикат f) принадлежит классу* $\mathbf{P}$, *если существует ПМТ M такая, что:*
1) *если $x \in L$, то $M(x) = 1$;*
2) *если $x \notin L$, то $M(x) = 0$.*

Определение 2.9. *Язык L (предикат f) принадлежит классу* $\mathbf{NP}$, *если существует ПМТ M и многочлен p такие, что:*
1) *если $x \in L$, то $M(x,y) = 1$ для некоторого слова y длины $|y| \leqslant p(|x|)$;*
2) *если $x \notin L$, то $M(x,y) = 0$ для любого y длины $|y| \leqslant p(|x|)$.*

Класс $\mathbf{P}$ — это класс языков, распознаваемых за полиномиальное время. В $\mathbf{NP}$ входят языки, которые также распознаются за полиномиальное время, но с использованием дополнительных данных — слов y. Слово y, которое часто называется *сертификатом*, подтвердает принадлежность $x \in L$, причем подтверждает доказательно, в том смысле, что для $x \notin L$ ни один из сертификатов не будет принят машиной M.

Знаменитая гипотеза $\mathbf{P} \neq \mathbf{NP}$ означает существование предиката f, для которого ответ $f(x)$ не может быть *вычислен* за полиномиальное время, но может быть *проверен* за полиномиальное время с помощью сертификата. Большинство специалистов считает, что гипотеза $\mathbf{P} \neq \mathbf{NP}$ справедлива.

В классе $\mathbf{NP}$ выделяют подкласс $\mathbf{NPC}$ так называемых $\mathbf{NP}$-*полных* языков. Языки из $\mathbf{NPC}$ имеют максимальную сложность: $L \in \mathbf{NPC}$, если для любого $L' \in \mathbf{NP}$ найдется вычислимая на ПМТ функция $g\colon \{0,1\}^* \to \{0,1\}^*$ такая, что $x \in L'$ тогда и только тогда, когда $g(x) \in L$. Если $\mathbf{NPC} \cap \mathbf{P} \neq \varnothing$, то $\mathbf{P} = \mathbf{NP}$ и наоборот.

Рассмотрим несколько примеров.

Еще один класс сложности связан с вероятностными алгоритмами.

Определение 2.10. *Язык L принадлежит классу* **BPP**, *если существует ПВМТ M такая, что*
1) *если* $x \in L$, *то* $\mathsf{P}\{M(x) = 1\} \geqslant 2/3$;
2) *если* $x \notin L$, *то* $\mathsf{P}\{M(x) = 0\} \geqslant 2/3$.

Другими словами, языки из класса **BPP** распознаются на машинах Монте-Карло с двусторонними ошибками, вероятность которых $\leqslant 1/3$. Выбор порога $1/3$ условен. Важно только, чтобы вероятность ошибки не превосходила $1/2 - \delta$ для некоторого фиксированного $\delta > 0$. Применяя M для распознавания x несколько раз и вынося решение по правилу большинства (см. § 2.7), можно достаточно быстро приблизить вероятность успешного распознавания к 1.

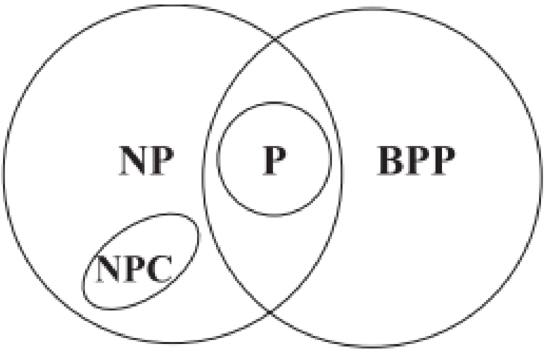

Рис. 2.1. Диаграмма классов сложности (гипотетическая)

На рисунке 2.1 представлено гипотетическое (признаваемое большинством специалистов) соотношение между описанными классами сложности.

2.10. Язык *PRIMES*

В этом параграфе мы рассмотрим сложность распознавания языка *PRIMES*.

Теорема 2.5 (Пратт). $PRIMES \in \mathbf{NP}$.

Доказательство. Пусть для $a \in \mathbb{Z}_n^*$ выполняется: $a^{n-1} \equiv 1 \pmod n$ и

$$a^{(n-1)/q_i} \not\equiv 1 \pmod n, \quad i = 1, 2, \ldots, k,$$

где $q_1, q_2, \ldots, q_k$ — все простые делители $n-1$. Тогда порядок a в группе $\mathbb{Z}_n^*$ равняется $n-1$. По теореме Лагранжа порядок элемента группы делит порядок группы. Следовательно, $n-1$ делит $|\mathbb{Z}_n^*| = \varphi(n)$, что справедливо только тогда, когда n — простое. Таким образом, выполнение указанных условий доказывает простоту числа n.

Рассмотрим машину M, которая берет на вход натуральное n и сертификат

$$\text{cert}(n) = (a, (q_1, \text{cert}(q_1)), \ldots, (q_k, \text{cert}(q_k))).$$

Простоту чисел q_i демонстрируют сертификаты $\text{cert}(q_i)$, которые имеют такую же структуру, как и $\text{cert}(n)$, т. е. включают основание из $\mathbb{Z}_{q_i}^*$, простые делители числа $q_i - 1$ и сертификаты этих делителей. В сертификаты делителей могут быть вложены новые сертификаты и так далее. В целом получается целое дерево сертификатов. Сертификаты $\text{cert}(1)$, $\text{cert}(2)$ отдельно определяются как пустые слова.

Общее число вершин в дереве сертификатов для $n = 2$ и для нечетного $n \geqslant 3$ не превосходит $2\log_2 n - 1$. Действительно, это верно для $n = 2$ и $n = 3$. Если $n > 3$, то $n - 1 = q_1 q_2 \ldots q_k$ — составное ($k \geqslant 2$) и число сертификатов по индукции не больше

$$1 + \sum_{i=1}^{k} (2\log_2 q_k - 1) = 1 + 2\log_2(q_1 q_2 \ldots q_k) - k < 2\log_2 n - 1.$$

Длина каждого сертификата за вычетом длины вложенных сертификатов есть $O(\log n)$. Поэтому $|\mathrm{cert}(n)| = O(\log^2 n)$.

Машина M обрабатывает $(n, \mathrm{cert}(n))$ следующим образом.

1. Если $n = 1$, то возвратить 0.
2. Если $n = 2$, то возвратить 1.
3. Если n — четное, то возвратить 0.
4. Если $a^{n-1} \not\equiv 1 \pmod n$, то возвратить 0.
5. Если $\prod_{i=1}^{k} q_i \neq n - 1$, то возвратить 0.
6. Для $i = 1, \ldots, k$:
   1) если $a^{(n-1)/q_i} \equiv 1 \pmod n$, то возвратить 0;
   2) если $M(q_i, \mathrm{cert}(q_i)) = 0$, то возвратить 0.
7. Возвратить 1.

Машина M работает за полиномиальное время (проверить самостоятельно), всегда дает ответ 1 на простых n и никогда не дает ответ 1 на составных n. Мы находимся в условиях определения класса **NP** и теорема доказана. $\square$

Дополнительный к $PRIMES$ язык $\overline{PRIMES} = \{n \in \mathbb{N} \colon n \text{ — составное}\}$ также лежит в классе **NP** — сертификатом принадлежности n языку является любой нетривиальный делитель. Анализ вычислительных задач показывает, что если и язык L, и дополнительный язык $\overline{L}$ одновременно лежат в **NP**, то, как правило, $L \in \mathbf{P}$. Действительно, в 2002 году индийские математики М. Агравал, Н. Каяла и Н. Саксена (АКС) разработали полиномиальный алгоритм распознавания простоты, т. е. доказали, что $PRIMES \in \mathbf{P}$.

К сожалению, алгоритм АКС является довольно медленным, даже самые эффективные его редакции работают за время $O(\log^6 n)$. В криптографии для проверки простоты в основном применяют вероятностный алгоритм Рабина — Миллера, который работает за время $O(\log^3 n)$, допускает только односторонние ошибки (составное может быть признано простым), вероятность ошибки не превосходит $1/4$. Существование этого алгоритма даже без результатов АКС доказывает, что $PRIMES \in \mathbf{BPP}$.