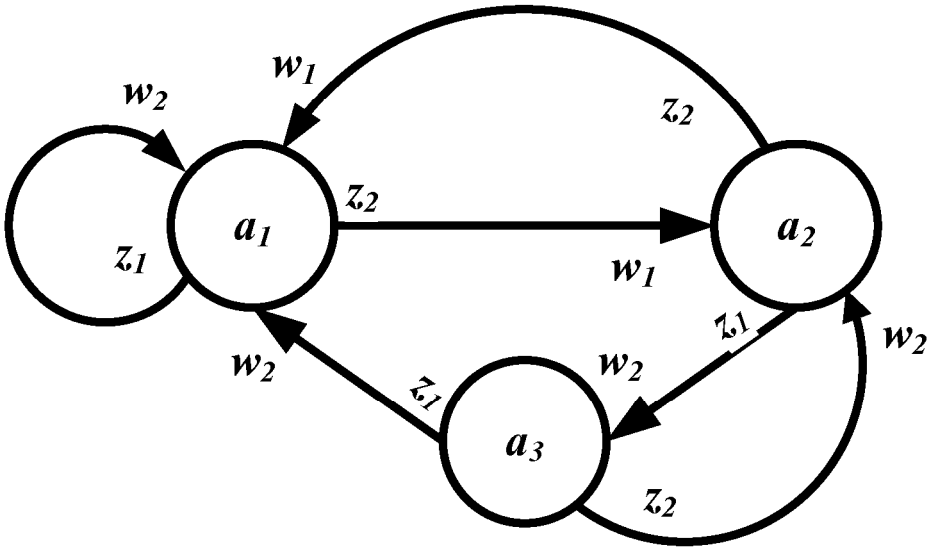
ТЕОРИЯ АВТОМАТОВ

Лекционный материал, разработанный
для самостоятельного освоения
в условиях удаленной системы обучения

часть 9



Теория сложности алгоритмов часть I

О вычислительной сложности

Классическая теория вычислимости, с введением в которую мы познакомились в двух предыдущих главах, подразумевает, что все абстрактные вычислительные устройства удовлетворяют *принципу потенциальной осуществимости*, т. е. предполагается, что при осуществлении процесса вычисления, определенного алгоритмом, мы имеем неограниченный запас времени и материалов. Однако количество простейших действий, необходимых для получения результата вычисления, может быть весьма большим. На первый взгляд, учитывая достижения индустрии компьютерных технологий, данная проблема кажется несущественной. Тем не менее, вычислительная практика показывает, что многие алгоритмические проблемы, которые в принципе разрешимы, невозможно решить на современных вычислительных устройствах, поскольку алгоритмы решения этих проблем требуют в буквальном смысле космических затрат времени и ресурсов. Примерами таких задач являются классические задачи из комбинаторики: задача о коммивояжере, задача о гамильтоновом цикле, задача выполнимости булевой формулы и др.

Пример. Рассмотрим, например, *задачу о коммивояжере*. Постановка этой задачи очень проста: на карте обозначены n городов и известны расстояния между ними, коммивояжеру (который находится в одном из этих городов) необходимо найти маршрут обхода всех городов, имеющий минимальную длину. Теоретически эта задача безусловно разрешима. Если необходимо посетить n городов, то число всевозможных маршрутов равно в точности $(n-1)!$. Поэтому достаточно найти длины всех маршрутов и выбрать из них минимальную. Однако если мы начнем реализовывать этот алгоритм на практике, то сразу придем к выводу о его непригодности. Допустим, коммивояжеру необходимо посетить 10 городов. Тогда потребуется перебрать $9! = 362880$ маршрутов, такая задача вполне по силам современному компьютеру. Если же, например, количество городов равно 40, то потребуется перебрать 39! маршрутов, а это уже больше, чем 10^{45}. Даже если мы сможем просчитывать 10^{15} маршрутов в секунду (что гораздо быстрее, чем быстродействие многих современных суперкомпьютеров), то время, необходимое для завершения вычислений, составит несколько миллиардов сроков жизни Вселенной!

Теория вычислительной сложности отказывается от принципа потенциальной осуществимости. Если в классической теории вычислимости совокупность всех алгоритмических проблем разбивалась на две категории — разрешимые и неразрешимые, то в теории сложности возникает более тонкая классификация алгоритмических проблем: в зависимости от затрачиваемых времени и ресурсов различают более приемлемые и менее приемлемые алгоритмы. Общепринято, что если задачу нельзя решить

быстрее, чем за экспоненциальное время, то ее следует рассматривать как трудно разрешимую. В нашем примере с задачей о коммивояжере функция $(n-1)!$ растет быстрее, чем 2^n, поэтому предложенный алгоритм перебора всех маршрутов оказывается непригодным. Поскольку экспоненциальная функция (такая, как 2^n) растет быстрее любой полиномиальной функции от n, то при таком соглашении задачи, решаемые алгоритмами полиномиальной сложности, будут легко разрешимыми.

Таким образом, постулируется следующий тезис: *алгоритмы, число операций которых оценивается некоторым полиномом $p(n)$ от длины входных данных n, являются реально пригодными.*

При формальном изложении основ теории сложности в качестве модели вычислений традиционно используются машины Тьюринга. При таком формализме время будет измеряться числом тактов машины Тьюринга, затраченных на вычисление, а память будет измеряться числом ячеек ленты, использованных при вычислении. По причинам, которые скоро станут ясными, ключевое понятие в теории сложности — это недетерминированная машина Тьюринга.

Недетерминированные машины Тьюринга

Обычные машины Тьюринга, введенные нами ранее, называют также *детерминированными машинами Тьюринга*, подчеркивая тем самым, что в программе любой такой машины для каждого состояния q_i $(i > 0)$ и любого внешнего символа a_j существует ровно одна команда вида $q_i a_j \to \ldots$ Процесс вычисления на такой машине мы представляли как последовательность конфигураций, каждая из которых однозначно определялась предыдущей.

По аналогии с недетерминированными конечными автоматами мы введем обобщенное понятие *недетерминированной машины Тьюринга*, которая отличается от детерминированного варианта тем, что в ее программе для любого q_i $(i > 0)$ и любого a_j может быть несколько (или нисколько) команд вида $q_i a_j \to \ldots$ Это означает, что, находясь в состоянии q_i и обозревая на ленте символ a_j, машина может выполнить любую из этих команд. Если же в состоянии q_i, обозревая символ a_j, машина не находит в программе команду вида $q_i a_j \to \ldots$, то дальнейшие шаги в данной ветви вычислений невозможны. Таким образом, шаг работы машины уже не определен однозначно, а процесс вычисления можно представлять как дерево конфигураций, в котором из одной конфигурации может получиться уже несколько других. Каждая ветвь дерева конфигураций либо является бесконечной, либо заканчивается в заключительном состоянии, либо обрывается на некотором незаключительном состоянии. В последнем случае мы будем говорить, что произошла *неправильная остановка машины.*

Перейдем к формальным определениям.

Определение. *Недетерминированная машина Тьюринга* — это упорядоченная пятерка $T = \langle \mathcal{A}, Q, P, q_1, q_0 \rangle$, где $\mathcal{A}$, Q, q_1, q_0 имеют тот же смысл, что и у детерминированных машин, а *программой* называется любое подмножество

$$P \subseteq (Q \setminus \{q_0\}) \times \mathcal{A} \times Q \times \mathcal{A} \times \{\Lambda, R, L\}.$$

Как и раньше, элементы программы P называются *командами*, и каждую команду $\langle q_i, a_j, q_k, a_l, S \rangle \in P$ мы записываем в виде слова $q_i a_j \to q_k a_l S$.

Замечание. Таким образом, в программе P недетерминированной машины для любых q_i и a_j может быть несколько команд вида $q_i a_j \to \ldots$ Допускается также случай, когда в P команды, начинающиеся на $q_i a_j \to \ldots$, отсутствуют. Если в программе для любого q_i $(i > 0)$ и любого a_j существует ровно одна команда вида $q_i a_j \to \ldots$, то машина T детерминирована.

Конфигурации (машинные слова) для недетерминированных машин Тьюринга определяются так же, как и раньше. Переопределим отношение преобразования конфигураций за один шаг работы машины Тьюринга в недетерминированном случае.

Определение. Пусть $M = Aq_i a_j B$ — конфигурация недетерминированной машины T. Говорят, что конфигурация M' *получается из* M *за один шаг работы машины* T, и пишут $M \vdash_T M'$, если выполняется одно из условий:

а) В программе P существует команда вида $q_i a_j \to q_k a_l$, и $M' = Aq_k a_l B$.

б) В программе P существует команда вида $q_i a_j \to q_k a_l R$, $B = a_s B'$, и $M' = Aa_l q_k a_s B'$.

в) В программе P существует команда вида $q_i a_j \to q_k a_l R$, $B = \Lambda$, и $M' = Aa_l q_k a_0$ (достраивается новая ячейка справа).

г) В программе P существует команда вида $q_i a_j \to q_k a_l L$, $A = A' a_s$, и $M' = A' q_k a_s a_l B$.

д) В программе P существует команда вида $q_i a_j \to q_k a_l L$, $A = \Lambda$, и $M' = q_k a_0 a_l B$ (достраивается новая ячейка слева).

Замечание. В силу недетерминированности для фиксированной конфигурации M может существовать несколько конфигураций M', таких, что $M \vdash_T M'$. Если $M = Aq_i a_j B$ и в программе нет команд вида $q_i a_j \to \ldots$ (в частности, если $i = 0$), то вообще не существует конфигурации M' такой, что $M \vdash_T M'$.

Определение. *Вычислением* на недетерминированной машине Тьюринга T называется любая конечная последовательность конфигураций $M_0, M_1, \ldots, M_n$ для некоторого $n \in \omega$, такая, что

$$M_0 \vdash_T M_1 \vdash_T \ldots \vdash_T M_n.$$

При этом говорят, что вычисление *начинается в конфигурации* M_0*, заканчивается в конфигурации* M_n *и имеет длину* n (или состоит из n *шагов*), и пишут $M_0 \vdash_T^n M_n$.

Если $M \vdash_T^n M'$ для некоторого $n \in \omega$, то пишут $M \vdash_T^* M'$. Если в вычислении $M_0 \vdash_T \ldots \vdash_T M_n$ последнее слово $M_n = Aq_0 a_j B$, то говорят, что в данном вычислении произошла *правильная остановка*. Если $M_n = Aq_i a_j B$, где $i > 0$, и в программе нет команды вида $q_i a_j \to \ldots$, то говорят, что в данном вычислении произошла *неправильная остановка*.

Замечание. Если T — детерминированная машина Тьюринга, то элементы вычисления однозначно определяются по начальной конфигурации M_0. Поэтому условие $M_0 \vdash_T^n M_n$ равносильно условию $M_n = \big(M_0\big)_T^{(n)}$ и можно говорить, что машина T перерабатывает конфигурацию M_0 в конфигурацию M_n.

Определение. Пусть $T = \langle \mathcal{A}, Q, P, q_1, q_0 \rangle$ — детерминированная машина Тьюринга. Говорят, что язык L над алфавитом $\mathcal{A}_0 \subseteq \mathcal{A} \setminus \{0\}$ *распознается машиной* T, если для любого слова $w \in \mathcal{A}_0^*$ выполняются следующие условия:

а) машина T в процессе переработки входной конфигурации $q_1 0w0$ не достраивает новых ячеек слева;

б) если $w \in L$, то машина T перерабатывает конфигурацию $q_1 0w0$ в конфигурацию $uq_0 1v$ для некоторых слов $u, v \in \mathcal{A}^*$;

в) если $w \notin L$, то машина T перерабатывает конфигурацию $q_1 0w0$ в конфигурацию $uq_0 0v$ для некоторых слов $u, v \in \mathcal{A}^*$.

Таким образом, детерминированная машина Тьюринга, распознающая язык L, всегда правильно останавливается на входных словах $w \in \mathcal{A}_0^*$ и после остановки выдает 1, если $w \in L$, и выдает 0, если $w \notin L$. Заметим, что в определении нет никаких требований на поведение машины при запуске с входными словами, содержащими символы, не принадлежащие $\mathcal{A}_0$.

С помощью недетерминированных машин Тьюринга также можно распознавать языки. Однако соответствующее определение выглядит необычным и несимметричным.

Определение. Пусть $T = \langle \mathcal{A}, Q, P, q_1, q_0 \rangle$ — недетерминированная машина Тьюринга. Говорят, что язык L над алфавитом $\mathcal{A}_0 \subseteq \mathcal{A} \setminus \{0\}$ *распознается машиной* T, если для любого слова $w \in \mathcal{A}_0^*$ выполняются следующие условия:

а) в любом вычислении машины T с входной конфигурацией $q_1 0w0$ не достраиваются новые ячейки слева;

б) существует натуральное число N, зависящее от T и w, такое, что не существует конфигурации M с условием $q_1 0w0 \vdash_T^N M$;

в) $w \in L$ тогда и только тогда, когда $q_1 0w0 \vdash_T^* uq_0 1v$ для некоторых $u, v \in \mathcal{A}^*$.

Таким образом, если недетерминированная машина T распознает язык L, то все ее вычисления, начатые на входном слове $w \in \mathcal{A}_0^*$, заканчиваются (это может произойти по двум причинам: либо в вычислении происходит правильная остановка, либо происходит неправильная остановка). При этом если хотя бы одно вычисление заканчивается в состоянии q_0 и выдает 1, то считается, что слово w распозналось. Даже если в данной ситуации среди остальных ветвей вычислений найдутся такие, которые выдают 0 в состоянии q_0 или заканчиваются ввиду неправильной остановки, машина по определению распознает слово w.

Пример. Рассмотрим язык L=$\{a^n \mid n$ — составное число$\}$ над алфавитом $\mathcal{A}_0$=$\{a\}$.

Обычная детерминированная машина Тьюринга, распознающая данный язык, действует следующим образом: если на вход подано слово a^n, $n > 0$, то машина последовательно перебирает числа $k = 2, 3, \ldots, [\sqrt{n}]$, и для каждого k проверяет, делится ли n на число k. Если хотя бы одно k делит n, то число n — составное, иначе — простое (или меньше двойки).

Мы построим недетерминированную машину Тьюринга, которая распознает тот же язык L, но устроена проще. Благодаря недетерминированности циклический перебор чисел $k = 2, 3, \ldots, [\sqrt{n}]$ можно разбить на несколько ветвей параллельных вычислений — для каждого значения k своя ветвь. Если хотя бы одна ветвь окажется успешной, то число — составное. Если все ветви неуспешные, то число — несоставное.

Внешний алфавит машины, кроме символов $a, 0, 1$, будет содержать вспомогательный символ b. Технически работа машины выглядит так. Допустим, на вход машины

подано слово $0aaaaaaaaaaaa0$ (т. е. $n = 12$ в данном частном случае). Машина пытается недетерминированно «угадать» нетривиальный делитель k числа n, для этого она, используя символ b, отмечает в входном слове начальный префикс длины k следующим образом: $0bb0aaaaaaaaa0$ (т. е. $k = 3$ в данном частном случае). После этого машина пытается «исчерпать» выбранным префиксом $bb0$ все буквы a, передвигая его слева направо примерно так:

$$0bb0aaaaaaaaa0 \vdash^* 0000bb0aaaaaa0 \vdash^* 0000000bb0aaa0 \vdash^* \ldots$$

Если входное слово удается покрыть кратным количеством слов вида $bb0$, то «догадка» машины оказалась правильной. Иначе, т. е. если n не делится нацело на k, данную ветвь вычислений можно считать неуспешной.

Программа имеет 13 состояний:

$$\begin{array}{ll}
q_1 0 \to q_2 0R & q_8 b \to q_8 bL \\
q_2 a \to q_3 bR & q_8 0 \to q_9 0L \\
q_3 a \to q_3 bR & q_9 b \to q_4 b \\
q_3 a \to q_4 0L & q_9 0 \to q_{10} 0R \\
q_4 b \to q_4 bL & q_{10} 0 \to q_{11} 0R \\
q_4 0 \to q_5 0R & q_{11} b \to q_{11} bR \\
q_5 b \to q_6 0R & q_{11} a \to q_{12} 0R \\
q_6 b \to q_6 bR & q_{12} 0 \to q_0 1 \\
q_6 0 \to q_7 0R & q_{12} a \to q_{13} aL \\
q_7 b \to q_7 bR & q_{13} 0 \to q_4 0L \\
q_7 a \to q_8 bL &
\end{array}$$

Вычисления по данной программе происходят следующим образом. Начальная конфигурация имеет вид $q_1 0a^n 0$. Если $n \leqslant 1$, то сразу происходит неправильная остановка. Если $n \geqslant 2$, то в состоянии q_3 машина недетерминированным образом выбирает число k, формально производя такие преобразования:

$$q_1 0a^n 0 \vdash^* 0b^{k-2} q_4 b0a^{n-k}0 = 00^{sk+t} b^{k-2-t} q_4 b0b^t a^{n-(s+1)k-t}0,$$

при значениях $s = 0$ и $t = 0$.

Далее запускается двойная циклическая структура. Внешний цикл имеет счетчик s, а его описанию соответствуют состояния q_4 – q_{13}. Вложенный в него внутренний цикл имеет счетчик t, ему соответствуют состояния q_4 – q_9. Одна циклическая итерация состоит из следующей цепочки преобразований:

$$\begin{gathered}
00^{sk+t} b^{k-2-t} q_4 b0b^t a^{n-(s+1)k-t}0 \vdash^* 00^{sk+t} q_5 b^{k-1-t} 0b^t a^{n-(s+1)k-t}0 \vdash \\
\vdash 00^{sk+t} 0q_6 b^{k-2-t} 0b^t a^{n-(s+1)k-t}0 \vdash^* 00^{sk+t} 0b^{k-2-t} 0q_7 b^t a^{n-(s+1)k-t}0.
\end{gathered}$$

Если в состоянии q_7 обозревается 0, то происходит неправильная остановка, это означает, что либо k не делит n, либо $k = n$. Иначе цепочка продолжается следующим образом:

$$00^{sk+t+1} b^{k-2-t} 0b^t q_7 a^{n-(s+1)k-t}0 \vdash^* 00^{sk+t+1} b^{k-2-t} q_8 0b^{t+1} a^{n-(s+1)k-(t+1)}0.$$

Далее из состояния q_8 машина переходит в состояние q_9 и смещается на ячейку влево. Если в состоянии q_9 обозревается b, то машина переходит на следующую итерацию

во внутреннем цикле. Если же в состоянии q_9 обозревается 0, то, значит, $t = k - 2$ и цепочка продолжается так:

$$00^{(s+1)k-2}q_900b^{k-1}a^{n-(s+2)k+1}0 \vdash^* 00^{(s+1)k}b^{k-1}q_{11}a^{n-(s+2)k+1}0.$$

Если в состоянии q_{11} обозревается 0, то происходит неправильная остановка, это означает, что $n = (s + 2)k - 1$, т. е. не делится на k. Иначе цепочка продолжается так:

$$00^{(s+1)k}b^{k-1}q_{11}a^{n-(s+2)k+1}0 \vdash 00^{(s+1)k}b^{k-1}0q_{12}a^{n-(s+2)k}0.$$

Если в состоянии q_{12} обозревается 0, то, значит, k делит n, машина производит правильную остановку и выдает 1. Иначе машина после исполнения последних двух команд из программы переходит на следующую итерацию во внешнем цикле.