# ЛИКБЕЗ

## Лекция 1: Теорема о рекуррентных оценках. Машины Тьюринга. Основы теории вычислимости. Основы теории сложности вычислений.

Дмитрий Ицыксон

ПОМИ РАН

23 сентября 2007

# План

- Теорема о рекуррентных оценках
- Машины Тьюринга
- Элементы теории вычислимости: разрешимые и перечислимые языки
- Элементы теории сложности: классы **P**, **LOGSPACE**, **NP**. **NP**-полнота.

# Литература


1. Н. К. Верещагин, А. Шень. Вычислимые функции.
2. А.Китаев, А.Шень, М.Вялый. Классические и квантовые вычисления.
3. Т. Кормен, Ч. Лейзерсон, Р. Ривест. Алгоритмы. Построение и анализ.
4. C.H. Papadimitriou. Computational complexity.

# Вопрос

## Вопрос

Все знают, что

- Полиномиальные алгоритмы эффективны;
- Неизвестно эффективного алгоритма разложения натурального числа на множители.

Рассмотрим следующий алгоритм разложения числа $n$ на множители:

- Перебираем все $2 \leq k \leq \sqrt{n}$;
- Если $n$ делится на $k$, то $n = k \cdot \frac{n}{k}$.

Сложность алгоритма не превосходит $\sqrt{n} \cdot n \leq n^2$, т.е. полиномиальна от $n$.

В чем проблема?

# Ответ

## Ответ

Сложность алгоритма измеряется как функция от длины записи входных данных. Число $n$ записывается $l = \log_{10} n$ знаками. $n^2 = 10^{2 \log_{10} n} = 10^{2l}$ — экспонента от $l$.

## Определение

Сложность алгоритма в худшем случае, как функция от длины входа $n$ — это максимум времени работы по всем входам длины $n$.

Сложность алгоритмов обычно измеряют асимптотически:

- Операции исполняются различное время и точной константы никто не знает

Хороший ли алгоритм, если его сложность $n + C$?

- Да, он линейный!
- А если $C$ операций выполняется 1000 лет?

# $O$, $o$, $\Omega$, $\Theta$

## Обозначения

Функции $f, g : \mathbb{N} \to \mathbb{R}_+$

- $f(n) = O(g(n))$, если $\exists c > 0 \exists n_0 \forall n > n_0, f(n) < cg(n)$;
- $f(n) = o(g(n))$, если $\lim\limits_{n \to \infty} \frac{f(n)}{g(n)} = 0$;
- $f(n) = \Omega(g(n))$, если $\exists c > 0 \exists n_0 \forall n > n_0, f(n) > cg(n)$;
- $f(n) = \Theta(g(n))$, если
  $\exists c_1, c_2 > 0 \exists n_0 \forall n > n_0, c_1 g(n) < f(n) < c_2 g(n)$;

$$f(n) = \Theta(g(n)) \iff \begin{cases} f(n) = O(g(n)) \\ g(n) = O(f(n)) \end{cases}$$

# Полезно знать

- $f(n), g(n)$ — многочлены, $\deg f = \deg g \implies f = \Theta(g)$;
- $f(n), g(n)$ — многочлены, $\deg f < \deg g \implies f = o(g)$;
- $\log_2 n = o(n^k)$;
- $\log_2 n = \Theta(\log_a n) = \Theta($ количество цифр в числе $n)$;
- $n^k = o(2^n)$;
- $n^k = o(n^{\log n})$.

# Пример: сортировка фон-Неймана

## Алгоритм

1. Отсортировать первую половину массива;
2. Отсортировать вторую половину массива;
3. Результат слить.

## Анализ сложности

- Слияние упорядоченных массивов $\Theta(n)$;
- Рекуррентное соотношение: $T(n) = 2T(\lceil \frac{n}{2} \rceil) + \Theta(n)$.

# Теорема о рекуррентных оценках

## Теорема

$$T(n) = \begin{cases} d, & n = 1 \\ aT(\lceil \frac{n}{b} \rceil) + \Theta(n^{\alpha}), & n > 1 \end{cases}$$

1. Если $\alpha < \log_b a$, то $T(n) = \Theta(n^{\log_b a})$;
2. Если $\alpha = \log_b a$, то $T(n) = \Theta(n^{\log_b a} \log n)$;
3. Если $\alpha > \log_b a$, то $T(n) = \Theta(n^{\alpha})$.

## Примеры

1. $T(n) = 9T(\lceil \frac{n}{3} \rceil) + n$: $\log_3 9 = 2$, $T(n) = \Theta(n^2)$;
2. $T(n) = 2T(\lceil \frac{n}{2} \rceil) + cn$: $\log_2 2 = 1$, $T(n) = \Theta(n \log n)$;
3. $T(n) = 2T(\lceil \frac{n}{2} \rceil) + cn^{\frac{3}{2}}$: $\log_2 2 = 1$, $T(n) = \Theta(n^{\frac{3}{2}})$.

# Идея доказательства

Для простоты считаем, что $n = b^k$, $T(n) = aT(\frac{n}{b}) + cn^\alpha$.
$T(n) = aT(\frac{n}{b}) + cn^\alpha = a(aT(\frac{n}{b^2}) + c(\frac{n}{b})^\alpha) + cn^\alpha$
$= c(n^\alpha + a\frac{n^\alpha}{b^\alpha} + a^2\frac{n^\alpha}{b^{2\alpha}} + \cdots + a^k\frac{n^\alpha}{b^{k\alpha}})$
$= cn^\alpha(1 + (\frac{a}{b^\alpha}) + (\frac{a}{b^\alpha})^2 + \cdots + (\frac{a}{b^\alpha})^k)$

1. $\alpha < \log_b a \iff a > b^\alpha \iff \frac{a}{b^\alpha} > 1$.
   $T(n) = \Theta(n^\alpha(\frac{a}{b^\alpha})^k) = \Theta(a^k) = \Theta(a^{\log_b n}) = \Theta(a^{\log_b a \cdot \log_a n}) = \Theta(n^{\log_b a})$;
2. $\alpha = \log_b a \iff a = b^\alpha \iff \frac{a}{b^\alpha} = 1$.
   $T(n) = \Theta(n^\alpha k) = \Theta(n^\alpha \log_b n) = \Theta(n^{\log_b a} \log n)$;
3. $\alpha > \log_b a \iff a < b^\alpha \iff \frac{a}{b^\alpha} < 1$.
   $T(n) = \Theta(n^\alpha)$.

# Модели вычислений

## Зачем они нужны

- Математическое определение понятия алгоритм;
- Строгое определение сложности алгоритма;
- Возможность что-то доказывать про все алгоритмы. Доказывать невозможность алгоритма.

## Какие Вы знаете модели вычисления?

Я знаю: $\lambda$-исчисление, машина Тьюринга, PAM-машина, машина Поста, нормальные алгоритмы Маркова...
Почти любой язык программирования может выступать в роли модели вычисления, если есть возможность использовать неограниченное количество памяти.

## Алфавит

Алфавит — это некоторое конечное множество символов.
Стандартное обозначение: $\Sigma$.

## Слова в алфавите

Слово — это конечная последовательность символов. Если $\Sigma$ — алфавит, то $\Sigma^*$ — множество всех слов в алфавите.
Например: $\Sigma = \{a, b\}$, то $\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, \dots\}$, где $\lambda$ — пустое слово.

# Языки и функции

## Язык

Языком над алфавитом $\Sigma$ называется подмножество $\Sigma^*$.

## Примеры языков

Язык простых чисел, язык четных чисел, язык двудольных графов, язык выполнимых формул в КНФ и т.д.

## Соглашение

Считаем, что любой алгоритм либо проверяет принадлежность входного слова некоторому языку, либо вычисляет значение функции $\Sigma^* \longrightarrow \Sigma^*$.

# Машина Тьюринга

- Бесконечная в одну сторону лента, разделенная на ячейки. В самой левой ячейке написан символ $\triangleright$.
- $Q$ — конечное множество состояний. $q_0 \in Q$ — начальное состояние. $q_f \in Q$ — конечное состояние.
- $\Sigma$ — алфавит символов, которые могут быть записаны на ленте. $\triangleright$, " _ " $\in \Sigma$.
- Головка машины указывает на одну из ячеек ленты
- Правило перехода: $\delta : \Sigma \times Q \to \Sigma \times Q \times \{\rightarrow, \leftarrow, \bullet\}$
- Согласно правилу перехода машина по символу, на который указывает головка, и по текущему состоянию, пишет на это место новый символ, переходит в новое состояние и, возможно, сдвигает головку на 1 символ влево или вправо.
- Начинает работу в состоянии $q_0$, головка указывает на первый символ. Заканчивает в состоянии $q_f$.

# Пример

## Прибавление единички в 2-ой системе счисления

Для простоты считаем, что число записано задом наперед.

- $(q_0, 0) \mapsto (q_f, 1, \bullet)$;
- $(q_0, 1) \mapsto (q_0, 0, \rightarrow)$;
- $(q_0, \_) \mapsto (q_f, 1, \bullet)$.

# Пример

## Прибавление единички в 2-ой системе счисления

Теперь число записано нормально

- $(q_0, 0) \mapsto (q_0, 0, \rightarrow)$;
- $(q_0, 1) \mapsto (q_0, 1, \rightarrow)$;
- $(q_0, \_) \mapsto (q_1, \_, \leftarrow)$;
- $(q_1, 0) \mapsto (q_f, 1, \bullet)$;
- $(q_1, 1) \mapsto (q_1, 0, \leftarrow)$;
- $(q_1, \triangleright) \mapsto (q_2, \triangleright, \rightarrow)$;
- $(q_2, 0) \mapsto (q_3, 1, \rightarrow)$;
- $(q_3, 0) \mapsto (q_3, 0, \rightarrow)$;
- $(q_3, \_) \mapsto (q_f, 0, \bullet)$.

# Машина Тьюринга

Вход машины Тьюринга — то, что записано на ленте. За входом следует бесконечное число пробелов.
Выход машины Тьюринга — то, что записано на ленте после того, как машина пришла в конечное состояние.
Если машина Тьюринга проверяет принадлежность языку, то удобно иметь два конечных состояния: $q_{yes}$ и $q_{no}$.
Машина Тьюринга может:

- закончить работу;
- работать бесконечно.

# Сложностные параметры

## Время

- Временем работы машины Тьюринга на входе $x$ называем количество шагов, которое машина делает, чтобы прийти в конечное состояние.
- Временной сложностью машины Тьюринга называем максимум по всем входам длины $n$ времени работы машины на этих входах.

## Память

- Сложностью по памяти работы машины Тьюринга на входе $x$ называем количество ячеек, в которых побывала головка машины.
- Емкостной сложностью машины Тьюринга называем максимум по всем входам длины $n$ сложности по памяти работы машины на этих входах.

# Многоленточная машина Тьюринга

Есть $k$ лент, головка есть на каждой ленте.
Правило перехода: $\delta : \Sigma^k \times Q \to \Sigma^k \times Q \times \{\rightarrow, \leftarrow, \bullet\}^k$

## Факт

По любой многоленточной машине Тьюринга можно построить одноленточную машину Тьюринга, вычисляющую ту же функцию. Причем сложностные характеристики этой машины будут лишь в полином раз хуже.

## Замечание

Иногда удобно считать, что машина снабжена специальной входной лентой, доступной только для чтения и выходной лентой, доступной для записи, но без исправлений.

# Недетерминированная машина Тьюринга

- Правила перехода неоднозначны. Возможно, что существует несколько правил для одной пары (символ, состояние);
- НМТ принимает слово $x$, если существует последовательность легальных шагов, приводящих в состояние $q_{yes}$;
- Определение не симметрично относительно ответов *yes* и *no*;
- Можно считать, что машина снабжена дополнительной лентой, доступной только для чтения слева направо. На этой ленте записана подсказка (какое из правил сейчас применять). Машина принимает $x$, если существует подсказка, с которой она попадет в состояние $q_{yes}$;
- Можно доказать, что если некоторая НМТ проверяет принадлежность языку $L$, то существует и детерминированная МТ, проверяющая принадлежность языку $L$.

# Сложностные параметры НМТ

Время и память НМТ считаются, как максимум по всем вариантам применения правила до прихода в конечное состояние. Если машина не останавливается при каком-то выборе правил, то время работы считается бесконечным.

# Тезис Черча-Тьюринга

Любой алгоритм можно реализовать в виде машины Тьюринга.

# Элементы теории вычислимости

- Машину Тьюринга можно записать: алфавит, состояния, правила... Каждой МТ соответствует строчка в некотором алфавите.
- Машин Тьюринга счетное число.
- Универсальная машина Тьюринга. Существует такая машина Тьюринга $U$, которая по записи машины Тьюринга $M$ и входу $x$ моделирует поведение машины $M$ на входе $x$. При этом сложностные параметры машины $U$ не более, чем в полином от записи $M$ и $|x|$ хуже.

# Разрешимые и перечислимые языки

$\Sigma$ — алфавит, $L \subset \Sigma^*$ — язык.
Язык $L$ называется алгоритмически разрешимым, если существует такая машина Тьюринга $M$, что

$$\begin{cases} x \in L \iff M(x) \text{ останавливается в состоянии } q_{yes} \\ x \notin L \iff M(x) \text{ останавливается в состоянии } q_{no} \end{cases}$$

Язык $L$ называется перечислимым, если существует такая машина Тьюринга $M$, что

$$\begin{cases} x \in L \iff M(x) \text{ останавливается в состоянии } q_{yes} \\ x \notin L \iff M(x) \text{ не останавливается} \end{cases}$$

# Простейшие свойства

## Лемма 1

Любой разрешимый язык является перечислимым.

Достаточно состояние $q_{no}$ заменить на $q_\infty$ и добавить правила $(q_\infty, *) \mapsto (q_\infty, *, \rightarrow)$

## Лемма 2

Язык $L$ — перечислим $\iff$ существует МТ $M'$, которая, работая на пустом входе, рано или поздно напечатает любой элемент языка $L$ без повторений. (Машина $M'$ может работать бесконечно долго).

$\Leftarrow$ Машина $M$ будет ждать, пока $M'$ напечатает слово $x$.

$\Rightarrow$ $M'$ моделирует $M$: 1 шаг на первом входе, 2 шага на первом, 2 шага на втором, 3 шага на первом, втором, третьем, 4 шага...

# Вопросы

## Существуют ли неперечислимые языки?

Существуют, так как машин Тьюринга счетно, а языков континуум.

Неконструктивное доказательство

## Существуют ли алгоритмически неразрешимые, но перечислимые языки?

Да.

# Пример неперечислимого языка

- Все записи машин Тьюринга можно перенумеровать с помощью алгоритма.
- Запись $< n >$ обозначает МТ с номером $n$.
- Рассмотрим язык
  $L = \{n | < n > \text{ не останавливается на входе } n\}$
- Пусть $L$ перечислим алгоритмом с номером $k$.
- Если $k \in L$, то $< k >$ не останавливается на $k \implies < k >$ не перечисляет $L$.
- Если $k \notin L$, то $< k >$ останавливается на $k \implies < k >$ не перечисляет $L$.
- Противоречие. Значит $L$ не перечисляется никаким алгоритмом.
- $L$ — алгоритмически неразрешим.

# Пример перечислимого, но не разрешимого языка

- $\overline{L} = \{n | < n > \text{останавливается на входе } n\}$;
- $\overline{L}$ неразрешим, так как иначе и язык $L$ был бы разрешимым;
- $\overline{L}$ перечислим: моделируем машину $< n >$ на входе $n$ и ждем, пока она остановится.

# Комментарий

- Задача остановки МТ: по МТ и ее входу определить, остановится она или нет. Эта задача алгоритмически неразрешима;
- Метод доказательства: диагонализация;
- Все результаты об алгоритмической неразрешимости используют неразрешимость задачи остановки МТ.

## Великая теорема Ферма

Если бы задача остановки была бы разрешима, то можно было бы доказать Великую теорему Ферма так:

1. Машина $M$ на пустом входе перебирает все $x, y, z, n \in \mathbb{N}, n > 2$ и останавливается, если $x^n + y^n = z^n$.
2. Узнаем, останавливается ли машина $M$ на пустом входе.

# Элементы теории сложности вычислений

## Класс **P**

Язык $L$ принадлежит классу **P**, если для него существует алгоритм, время работы которого полиномиально от длины входных данных, который распознает этот язык.

$\widetilde{P}$ — класс функций, вычислимых за полиномиальное время

Принято считать, что задачи, для которых существует полиномиальный алгоритм, реально решать на практике. Хотя это не всегда так...

# LogSpace $\subseteq$ P

- **LogSpace** — это класс языков, которые распознаются машинами, которые используют $O(\log n)$ памяти. (Вход и выход не считаются, можно считать, что для них есть отдельная лента).
- Докажем, что **LogSpace** $\subseteq$ **P**.
- Конфигурация МТ: содержимое лент, текущее состояние, положение головки.
- У **LogSpace**-машины состояний $|Q| = O(1)$;
- положений головки $O(n^2)$;
- содержимого лент: $|\Sigma|^{c \log n} = O(n^c)$;
- Итого: $O(n^{c+2})$ конфигурации.
- Если две конфигурации повторились, то машина не закончит работу.
- Вывод: время работы такой машины полиномиально.

# Классы $\widetilde{\mathbf{NP}}$ и $\mathbf{NP}$

- $R$ — бинарное отношение, т.е. $R \subseteq \Sigma^* \times \Sigma^*$;
- $R$ называется полиномиально ограниченным, если $\exists k$ $\forall x, y, (x, y) \in R \implies |y| < |x|^k$;
- $R$ называется полиномиально проверяемым, если существует полиномиальный алгоритм, который по $(x, y)$ проверяет, верно ли, что $(x, y) \in R$.
- Задача поиска, ассоциированная с $R$: по $x$ найти $y$ так, чтобы $(x, y) \in R$.
- Язык, ассоциированный с $R$: $L = \{x | \exists y : (x, y) \in R\}$.
- $\widetilde{\mathbf{NP}}$ — класс задач поиска, ассоциированных с полиномиально проверяемыми и полиномиально ограниченными отношениями.
- $\mathbf{NP}$ — класс языков.

# Примеры полиномиально ограниченных и проверямых отношений

1. $R_{SAT}$: пары (формула в КНФ, ее выполняющий набор). $((x \vee y) \wedge (\neg x \vee \neg y), [x \mapsto 1, y \mapsto 0]) \in R_{SAT}$
2. $R_{HAM}$: пары (граф, его Гамильтонов путь);
3. $R_{CLIQUE}$: пары (граф + число $k$, его клика размера $k$);
4. $R_{3-COLORIG}$: пары (граф, раскраска его вершин в 3 цвета правильным образом);

# P vs NP

## Предложение: $\mathbf{P} \subseteq \mathbf{NP}$

Если $L \in \mathbf{P}$, можно рассмотреть $R = \{(x, \lambda), x \in L\}$, где $\lambda$ — пустое слово.

## Замечание: язык $L \in \mathbf{NP}$ может быть распознан полиномиальной НМТ

Вторую часть отношения $R$ можно использовать, как подсказку к НМТ. Полиномиальность следует из полиномиальной проверяемости $R$.

## Открытый вопрос: является ли включение $\mathbf{P} \subseteq \mathbf{NP}$ строгим

## Сведение по Левину

Задача поиска $R_1$ сводится к задаче поиска $R_2$, если существуют такие полиномиальные алгоритмы $f$ и $g$, что

- $\exists y R_1(x, y) \iff \exists z R_2(f(x), z)$;
- $R_1(x, g(x, y)) \iff R_2(f(x), y)$.

## Сведение по Куку

Задача поиска $R_1$ сводится к задаче поиска $R_2$, если задачу поиска $R_1$ можно решить за полиномиальное время, при условии, что задачу $R_2$ мы умеем решать за 1 шаг.

## Лемма

Если $R_1$ сводится к $R_2$, то если $R_2$ можно решить за полиномиальное время, то и $R_1$ можно решить за полиномиальное время.

# $\widetilde{\mathbf{NP}}$-трудные задачи

## Определение

Задача поиска $R$ называется **$\widetilde{\mathbf{NP}}$-трудной**, если все задачи поиска из $\widetilde{\mathbf{NP}}$ сводятся к ней.

Задача поиска $R$ называется **$\widetilde{\mathbf{NP}}$-полной**, если она $\widetilde{\mathbf{NP}}$-трудна и принадлежит $\widetilde{\mathbf{NP}}$.

# Задача об ограниченной остановке

Определим отношение $R_{BH}$:
$(< M, x, \underbrace{11\ldots1}_{t \text{ штук}} >, y) \in R_{BH} \iff$ МТ $M$ на входе $(x, y)$
останавливается за не более, чем $t$ шагов в принимающем состоянии $(q_{yes})$.

## Теорема: $R_{BH}$ есть $\widetilde{\mathbf{NP}}$-полная

- $R_{BH} \in \widetilde{\mathbf{NP}}$, так как можно проверить за полиномиальное время моделированием;
- $R_{BH}$ полиномиально ограниченно: $|y| < t$;
- Пусть $R \in \widetilde{\mathbf{NP}}$, пусть $A$ — проверяющая Машина Тьюринга, ограниченная полиномом $n^k$.
- Сведение: $x \mapsto < A, x, \underbrace{11\ldots1}_{n^k \text{ штук}} >$

# Теорема Кука-Левина

## $R_{SAT}$ есть $\widetilde{\mathbf{NP}}$-полная

Идея доказательства: с помощью КНФ формул кодируется машина Тьюринга, шаги машины Тьюринга. И задача об ограниченной остановке $R_{BH}$ сводится к $R_{SAT}$.

# Сведение для языков

## Сведение по Карпу

Язык $L_1$ сводится к языку $L_2$, если существует полиномиально вычислимая функция $f$, что $x \in L_1 \iff f(x) \in L_2$.

## Определение

Язык $L$ называется **NP** – трудным, если любой язык из **NP** сводится к нему. И **NP** -полным, если он к тому же пренадлежит **NP**.

## Замечание

Сведение по Левину для задач поиска влечет сведение по Карпу для языков. Значит, язык выполнимых формул $SAT$ является $NP$-полным.

# От задач поиска к языкам

## $R_{SAT}$ сводится по Куку к SAT

Если формула $\varphi$ выполнима, то выполнима либо $\varphi[x \mapsto 1]$, либо $\varphi[x \mapsto 0]$. Выбираем правильное значение переменной $x$ и подставляем его.

## Замечание: не всегда задача поиска сводится к языку

Пример: проверить число на простоту можно за полиномиальное время, а раскладывать на множители быстро никто не умеет.

# Упражнения и задачи

1. Докажите, что нет алгоритма, который определял бы, закончит ли МТ работу на пустом входе.
2. Существует ли алгоритм, проверяющий, работает ли данная МТ полиномиальное время или нет?
3. Докажите, что не существует алгоритма, который определил бы по МТ $M$ определил бы, является ли последовательность $M(1), M(2), M(3) \ldots$ периодической с некоторого места.
4. Докажите **NP**-полноту модифицированной задачи об ограниченной остановке: $(< M, x >, y) \in R'_{BH}$, если МТ $M$ останавливается в принимающем состоянии за не более, чем $|x|^2$ шагов.