1. Семинар 1 (07.02.2017)

1.1. Алгоритмы

Алгоритм — “набор инструкций, описывающих порядок действий исполнителя для достижения некоторого результата”.

Пример 1 (алгоритм Евклида)**.** Рассмотрим алгоритм, который по заданным натуральным числам $a > 0$, b находит их наибольший общий делитель $\gcd(a, b)$.

```
a_0 := a; b_0 := b; n := 0;
while b_n ≠ 0 do
    b_{n+1} := a_n mod b_n;
    a_{n+1} := b_n;
    n := n + 1;
end
return a_n
```

Почему этот алгоритм действительно возвращает наибольший общий делитель? Заметим, что $\gcd(x, y) = \gcd(x \bmod y, y)$ для всех $x, y \in \mathbb{Z}$. Поэтому для всех n выполняется $\gcd(a_n, b_n) = \gcd(a, b)$, и $a_n \bmod b_n = 0 \Leftrightarrow b_n = \gcd(a_n, b_n) = \gcd(a)$. Так как $0 \leq a_{n+2} = b_{n+1} = a_n \bmod b_n < a_n$, то алгоритм совершит лишь конечное число шагов, а затем остановится, т.е. не может быть зацикливания. Отсюда следует, что алгоритм корректен.

Как мы видели, понятие алгоритма довольно расплывчато. Будем считать, что алгоритм принимает на вход некоторый *конструктивный объект*, выполняет определенную последовательность действий и возвращает (если останавливается) новый конструктивный объект. Конструктивный объект — тот, для которого существует конструктивный способ задать его, например, натуральное число, матрица, граф и т.д. Вообще, можно считать, что у нас есть некоторый конечный *алфавит* Σ, и рассматриваемые объекты можно записать как *слова* конечной длины в этом алфавите. Σ^k — все слова длины k в алфавите Σ, $\Sigma^* := \bigcup_{k=0}^{\infty} \Sigma^k$ — вообще все слова в Σ. Тогда **язык** L — это некоторое подмножество Σ^*.

Для теории вычислений удобна такая постановка задачи: пусть дан некоторый язык $L \subset \Sigma^*$. Алгоритм должен определять принадлежность слова этому языку (membership problem), а именно: алгоритм **принимает** язык L, если

- на каждом слове $w \in L$ алгоритм останавливается и выдает $Accept$;
- на каждом слове $w \notin L$ алгоритм либо не останавливается, либо останавливается и выдает $Reject$.

Если к тому же алгоритм останавливается на любом входе, то будем говорить, что он **разрешает** язык L.

1.2. Машина Тьюринга

Для формализации понятия алгоритма требуется некоторая *модель вычислений*. Для теоретического анализа в качестве такой модели можно выбрать **машину Тьюринга**. Машина Тьюринга состоит из следующих компонент: бесконечной в обе стороны ленты, поделенной на ячейки, и управляющего устройства (головки). Конкретная машина M задается с помощью кортежа $(\Sigma, \Pi, Q, q_0, F, \delta)$, где

- Σ — входной алфавит, $\Pi \supset \Sigma$ — алфавит МТ (конечный);
- $\sqcup \in \Pi \setminus \Sigma$ — пробельный символ;
- Q — *конечное* множество состояний;
- $q_0 \in Q$ — стартовое состояние, $F \subset Q$ — терминальные состояния;
- $\delta\colon Q \times \Pi \to Q \times \Pi \times \{-1, 0, 1\}$ — функция перехода.

МТ работает следующим образом: в начальный момент на ленте записано входное слово $x \in \Sigma^*$, машина находится в состоянии q_0 и головка расположена над первым символом входа. На каждом шаге (такте) машина, находящаяся в состоянии q над символом a, осуществляет переход по правилу δ: если $\delta(q, a) = (p, b, i)$, то МТ записывает в текущую ячейку символ b вместо a, переходит в состояние p и затем сдвигается в соответствии с i: влево, если $i = -1$, вправо, если $i = +1$, остается на месте, если $i = 0$. Если машина оказалась в одном из терминальных состояний $q \in F$, то она прекращает работу. Выходом алгоритма считаем слово y, записанное на ленте при окончании работы. *Конфигурацию МТ* в ходе работы (т.е. текущее состояние МТ, положение головки и слово на ленте) удобно описывать в виде строки $u\,q\,v$, где u — слово слева от головки (до пустой части ленты), v — слово справа, q — состояние МТ, головка расположена над первым символом слова v.

Существуют различные вариации МТ, которые эквивалентны данному определению: с маркерами конца входа, с полубесконечной лентой, с несколькими лентами и т.д. Кроме того, заметим, что входной алфавит можно всегда считать бинарным: $\Sigma = \{0, 1\}$, т.к. любой конечный алфавит можно в нем закодировать.

Для задачи распознавания языка будем считать, что у машины Тьюринга всего два терминальных состояния: $F = \{Accept, Reject\}$. Язык, принимаемый МТ M, обозначается как $L(M)$.

Пример 2. Построим МТ, которая удваивает входное слово, т.е. при входе x возвращает $y = xx$. Входной алфавит будем считать бинарным (хотя это не принципиально). Наша МТ будет копировать по одному символу входного слова.

- $\Sigma = \{0, 1\}$, $\Pi = \{0, 1, \tilde{0}, \tilde{1}, \hat{0}, \hat{1}, \sqcup\}$;
- $Q = \{q_0, copy\_0, copy\_1, go\_back, rewrite, stop\}$, $F = \{stop\}$;
- функция перехода:

$$\begin{aligned}
&\delta(q_0, a) = (copy_a, \hat{a}, +1),\ \ a \in \Sigma;\\
&\delta(copy_a, b) = (copy_a, b, +1);\\
&\delta(copy_a, \tilde{b}) = (copy_a, \tilde{b}, +1);\\
&\delta(copy_a, \sqcup) = (go_back, \tilde{a}, -1);\\
&\delta(go_back, \tilde{a}) = (go_back, \tilde{a}, -1);\\
&\delta(go_back, a) = (go_back, a, -1);\\
&\delta(go_back, \hat{a}) = (q_0, a, +1);\\
&\delta(q_0, \tilde{a}) = (rewrite, a, +1);\\
&\delta(rewrite, \tilde{a}) = (rewrite, a, +1);\\
&\delta(rewrite, \sqcup) = (stop, \sqcup, 0).
\end{aligned}$$

Заметим, что мы определили функцию перехода не для всех возможных аргументов, т.к. некоторые из них не могут реализоваться в ходе работы. На них $\delta(\cdot,\cdot)$ можно доопределить произвольным образом.

Доказательство корректности построенной МТ обычно проводят по индукции. Покажем, что наша машина действительно копирует по одному символу. Пусть она находится в конфигурации $u\,[q_0]\,av\tilde{u}$, где $a \in \{0,1\}$, а $\tilde{u}$ — слово u, в котором все 0 заменены на $\tilde{0}$, все 1 — на $\tilde{1}$ (для удобства состояние выделено скобками). Символом $\vdash$ будем обозначать переход от одной конфигурации к другой за один такт, $\vdash^*$ — за конечное число шагов. Как несложно видеть по нашей функции перехода, МТ действует следующим образом:

$$u\,[q_0]\,av\tilde{u} \vdash u\hat{a}\,[copy\_a]\,v\tilde{u} \vdash^* u\hat{a}v\tilde{u}\,[copy\_a] \vdash \\ \vdash u\hat{a}v\tilde{u}\,[go\_back]\,\tilde{a} \vdash^* u\,[go\_back]\,\hat{a}v\tilde{u}\tilde{a} \vdash ua\,[q_0]\,v\tilde{u}\tilde{a}.$$

Таким образом, мы приходим к конфигурации того же вида, но со скопированным символом a. Нужно еще показать, что после копирования машина уберет все метки $\tilde{}$, и оставит удвоенное исходное слово: пусть $x = aw$; тогда

$$[q_0]\,x = [q_0]\,aw \vdash^* aw\,[q_0]\,\tilde{a}\tilde{w} \vdash awa\,[rewrite]\,\tilde{w} \vdash^* \\ \vdash^* awaw\,[rewrite] = xx\,[rewrite] \vdash xx\,[stop],$$

что и требовалось.

Пример 3. Построим МТ, которая распознает слова из алфавита $\Sigma = \{a, b, c\}$, в которых равное количество букв a и b (язык $L_=$). Она будет искать каждому символу a или b парный символ, который еще не занят, и ставить на них метки, что они заняты. Если какому-то символу не нашлось парного, то машина отвергает слово, а если она дошла до конца входа, и все символы спарены, то принимает слово.

- $\Pi = \{a, b, c, \hat{a}, \hat{b}, \tilde{a}, \tilde{b}, \sqcup\}$;
- $Q = \{q_0, find\_a, find\_b, go\_back, Accept, Reject\}$, $F = \{Accept, Reject\}$;
- функция перехода:

$$\begin{aligned}
&\delta(q_0, a) = (find_b, \hat{a}, +1);\\
&\delta(q_0, b) = (find_a, \hat{b}, +1);\\
&\delta(q_0, \gamma) = (q_0, \gamma, +1),\ \ \gamma \neq a, b, \sqcup;\\
&\delta(q_0, \sqcup) = (Accept, \sqcup, 0);\\
&\delta(find_\alpha, \alpha) = (go_back, \tilde{\alpha}, -1),\ \ \alpha \in \{a, b\};\\
&\delta(find_\alpha, \beta) = (find_\alpha, \beta, +1),\ \ \beta \neq \alpha, \sqcup;\\
&\delta(find_\alpha, \sqcup) = (Reject, \sqcup, 0);\\
&\delta(go_back, \gamma) = (go_back, \gamma, -1),\ \ \gamma \neq \hat{a}, \hat{b};\\
&\delta(go_back, \hat{\alpha}) = (q_0, \hat{\alpha}, +1).
\end{aligned}$$

Почему эта МТ корректна? Пусть на каком-то шаге она находится в конфигурации вида $u\,[q_0]\,av\alpha bw$, где в слове $v\alpha$ нет символа b, и в слове $av\alpha bw$ справа от головки нет символа с меткой $\hat{}$. Тогда

$$u\,[q_0]\,av\alpha bw \vdash u\hat{a}\,[find\_b]\,v\alpha bw \vdash^* u\hat{a}v\alpha\,[find\_b]\,bw \vdash u\hat{a}v\,[go\_back]\,\alpha\tilde{b}w \vdash^* \\ \vdash^* u\,[go\_back]\,\hat{a}v\alpha\tilde{b}w \vdash u\hat{a}\,[go\_back]\,v\alpha\tilde{b}w.$$

То есть, за один такой цикла МТ находит одну пару символов a, b, ставит на них метки, и сдвигается на один шаг вправо. Если МТ находится в конфигурации вида $u\,[q_0]\,av$, где в слове v нет символа b, то

$$u\,[q_0]\,av \vdash u\hat{a}\,[find\_b]\,v \vdash^* u\hat{a}v\,[find\_b] \vdash u\hat{a}v\,[Reject],$$

и машина отвергает входное слово. В то же время, из конфигурации $u\,[q_0]\,v$, где в слове v нет символов a и b, машина переходит в состояние $Accept$:

$$u\,[q_0]\,v \vdash^* uv\,[q_0] \vdash uv\,[Accept].$$

Оставшиеся случаи разбираются аналогично, и отсюда мы видим, что построенная МТ разрешает язык $L_=$.

Может показаться, что МТ — очень слабая модель вычислений. Но на машине Тьюринга можно, например, реализовать сложение и умножение чисел, записанных в двоичной записи (подумайте, как!), копирование слов, сравнение и т.д. Более того, тезис Тьюринга ($\sim$тезис Черча, Черча-Тьюринга, Поста) утверждает, что любой алгоритм можно реализовать на машине Тьюринга. В теории вычислений это принимается как постулат, либо вообще как определение алгоритма. Этот тезис обосновывается тем фактом, что пока неизвестно ни одного алгоритма, который *нельзя* было бы реализовать с помощью некоторой МТ.

1.3. Сложность алгоритма

Важным вопросом при анализе алгоритма является время его работы и объем используемой памяти. Чтобы иметь возможность строго определить их, необходима какая-нибудь модель вычислений, например, МТ. Временем работы МТ M на входе x назовем число шагов $t_M(x)$ до завершения ее работы. Если машина не останавливается на этом входе, то будем считать, что $t_M(x)$ не определено. Соответственно, объем памяти $s_M(x)$ — это число ячеек, которые используются МТ в ходе работы (на которые она заходила). Очевидно, что всегда $s_M(x) \le t_M(x)+1$. Однако, анализ времени работы при конкретных входных данных не представляет особого интереса, поэтому определим функцию $T_M(n) := \max\{t_M(x) : |x| = n\}$, где $|x|$ — длина слова x. Это **временна́я сложность** алгоритма. Если $t_M(x)$ не определено для какого-то из этих слов, то и $T_M(n)$ тоже не определено. Таким образом, $T_M(n)$ представляет наихудший случай среди слов длины n. Аналогично можем определить функцию **емкостной сложности** $S_M(n) := \max\{s_M(x) : |x| = n\} \le T_M(n) + 1$.

Пример 4. Рассмотрим МТ из Примера 2. Для любого входа длины n при копировании каждого символа она делает $2n + 1$ шагов, и в конце еще n шагов для снятия меток со второй половины слова. Таким образом, $T_M(n) = n(2n+1)+n = 2n^2+2n$. При этом $S_M(n) = 2n+1$ (если учитывать последний пробел).

Анализ МТ из Примера 3 несколько сложнее. Пусть $|x| = n$, и в нем по k символов a и b, т.е. $x \in L(M)$. Тогда при поиске каждого парного элемента (всего k раз) машина делает не больше $2(n-k)$ шагов, и еще n шагов в сумме, чтобы прочитать все слово x. Если же $x \notin L$, то какого-то парного элемента не существует, и машина доходит до конца входного слова. В любом случае,

$$T_M(n) \leq \max_k (2k(n-k)+n) \leq \frac{n^2}{2} + n,$$

и если $k = \lfloor \frac{n}{2} \rfloor$, то оценка достигается с точностью ± 1. Таким образом, наихудший случай для данной МТ — слова вида

$$x = a^k b^k := \underbrace{a \dots a}_{k \text{ раз}} \underbrace{b \dots b}_{k \text{ раз}}.$$

Как несложно видеть, МТ не выходит за пределы входа (за исключением первого пробела после него), поэтому $S_M(n) = n + 1$.

При теоретическом анализе сложности алгоритма, как правило, нас не интересуют конкретные константы в функциях, а только их асимптотики при длине входа $|x| \to \infty$.

- Функция натурального аргумента $g(n) = O\big(f(n)\big)$, если существует такая константа $C > 0$ и число n_0, что для всех $n \geq n_0$ выполняется $|g(n)| \leq C|f(n)|$.
- $g(n) = o\big(f(n)\big)$, если $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$.
- $g(n) = \Omega\big(f(n)\big)$, если $f(n) = O\big(g(n)\big)$.
- $g(n) = \omega\big(f(n)\big)$, если $f(n) = o\big(g(n)\big)$.
- $g(n) = \Theta\big(f(n)\big)$, если $g(n) = O\big(f(n)\big)$ и $f(n) = O\big(g(n)\big)$.

Например, в рассмотренных выше примерах $T_M(n) = \Theta(n^2)$, $S_M(n) = \Theta(n)$.

1.4. Эквивалентность различных видов МТ

Во-первых, рассмотрим МТ с односторонней (бесконечной вправо) лентой. Очевидно, это частный случай МТ. Тем не менее, они эквивалентны: для любой МТ с двусторонней лентой можно построить эквивалентную (дающую тот же результат на том же входе) МТ с полубесконечной лентой. Действительно, пусть четные ячейки ленты соответствуют левой половине, а нечетные — правой. Тогда мы должны хранить в состоянии машины информацию о том, “на какой половине” находится головка. Шаг i в нечетной ячейке соответствует шагам $2i$, а в четной — шагам $-2i$. Отдельно нужно учесть переход через начало ленты, когда меняется четность. Как видим, число шагов на такой МТ $T_{M'}(n) = O\big(T_M(n)\big)$ (если не учитывать переписывание результата в правильном порядке).

Во-вторых, МТ с несколькими лентами: она определяется аналогично обычной МТ, но у нее несколько головок (своя для каждой ленты), которые могут перемещаться независимо, и функция перехода

$$\delta \colon Q \times \Pi^k \to Q \times \Pi^k \times \{-1, 0, +1\}^k,$$

где k — число лент. Бывает удобно считать, что вход подается на первую ленту, а результат считывается со второй.

МТ с несколькими лентами тоже эквивалентна стандартной. Пусть, например, у МТ M число лент $k = 2$. Построим эквивалентную одноленточную МТ M': будем считать, что нечетные ячейки соответствуют первой ленте, а четные — второй. Добавим также специальные символы алфавита с меткой, которые указывают, где находится головка. МТ действует следующим образом: находит первую головку, делает шаг, соответствующий данной ленте, затем находит вторую головку и делает там шаг, как на другой ленте. В состояниях МТ должна хранить состояние исходной многоленточной МТ и информацию о символах, над которым расположены головки. Несложно видеть, что для такой машины временная сложность будет $T_{M'}(n) = O\big((n + S_M(n))T_M(n)\big) = O\big(|x|T_M(n) + T_M^2(n)\big)$, т.к. чтобы сымитировать один шаг исходной МТ, надо найти положение каждой головки, что делается за $O(|w|) = O\big(n + S_M(n)\big)$, если w — текущее слово, записанное на ленте. Так как в большинстве разумных случаев $S_M(n) = \Omega(n)$ (машина должна хотя бы считать входные данные), то тогда $T_{M'}(n) = O\big(T_M^2(n)\big)$, значит, многоленточная МТ M *полиномиально сводится* к одноленточной M'.

Пример 5. Построим МТ с двумя лентами, которая разрешает язык $L_=$, как в Примере 3, но за линейное время. т.е. $T_M(n) = \Theta(n)$. Идея состоит в том, чтобы использовать вторую ленту как счетчик, увеличивая значение на 1 за каждый символ a, и уменьшая на 1 за каждый символ b. Изначально на первой ленте записано входное слово x, вторая лента пустая.

- $\Pi = \{a, b, c, s, \sqcup\}$;
- $Q = \{q_0, count, Accept, Reject\}$, $F = \{Accept, Reject\}$;
- функция перехода:

$$\begin{aligned}
&\delta(q_0, \alpha, \sqcup) = (count, \alpha, s, 0, 0),\ \ \alpha \in \Sigma;\\
&\delta(count, a, \beta) = (count, a, \beta, +1, +1),\ \ \beta \in \{\sqcup, s\};\\
&\delta(count, b, \beta) = (count, b, \beta, +1, -1);\\
&\delta(count, \sqcup, s) = (Accept, \sqcup, s, 0, 0);\\
&\delta(count, \sqcup, \sqcup) = (Reject, \sqcup, \sqcup, 0, 0).
\end{aligned}$$

Эта МТ проходит по входному слову всего один раз, и $t_M(x) = |x| + 1$ для любого x, следовательно, $T_M(n) = \Theta(n)$, в отличие от $\Theta(n^2)$ для одноленточной МТ.