

Конспекты по дискретной математике

Анатолий Коченюк, Георгий Каданцев, Константин Бац

2022 год, семестр 4

Последний семестр дискретной математики. Две больших темы: производящие функции (комбинаторика) и введение в теорию вычислимости.

1 Производящие функции

Рассмотрим последовательности $\{a_n\}_{n \in \mathbb{N}}, \{b_n\}_{n \in \mathbb{N}} \subset \mathbb{R}(\mathbb{C})$. Назовём эти последовательности A и B и будем почленную сумму обозначать кратко $A + B$. Это несколько неудобно и неестественно, об этих конвенциях нужно договариваться.

Вместо этого давайте рассмотрим формальный степенной ряд, у которого члены последовательности это коэффициенты ряда.

$$A(t) = a_0 + a_1 t + a_2 t^2 + \dots + a_n t^n + \dots$$

Тогда почленная сумма последовательностей будет соответствовать обычной сумме рядов $A(t) + B(t)$.

Чтобы сдвинуть последовательность на 1 вправо, можно просто умножить степенной ряд на x .

Можем рассмотреть степенной ряд-композицию $A(t^2) = a_0 + 0t + a_1 t^2 + \dots$. Это степенной ряд, соответствующий последовательности $a_0, 0, a_1, 0, a_2, \dots$

Таким образом, мы можем "оперировать" над последовательностью как единым целым, и это очень удобно.

Мы не рассматриваем степенные ряды с точки зрения, с которой на них смотрит мат. анализ: как способ приблизить функцию, с некоторым радиусом сходимости и т.д. У нас степенные ряды *формальные* и не всегда (всегда не) должны пониматься как функции, в которой в переменную можно подставить значение.

$\mathbb{R}[x]$ — кольцо многочленов с коэффициентами из кольца R , состоящий из формальных многочленов. $\mathbb{R}[x]^+$ — множество формальных степенных рядов.

Определение 1.0.1. Формальный степенной ряд $A(t)$ последовательности $\{a_n\}_{n \in \mathbb{N}}$ называется производящей функцией (generating function).

Название неудачное. Оно связано с другими корнями понятия производящей функции (они нужны не только в комбинаторике).

Определена сумма производящих функций и произведение

$$A(t)B(t) = C(t) \quad c_n = \sum_{i=0}^n a_i b_{n-i}$$

Несмотря на то, что мы работаем с бесконечным по размеру объектом, нам необходимо только конечное число элементов, чтобы посчитать каждый отдельный его член. Этот раздел дискретной математики не любит предельных переходов.

Определено умножение на скаляр.

$$\lambda A(t) = C(t) \quad c_n = \lambda a_n$$

Определено даже деление!

$$\frac{A(t)}{B(t)} = C(t); \quad b_0 \neq 0 \quad c_n = \frac{a_n - \sum_{i=0}^{n-1} c_i b_{n-i}}{b_0}$$

Так можно посчитать, например, что

$$C(t) = \frac{1}{1-t} = 1 + t + t^2 + \dots + t^n + \dots; \quad a_n = 1$$

Мы записали короткой (конечной) производящей функцией бесконечную последовательность. Более того, мы можем эту запись взять и производить с ней операции (умножать и складывать с другими производящими функциями).

$$\frac{1}{1-2t} = 1 + 2t + 4t^2 + \dots + 2^n t^n; \quad c_n = 2^n$$

Обобщая мы видим, что

$$\frac{1}{1-bt} = \sum_{n=0}^{\infty} b^n t^n = C(bt)$$

Вообще говоря,

$$A(t) = \sum a_n t^n \quad A(bt) = \sum a_n b^n t^n$$

Замечание. Если $b_0 = \pm 1$, $a_i, b_i \in \mathbb{Z}$, тогда $C = \frac{A}{B}$ с целочисленными коэффициентами $c_i \in \mathbb{Z}$.

$$\frac{1}{1-t-t^2} = 1 + t + 2t^2 + \dots + F_n t^n + \dots$$

Мы одной дробью породили целую последовательность Фибоначи!

А как быть, если мы хотим взять последовательность и найти представление для её производящей функции? Мы можем поступить так.

$$F_0 = 1, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2}$$

Отсюда $F(t) + F(t)t = \frac{F(t)-1}{t}$ (следует из операций над производящими функциями и рекуррентным соотношением последовательности фиббоначи).

А что с дифференцированием? Обыкновенная операция взятия производной (формального) степенного ряда позволяет нам умножать член последовательности на его номер.

$$A(t) \rightarrow A(t)' \cdot t$$

Эту операцию можно производить многократно, получая последовательность членов исходной п-ти в k степени.

Как найти представление производящей функции для последовательности $a_n = n$?

$$a_n = n * 1$$

Производящая функция для п-ти единиц это $\frac{1}{1-t}$. Тогда

$$A(t) = \left(\frac{1}{1-t} \right)' t = \frac{t}{(1-t)^2}$$

Формальное деление это подтверждает.

А что с интегрированием? С интегрированием всё не очень мило.

А что с композицией?

$$C(t) = A(B(t))$$

Здесь много проблем доставляет свободный коэффициент у B . Давайте его уберем — $b_0 = 0$. Теперь мы можем посчитать

$$c_n = \sum_{k=0}^n a_k \sum_{n=i_1+i_2+\dots+i_k} b_{i_1} b_{i_2} \dots b_{i_k}$$

Пример с доминошками.

Пример с деревьями.

1.1 Линейные рекуррентные последовательности. Регулярные производящие функции

Определение 1.1.1 (Линейные рекуррентные последовательности). Пусть даны первые k членов последовательности a_0, a_1, \dots, a_{k-1} . А все следующие члены определяются, как линейная комбинация k предыдущих.

$$a_n = a_{n-1} \cdot c_1 + a_{n-2} \cdot c_2 + \dots + a_{n-k} \cdot c_k.$$

Такая последовательность называется **линейной рекуррентной последовательностью**.

Пример. Числа Фибоначчи. $F_0 = 1, F_1 = 1, \forall n \geq 2 : F_n = F_{n-1} + F_{n-2}$

$$\frac{1}{1-t-t^2} = \sum_{i=0}^{\infty} F_i t^i.$$

$$\begin{aligned} \text{Обозначим } F(t) &= \sum_{i=0}^{\infty} F_i t^i = F_0 t^0 + F_1 t^1 + \sum_{i=2}^{\infty} F_i t^i = \\ &= 1 + t + \sum_{i=2}^{\infty} F_{i-1} t^i + \sum_{i=2}^{\infty} F_{i-2} t^i = 1 + t + t \cdot \sum_{i=1}^{\infty} F_i t^i + t^2 \sum_{i=0}^{\infty} F_i t^i = \\ &= 1 + t + t \cdot (F(t) - 1) + t^2 \cdot F(t) \implies F = 1 + t \cdot F + t^2 F \implies \\ F(t) &= \frac{1}{1-t-t^2}. \end{aligned}$$

Теорема 1.1.1. Пусть есть линейная рекуррентная последовательность порядка k :
 $a_0, a_1, \dots, a_{k-1}, \dots$

Даны $a_0, \dots, a_{k-1}, \forall n \geq k : a_n = \sum_{i=1}^k a_{n-i} \cdot c_i$.

Тогда $A(t) = \sum_{i=0}^{\infty} a_i t^i = \frac{P(t)}{Q(t)}$ — рациональная функция, где
 $Q(t) = 1 - c_1 t - c_2 t^2 - \dots - c_k t^k$, а $P(t) = \dots$

Доказательство. Обозначим

$$A(t) = \sum_{i=0}^{\infty} a_i t^i = \sum_{i=0}^{k-1} a_i t^i + \sum_{n=k}^{\infty} a_n t^n.$$

Сразу заменим последнюю сумму предположением из теоремы, получим

$$\begin{aligned} A(t) &= \sum_{i=0}^{k-1} a_i t^i + \sum_{n=k}^{\infty} t^n \sum_{i=1}^k a_{n-i} \cdot c_i = S + \sum_{i=1}^k c_i \sum_{n=k}^{\infty} a_{n-i} t^n = S + \sum_{i=1}^k c_i \cdot t^i \cdot \sum_{n=k-1}^{\infty} a_n t^n = \\ &= S + \sum_{i=1}^k c_i \cdot t^i \cdot (A(t) - A_{k-1}(t)) = X. \end{aligned}$$

Пусть $C(t) = \sum_{k=1}^k c_i t^i$, тогда $Q(t) = q - C(t)$. $X = S + C(t) \cdot A(t) - \sum_{k=1}^k c_i t^i A_{k-i}(t) = Y$.

Пусть $F(t) \% t^k = \sum_{i=0}^{k-1} f_i t^i$, тогда $A_{k-i}(t) = A(t) \% t^{k-1}$.

$$\sum_{k=1}^k c_i t^i A_{k-i}(t) \cdot A_{k-i}(t) = A(t) \% t^{k-i} = (C(t) \cdot A(t)) \% t^k \implies$$

$$A(t) = \sum_{i=0}^{k-1} a_i t^i + C(t) \cdot A(t) - (C(t) \cdot A(t)) \% t^k \implies A(t)(1 - C(t)) = ((1 - C(t)) \cdot A(t)) \% t^k$$

$$\implies A(t) = \frac{P(t)}{Q(t)}, \quad \text{где } Q(t) = 1 - C(t) = 1 - c_1 t - c_2 t^2 - \dots - c_k t^k,$$

$$P(t) = \left(\left(\sum_{i=0}^{k-1} a_i t^i \right) \cdot Q(t) \right) \mod t^k.$$

■

Пример. Для чисел фибоначчи: $a_0 = a_1 = 1, c_1 = c_2 = 1 \implies$

$$A(t) = \frac{(1+t) \cdot (1-t-t^2) \mod t^2}{1-t-t^2}.$$

$$a_0 = 6, a_1 = -3, c_1 = c_2 = 1 \implies A(t) = \frac{(6-3t) \cdot (1-t) \mod t^2}{1-t-t^2} = \frac{6-9t}{1-t-t^2}.$$

Доказательство в обратном направлении. Частный случай:

$$\frac{1}{1-C(t)} = A(t), \quad A(t) \cdot (1-C(t)) = 1,$$

$$t^0 = a_0 = 1, \quad t^1 : a_1 \cdot 1 - a_0 c_1 = 0, \quad t^2 : a_2 \cdot 1 - a_1 \cdot c_1 - a_0 c_2 = 0.$$

Посмотрим на некоторую производящую функцию, например $\frac{1-3t+6t^3}{1-t-t^2-t^4}$. Понимаем, что $a_n = a_{n-1} + a_{n-2} + a_{n-4}$.

$$a_0 = 1, \quad a_1 = 1 - 3 = -2, \quad a_2 = 1 - 2 = -1, \quad = -1 - 2 + 6 = 3$$

$$A(t) \cdot Q(t) = P(t). \quad \sum_{i=0}^n q_i \cdot a_{n-i} = p_n \quad a_n = p_n - \sum_{i=1}^k q_i \cdot a_{n-i}.$$

■

Пусть $a_0, a_1, \dots, a_{k-1}, \forall n \geq k : a_n = \sum_{i=1}^k a_{n-i} \cdot c_i$.

Задача: посчитать a_n .

Можно явно за $\mathcal{O}(n \cdot k)$.

Можно через возведение матрицы в степень за $\mathcal{O}(k^3 \log_2 n)$.

Потом мы научимся делать это за $\mathcal{O}(k^2 \log_2 n)$.

На самом деле, для одной и той же числовой последовательности можно получить несколько производящих функций.

$$A(t) = \frac{P(t)}{Q(t)} \cdot \frac{Q(-t)}{Q(-t)} = \frac{P(t) \cdot Q(-t)}{Q(t) \cdot Q(-t)}.$$

Например, для чисел Фибоначчи:

$$\frac{1}{1-t-t^2} \cdot \frac{1+t+t^2}{1+t+t^2} = \frac{1+t-t^2}{1-3t^2+t^4}. \quad F_n = F_{n-2} \cdot 3 - F_{n-4}.$$

Теорема 1.1.2. Для производящих функций, задающих рекуррентные соотношения эквивалентны следующие высказывания

- $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \dots +$
- $A(t) = \frac{P(t)}{Q(t)}$
- $a_n = \sum_{i=1}^b p_i(n) \cdot r^i$, где $r_i \in \mathbb{C}$

$$Q(t) = 1 - c_1 t - c_2 t^2 - \dots - c_k t^k$$

$P(t)$ определяет то, как надо подправить первые члены, чтобы получились те, которые нужны.

$$A(t)Q(t) - P(t) = 0.$$

Как посчитать r ?

$$\text{Пусть } Q(t) = 1 - rt,$$

$$a_n = r \cdot a_{n-1}$$

$$a_m = r \cdot a_{m-1}$$

$$a_{m+1} = r \cdot a_m$$

...

$$a_n = r^n \cdot \frac{a_{m-1}}{r^{m-1}}$$

$$\text{Пусть } Q(t) = (1 - r_1 t)(1 - r_2 t), \quad r_1 \neq r_2.$$

$$\text{Лемма 1.1.2.1. } Q(t) = \prod_{i=1}^n (1 - r_i t), \text{ где } r_i \neq r_j \quad \frac{P(t)}{Q(t)} = \sum_{i=1}^n \frac{P_i(t)}{1 - r_i t}$$

$$Q(t) = \sum_{i=1}^n d_i r_i^n.$$

r_i — числа, обратные корням многочлена Q . Если степень Q равно k , то Q имеет ровно k корней (с учетом кратности).

$$\begin{aligned} \text{Таким образом, } Q(t) &= q_k \prod_{i=1}^k (t - t_i) = (-1)^k q_k \prod_{i=1}^k \left(1 - \frac{t}{t_i}\right) \cdot t_i = \\ &= \left[(-1)^k q_k \cdot \prod_{i=1}^k t_k \right] \prod_{i=1}^k (1 - r_i t) = \alpha \prod_{i=1}^k (1 - R_i t). \end{aligned}$$

Почему нет корня 0? Потому что $Q(t)$ имеет вид $Q(t) = 1 - c_1 t - \dots$.

Пример. Рассмотрим числа Фибоначчи. $F(t) = \frac{1}{1 - t - t^2}$.

Корни $t_{1,2} = \frac{-1 \pm \sqrt{1+4}}{2}$, обратные корни $r_{1,2} = \frac{1 \mp \sqrt{5}}{2}$. Обратные корни разные — нам очень приятно.

$$Q(t) = \left(1 - \frac{1 - \sqrt{5}}{2} t\right) \left(1 - \frac{1 + \sqrt{5}}{2} t\right).$$

$$\frac{1}{(1 - r_1 t)(1 - r_2 t)} = \frac{c_1}{1 - r_1 t} + \frac{c_2}{1 - r_2 t}, \quad c_1(1 - r_2 t) + c_2(1 - r_1 t) = 1 \implies$$

$$\begin{cases} c_1 + c_2 = 1 \\ c_1(-r_2) + c_2(-r_1) = 0 \end{cases} \implies c_2 = \frac{-r_2}{r_1 - r_2} = \frac{-1 - \sqrt{5}}{2 \cdot (-\sqrt{5})} = \frac{5 + \sqrt{5}}{10}, \quad c_1 = \frac{5 - \sqrt{5}}{10}.$$

$$a_n = c_1 r_1^n = \frac{5 - \sqrt{5}}{10} \cdot \left(\frac{1 - \sqrt{5}}{2}\right)^n \quad b_n = c_2 r_2^n = \frac{5 + \sqrt{5}}{10} \cdot \left(\frac{1 + \sqrt{5}}{2}\right)^n \implies$$

$$f_n = \frac{5 - \sqrt{5}}{10} \cdot \left(\frac{1 - \sqrt{5}}{2}\right)^n + \frac{5 + \sqrt{5}}{10} \cdot \left(\frac{1 + \sqrt{5}}{2}\right)^n = \Theta\left(\left(\frac{1 + \sqrt{5}}{2}\right)^n\right).$$

Замечание. Если λ — единственный минимальный по модулю комплексный корень $Q(t)$, $A(t) = \frac{P(t)}{Q(t)}$,

то $a_n = \Theta\left(\frac{1}{\lambda^n}\right)$.

$$\frac{1}{(1 - rt)^2} = \frac{1}{1 - 2rt + r^2 t^2}. \quad a_0 = 1, \quad a_1 = 2r, \quad a_2 = 3r^2, \quad a_3 = 4r^3, \quad \dots, \quad a_n = (n+1)r^n.$$

$$\frac{1}{r} (r^n t^n)' = \frac{1}{r} \sum n r^n t^{n-1} = \sum n r^{n-1} t^{n-1} = \sum (n+1) r^n t^n.$$

$$\text{Лемма 1.1.2.2. } \frac{1}{1-rt} = \sum_{n=0}^{\infty} p_s(n) r^n t^n$$

Доказательство. Докажем по индукции.

1. База. $s = 0$ — просто

2. Переход. Далее много формул. $\left(\frac{1}{(1-rt)^s}\right)' = \frac{-r(-s)}{(1-rt)^{s+1}} \cdot \left(\sum_{n=0}^{\infty} p_s(n)r^n t^n\right)' = \sum_{n=1}^{\infty} n p_s(n) r^n t^{n-1} =$
 $\sum_{n=0}^{\infty} (n+1) p_s(n+1) r^{n+1} t^n \cdot \frac{1}{(1-rt)^{s+1}} = \sum_{n=0}^{\infty} \frac{n+1}{s} p_s(n+1) r^n t^n, p_{s+1}(n) = p_s(n+1) \frac{n+1}{s} = \sum_{i=0}^{s-1} p_{s,i}(n+1)^i \frac{n+1}{s}.$

$$p_{s,i} = \frac{a_{s,i}}{b}, \quad b = s!, \quad a_{s,i} \in \mathbb{Z}$$

■

Теорема 1.1.3. Пусть $A(t) = \frac{P(t)}{Q(t)}$, r_i — обратный корень кратности s_i Q_i , количество различных корней b . Тогда начиная с некоторого места (но точно, начиная с k) $a_n = \sum_{i=1}^b p_i(n) r_i^n$, $\deg p_i = s_i - 1$, $\sum_{i=1}^b s_i = k$.

Доказательство.

$$Q(t) = \prod_{i=1}^b (1 - r_i t)^{s_i}, \quad \frac{P(t)}{Q(t)} = \sum_{i=1}^b \frac{P_i(t)}{(1 - r_i t)^{s_i}}.$$

■

Если λ_i — единственный минимальный комплексный корень $Q(t)$ кратности s_i . Тогда $a_n = \Theta\left(\frac{n^{s_i-1}}{\lambda_i^n}\right)$.

Пример. $a_n = n^3$, $a_n = 4 \cdot a_{n-1} - 6 \cdot a_{n-2} + 4 \cdot a_{n-3} - a_{n-4}$.

Подберем поправку первых членов: $P(n) = t + 4t^2 + t^3$.

Утверждение 1.1.1. Асимптотическое поведение рекуррентности не зависит от начальных значений, оно зависит только от коэффициентов соотношений.

Утверждение 1.1.2. Пусть $\lambda_1, \lambda_2, \dots, \lambda_z$ — минимальные корни максимальной кратности.

$$\lambda_j = \frac{e^{i\varphi_j}}{r}, \quad \varphi_j = \frac{p_j}{q_j} \cdot 2\pi.$$

Пусть $\bar{q} = LCM(q_j)$. Тогда последовательность a_i имеет асимптотическое поведение при $i \% \bar{q} = const$.

1.2 Комбинаторика и производящие функции

Пример. Замещение прямоугольника $2 \times n$ доминошками вида 1×2 .

$$\frac{1}{1-t-t^2} = 2 + t + t^2 + t^2 + t^3 + t^3 + t^3 + \dots = 1 + t + 2t^2 + 6t^3 + \dots + F_n t^n$$

Комбинаторные объекты это конструкции, которые состоят из атомов и разных связей атомов между собой. Под атомом мы понимаем некоторую неделимую часть комб. объекта. Давайте все наши комбинаторные объекты сложим.

В этой сумме заменим каждый атом на t^ω , где ω — вес данного атома. Потом t^ω атомов одного объекта перемножим.

Вес объекта — сумма весов его атомов.

Пример. A — множество комбинаторных объектов. Давайте их просуммируем.

$$\Delta_1 + \Delta_2 + \Delta_3 + \dots$$

атом (неделимое) — то, что мы считаем.

$$t^{\omega(\Delta_1)} + t^{\omega(\Delta_2)} + t^{\omega(\Delta_3)} + \dots = \sum_{n=0}^{\infty} a_n t^n = A(t) — \text{производящая функция для объектов веса } t.$$

Определение 1.2.1 (Базовые объекты). $U = \{u\}$ $\omega(u) = 1$ $u(t) = t$ – производящая функция для этих комбинаторных объектов

$$B = \{a, b\} \quad \omega(a) = \omega(b) = 1 \quad B(t) = 2t$$

$$E = \{\varepsilon\} \quad \omega(\varepsilon) = 0 \quad E(t) = 1$$

$$E_k = \{\varepsilon_1, \varepsilon_2, \dots, \varepsilon_k\} \quad E_k(t) = k$$

$$D = \{a, A\} \quad \omega(a) = 1 \quad \omega(A) = 2 \quad D(t) = t + t^2$$

Операции конструирования

Определение 1.2.2 (Дизъюнктное объединение). A, B – множества комбинаторных объектов и $A \cap B = \emptyset$

Пусть $C = A \cup B$. Тогда производящая функция

$$\begin{aligned} C(t) &= A_1 + A_2 + \dots + B_1 + B_2 + \dots \\ &= t^{\omega(A_1)} + t^{\omega(A_2)} + \dots + t^{\omega(B_1)} + t^{\omega(B_2)} + \dots \\ &= A(t) + B(t) \end{aligned}$$

Определение 1.2.3 (Упорядоченная пара (прямое произведение)). Пусть A, B , $A(t), B(t)$ – их производящая функция. Определим пару C , как $A \times B = C = \{\langle a, b \rangle \mid a \in A, b \in B\}$.

$$C_n = C \cap \{x \mid \omega(x) = n\} \quad \langle a, b \rangle \omega(a) = i \quad \omega(b) = j \quad i + j = n \quad j = n - i.$$

$$C_n = \cup A_i \times B_{n-i}. \quad c_n = \sum_{i=0}^n a_i \cdot b_{n-i} \implies C(t) = A(t) \cdot B(t).$$

Замечание (Комбинаторный мысл прямого произведения). Пусть у нас есть объекты $A = A_1 + A_2 + \dots + A_k + \dots$, $B = B_1 + B_2 + \dots + B_k + \dots$

$$(A_1 + A_2 + \dots) \cdot (B_1 + B_2 + \dots) = A_1 \cdot B_1 + A_1 \cdot B_2 + \dots + A_2 \cdot B_1 + A_2 \cdot B_2 + \dots$$

$$\langle a, b \rangle \quad t^{\omega(a)} t^{\omega(b)} = t^{\omega(a) + \omega(b)}.$$

Определение 1.2.4 (Последовательность (sequence)). Определим последовательность из A , как $SeqA = \{\square, [A_1], [A_2], \dots, [A_1, A_2], [A_1, A_3], \dots\}$.

$$SeqA = \square \cup A_1 \cdot (\square + [A_1] + [A_2] + \dots) + A_2 \cdot (\square + \dots) = 1 + A \times SeqA.$$

$$B = SeqA \implies B(t) = 1 + A(t)B(t) \implies B(t) = \frac{1}{1-A(t)}.$$

Определение 1.2.5 (Последовательность (sequence), второй способ). $SeqA = A^0 \cup A^1 \cup A^2 \cup \dots \cup A^k \cup \dots$

A^i – декартова степень, последовательности длины i

$$B(t) = A(t)^0 + A(t)^1 + A(t)^2 + \dots + A(t)^k + \dots = \frac{1}{1-A(t)}$$

Пример. $SeqU = \{\square, [u], [u, u], [u, u, u], \dots\} = N$. $n_k = 1$, $U(t) = 1 \implies SeqU = \frac{1}{1-t}$.

$$SeqB = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, \dots\} = C. \quad c_n = 2^n, \quad B(t) = 2t \implies C(t) = \frac{1}{1-2t}, \quad c_n = 2 \cdot c_{n-1}.$$

$$C_n = 2^n \quad B(t) = 2t \quad C(t) = \frac{1}{1-2t}$$

$$\begin{aligned}
C_n &= 2C_{n-1} \\
SeqE &= \{\emptyset, [\varepsilon], [\varepsilon, \varepsilon], \dots\} = C \\
E(t) &= 1 \quad C(t) = \frac{1}{1-E(t)} = \frac{1}{1-1} = \frac{1}{0} \quad \odot \\
C_0 &= +\infty??
\end{aligned}$$

Пример. $C = SeqD = \{\varepsilon, a, aa, aA, A, Aa, AA, \dots\}$

$$C(t) = \frac{1}{1-D(t)} = \frac{1}{1-t-t^2}$$

a – одна вертикальная доминошка, вес 1. A – две горизонтальные доминошки, вес 2.

$$C = aC + AC \quad C(t) = tC(t) + t^2C(t)$$

Определение 1.2.6 (Множество). Обозначается Set или $PSet$.

$$B = \{a, A\} \quad SetB = \{\emptyset, \{a\}, \{A\}, \{a, A\}\}.$$

$$C = SetA.$$

$a \in A$ $B_a = \varepsilon + a$ – либо берём, либо не берём. C – декартово произведение по всем a .

$$C(t) = \prod_{a \in A} (1 + t^{\omega(a)}) = \prod_{n=0}^{\infty} (1 + t^n)^{a_n}.$$

Пример. Возьмем $U = \{u\}$. $SetU = C = \{\emptyset, \{u\}\}$. Найдём $C(t)$.

$$C(t) = \prod_{n=0}^{+\infty} (1 + t^n)^a = (1 + t)^1 = 1 + t.$$

Пусть $B = \{a, A\}$, $C = SetB$. Заметим, что $b_1 = 1, b_2 = 1$.

$$C(t) = \prod_{n=0}^{+\infty} (1 + t^n)^{b_n} = (1 + t)(1 + t^2) = \underbrace{1}_{\emptyset} + \underbrace{t}_a + \underbrace{t^2}_A + \underbrace{t^3}_{a,A}.$$

$$\prod_{n=0}^{\infty} (1 + t^n)^{a_n} = (a + t_0)^{a_0} \cdot \prod_{n=1}^{\infty} (1 + t^n)^{a_n} = 2^{a_0} \prod_{n=1}^{\infty} (1 + t^n)^{a_n}.$$

Определение 1.2.7 (Мультимножество). Обозначается $MSetA$.

Мы можем включить каждый объект $0, 1, 2, \dots$

$$\varepsilon + a + aa + \dots = Seq\{a\}.$$

$$a_1 \in A, a_2 \in A \implies Seq\{a_1\} \times Seq\{a_2\} = MSet\{a_1, a_2\}.$$

$$MSetA = \prod_{a \in A} Seq\{a\}.$$

$$C(t) = \prod_{a \in A} Seq\{a\} = \prod_{a \in A} \frac{1}{1 - t^{\omega(a)}} = \prod_{n=1}^{\infty} \left(\frac{1}{1 - t^n} \right)^{a_n} = \prod_{n=1}^{\infty} (1 - t^n)^{-a_n}.$$

Пример. $U = \{u\}$ $C = MSetU$ $C(t) = \prod_{n=1}^{\infty} (1 - t^n)^{-u_n} = (1 - t)^{-1} = \frac{1}{1-t}.$

$$B = \{a, A\} \quad C = MSetB \quad b_1 = 1 = b_2.$$

$$C(t) = \prod_{n=1}^{\infty} (1 - t^n)^{-1} = (1 - t)^{-1} (1 - t^2)^{-1} = \frac{1}{(1-t)(1-t^2)}.$$

Асимптотика C_n .

$$Q(t) = (1 - t)(1 - t^2) = (1 - t^2)(1 + t).$$

Корни: $t = \pm 1$. Обратные корни $r = \pm 1$ Кратность $r_1 = 1 \quad s_1 = 2 \quad r_2 = -1 \quad s_2 = 1$.

$$(a_n + b) \cdot 1^n + c \cdot (-1)^n$$

$$\frac{1}{2}n + \frac{1}{2}(-1)^n + const$$

1.2.1 Циклы (cycle)

$B = \{a, b\}$ $CycB = \{\varepsilon, a, b, ab, aa, bb, aaa, aab, abb, bbb, aaaa, aaab, aabb, abab, abbb, bbbb, \dots\}$

Раньше мы называли такие комбинаторные объекты *ожерельями*.

$$C = CycB = \bigcup_{k=1}^{\infty} (CycA)_k.$$

$$C(t) = \sum_{k=1}^{\infty} C_k(t), \quad C_k(t) \text{ — производящая функция длины } k.$$

$C_k(t)$ — последовательности длины k с точностью до циклического сдвига.

S_k — последовательности длины k $= (A(t))^k \cdot C_k(t)/G$ G — группа циклических сдвигов.

$$C_{k,n} = \frac{1}{k} \cdot \sum_{i=0}^{k-1} |I(i)|.$$

Количество классов эквивалентности по лемме Бёрнсайда равно $\gcd(i, k)$. Внутри класса одинаковые объекты. Размер класса $\frac{k}{\gcd(i, k)}$.

$$\text{и кратно } \frac{k}{\gcd(i, k)} \cdot S_{\gcd(i, k)} \frac{n \cdot \gcd(i, k)}{k}.$$

$$C_{n,k} = \frac{\sum_{j=0}^{k-1} S_{\gcd(i, k), \frac{n \cdot \gcd(i, k)}{k}}}{k}$$

2 Формальные языки

Как объяснить компьютеру, что я вляется словом в нашем языке.

Пусть $L \subset L^*$. Мы знаем два способа задания языка.

1. ДКА = Р.В.

2. КСГ = МП-автомат.

А что мы вообще сделать с помощью компьютера? Ну что-то нельзя сделать из-за фикики: путешествовать во времени, ... А что нельзя сделать с помощью математики?

Есть два способа рассказать компьютеру о языке:

- Научить его распознавать слово из языка.

Например, ДКА. Нужен мета-язык описания конечных автоматов и само описание автомата.

- Научить компьютер порождать слова из языка.

У нас есть также мета-язык описания порождения языка и само описание.

Например, даем компьютеру парсер регулярных выражений, само регулярное выражение, компьютер строит дерево разбора и генерирует нам слова.

На сколько сильно мы можем усложнить грамматику нашего языка. Регулярные языки не умели генерировать палиндромы, КСГ не умели генерировать $1^n 2^n 3^n$. А на сколько еще мы можем усложнить наши описания?

Вообще, языков может быть 2^{Σ^*} — несчетное количество. Но для нас это не страшно, почти все из них описать не возможно. А что можно описать? Ну пусть то, что умеет понимать компьютер. **А что такое вообще компьютер?**

Для осознания мощности компьютеров существуют модели. Мы будем изучать **Машину Тьюринга**. Машина Тьюринга основывается на ленте. Так же существует, например, машина Маркова, там немного другой принцип.

Метод описания

Пусть у нас есть *современный x86* компьютер. Как описать модель такого компьютера? Ну есть почти неограниченная память и есть какие-то операции (сложение, умножение, вызов функции). Что такое программа для такого компьютера? Программа — $\underbrace{\hspace{1cm}}_{\text{строка}} \in \Pi^*$. На самом деле можно записать в битовом

формате $\mathbb{B} = \{0, 1\}$, то есть $\underbrace{\quad}_{\text{Description}} \in \mathbb{B}^*$. На самом деле, описание программ самый мощный инстру-

мен. Пусть есть описание конечного автомата, запишем его в константу, добавим код имплементации конечного автомата, получим описание языка от автомата на языке программ.

Утверждение 2.0.1 (Тезиси Тьюринга–Чёрча). Все, что можно выразить на «обычном компьютере» можно выразить на Машине Тьюринга.

Замечание. Почему это не утверждение или теорема? Надо доопределить понятие «обычный компьютер» и тогда получится какое-то утверждение.

А что вообще может выдавать программа? *Comilation error* не большая проблема, давайте в таком случае программа будет делать что-то конкретное, например *No*. Если *Runtime error*, пусть тоже будем возвращать что-то конкретное. *Memory limit* — не проблема, попросим пользователя добавить память или подождём, пока изобретут компьютер лучше. Самое интересное — *Time limit*, это обозначают \perp . Таким, образом наша программа будет возвращать три значения I, L, \perp . Тогда хочется добавить ограничение на время исполнения, правда, это немного ограничит класс языков.

Определение 2.0.1. Язык L разрешимый (рекурсивный), если \exists программа $p \forall x \in L \implies p(x) = 1, x \notin L \implies p(x) = 0$.

Определение 2.0.2. Язык L полуразрешимый (перечислимый, рекурсивно перечислимый), если \exists программа $p \forall x \in L \iff p(x) = 1$.

На самом деле полуразрешимые описания языков — мкисмальные по мощности. Разрешимый — максимальный по мощности прикладной способ описания.

Существуют не разрешимые и не полуразрешимые языки.

Метод порождения

Пусть у нас есть компьютер, который по описанию выводит список слов. Можно выводить первые n слов.

Опять же, понятно, что описание с помощью компьютера макисмальное по мощности.

Определение 2.0.3. Язык L перечислимый, если можно написать программу, которая выводит его слова.

Теорема 2.0.1. L полуразрешим $\iff L$ перечислим.

Определение 2.0.4. Градуированный лексикографический порядк — перечисление в порядке увеличения длины, а среди слов с равной длины лексикографически.

Доказательство. \implies

Неверный подход 1.

```
1  for (x ∈ Σ*)
2      if p(x):
3          print(x)
```

Не верный поход, попытка 2

```
1  for (TL = 1; True; TL++)
2      for (x ∈ \Sigma^{*})
3          if p |_T (x):
4              print(x)
```

Подход правильный:

```
1  for (TL = 1; True; TL++)
2      for (x ∈ Σ*[: TL])
3          if p |T (x):
4              if (x ∉ was):
5                  print(x)
6                  was.add(x)
```

⇐

Пусть у нас есть q — перечислитель L .

```
1 p(x):
2     while q.next() != x:
3         pass
4     return True
```

■

Пример непререцилимого языка

Программа набор из 0 и 1. Пусть A — предикат. $L_A = \{p \mid A(p)\}$ — формальный язык.

Определение 2.0.5 (Универсальный язык). $U = \{\langle p, x \rangle \mid p(x) = 1\}$.

Замечание. U — полурешим.

Доказательство. Давайте сделаем

```
1 inU(⟨p, x⟩):
2     return p(x)
```

■

Теорема 2.0.2. U не разрешим.

Доказательство. Пусть есть функция $\text{inU}(\langle p, x \rangle)$ — всегда завершается.

```
1 q(x):
2     if inU(⟨x, x⟩):
3         return 0
4     else:
5         return 1
```

Посчитаем $q(q)$.

Если $q(q) = 1 \implies \text{inU}(\langle q, q \rangle) = \text{false} \implies q(q) = 0$ (плохо).

Пусть $q(q) = 0 \implies \text{inU}(\langle q, q \rangle) = \text{true} \implies q(q) = 1$ (тоже плохо).

■

Теорема 2.0.3. Если A и \bar{A} — полурешим $\implies A$ — разрешим.

Утверждение 2.0.2. A разрешим $\implies \bar{A}$ — разрешим.

Доказательство теоремы. .

```
1 inA(x):
2     for (TL = 1; +∞):
3         if p |TL(x):
4             return 1
5         if q |TL(x):
6             return 0
```

■

Не полуразрешим

Пример. Язык дополнение до U не полуразрешим.

На самом деле эти множества биективны:

- Строки Σ^*
- Программы Prog
- Числа \mathbb{N}^+

$$\text{Prog} \xleftrightarrow{\text{id}} \Sigma^* \xleftrightarrow{\text{grad.order}} \mathbb{N}^+.$$

- Полуразрешимые языки \iff вычислимые функции $A \subset \mathbb{N} \rightarrow \{0, 1\}$.
- Разрешимые языки \iff всюду определенные (Hall) вычислимые функции $\mathbb{N} \rightarrow \{0, 1\}$.

2.1 А обязательно ли разрешать компьютеру зависать?

Определение 2.1.1. Язык программирования называется **полным** для A , если для любого перечислимого языка из A можно задать его описание на этом языке.

Иными словами, любую вычислимую функцию можно задать при помощи этого языка программирования.

Определение 2.1.2. Язык называется **вычислимым** (компилируемым), если по описанию и словы мы можем сказать подходит или нет.

Определение 2.1.3. Язык программирования **независающий**, если \forall программы и \forall слова он не зависнет.

Теорема 2.1.1. Не существует полного для разрешимых языков вычислимого, не зависающего метаязыка описания.

То есть, не существует вычислимой нумерации всех всюду определенных вычислимых функций.

Доказательство. Предположим, что она (нумерация) существует. Вот эта нумерация f_1, f_2, f_3, \dots

	x_1	x_2	x_3	x_4
f_1				
f_2				
f_3				
f_4				

$g(i) = f_i(x_i) + 1$, g — всюдуопределенная, вычислимая.
 $g(i) \neq f_i(x_i)$



Следствие 2.1.1.1. На конечном автомате нельзя интерпретировать конечный автомат.

2.2 Разрешимость

У нас есть пример неразрешимого языка. Это язык $U = \{\langle p, x \rangle \mid p(x) = 1\}$. А если мы хотим проверить еще какой-то язык? Можно ли куда-то замести под ковер рассуждения?

Определение 2.2.1. m — сведение (исторически many to one reduction, но в реальности это оказалось не удачным, поэтому называют mapping сведение).

Говорят язык $A \leq_m B$, если существует всюдуопределенная вычислимая функция f , такая, что $x \in A \iff f(x) \in B$.

Теорема 2.2.1. Если A не разрешимый, $A \leq_m B \implies B$ — не разрешимый.

Доказательство. Предположим B разрешимый, то есть есть $\text{inB}(x)$. Тогда A разрешается программой

```
1 inA(x):
2     return inB(f(x))
```

■

Пример (Задача останова). Пусть $HALT = \{p \mid p(\epsilon) \text{ не зависит} \}$.

Сведем U к $HALT$.

```
1 f(<p,x>):
2     return "q(y):
3         def p = ...
4         if p(x) != 1:
5             while True:
6                 pass
7     "
```

Заметим, что f всюду определенная и вычислимая.

f является m сведением U к $HALT$.

$q \in HALT \iff \langle p, x \rangle \in U$.

Разминка перед Т. У-Р. Пусть мы хотим проанализировать поведение программы.

$A = \{p \mid p \text{ чему-то удовлетворяет, но не зависит} \}$. Тогда A не разрешим.

Доказательство. Также напомним

```
1 f(<p,x>):
2     return "q(y):
3         if p(x) = 1:
4             сделай то, что удовлетворяет A
5         else: while True: pass
6     "
```

■

2.3 Теорема Успенского-Райса

E = множества перечислимых языков.

с: $\Sigma \equiv \text{char}$, $s: \Sigma^* \equiv \text{string}$, $L \subset \Sigma^* \equiv \text{set}\langle \text{string} \rangle = \text{lang}$, $2^{\Sigma^*} \equiv \text{set}\langle \text{lang} \rangle$, $E \subset 2^{\Sigma^*} \equiv \text{set}\langle \text{lang} \rangle$.

Свойство перечислимых языков:

$A \subset E \equiv \text{set}\langle \text{lang} \rangle$

$L \in A \rightarrow L$ удовлетворяет A

$L \notin A \rightarrow L$ не удовлетворяет A .

$L(A) = \{p \mid L(p) \in A\}$

$L(p) = \{x \mid p(x) = 1\}$.

Теорема 2.3.1 (Успенского-Райса). Язык любого нетривиального свойства перечислимых языков не разрешим.

$A = \emptyset \quad L(A) = \emptyset, \quad A = E \quad L(A) = \Sigma^*$. Есть $A \neq \emptyset, A \neq E \implies L(A)$ не разрешим.

Доказательство. Пусть $\emptyset \notin A$. Пусть какой-то язык $X \in A$.

$L(A)$ — разрешим $\text{inA}(p)$. Напишем следующий код, разрешающий U .

```

1 in U(⟨p, x⟩):
2     q = "q(y):
3         if p(x) = 1:
4             return inX(y)
5         else:
6             return 0
7     "
8     return inA(q)

```

Утверждается, что получился разрешитель для U . Если $p(x) = 1$, то $L(q) = X \in A$. А если $p(x) = 0$ или $p(x)$ зависит, то $L(q) \emptyset$.

Таким образом, $\langle p, x \rangle \in U \iff q \in L(A)$.

То есть f m -сводит U к $L(A)$. ■

2.4 Теорема о рекурсии

Бывают программы, что

```

1 q - разрешитель
2 f(x): ...
3     if q(f):
4         while True: pass
5     else:
6         return 1

```

Пример (игрушечный). Что выведет эта инструкция?

Написать 2 раза, второй раз в кавычках:

"Написать 2 раза, второй раз в кавычках:"

Действительно, получится:

Написать 2 раза, второй раз в кавычках:

"Написать 2 раза, второй раз в кавычках:"

То есть в каком-то месте программы мы можем вывести исходный код программы.

```

1 f(...):
2 .
3     getSource()
4 .
5 .

```

Программа выведет:

```

.
getSource():
.
.
.
.

```

Наиболее интересное применение — quine — программа, которая выводит свой исходный код.

Как же это делать?

```

1 f(...):
2 .
3 .
4     getSource()
5 .
6 .
7 getSource():
8     s ← getAuxSource()
9     return s + "getAuxSource():" +
10         "    return \"" + s + "\""
11
12 getAuxSource():
13     return "f(...):

```

```

14     ...
15     getSource():
16         s ← getAuxSource()
17         return s + "\"getSource():\" +
18             \"    return \\\"\\\"\" + s + \"\\\"\\\"\"
19     \"

```

Замечание. Заметим, что весь код программы состоит из того, что выведет `getAuxSource`, плюс определение `getSource`.

Теорема 2.4.1 (О рекурсии). Пусть $V(x, y)$ — вычислимая функция $\implies \exists$ вычислимая функция $f : f(y) = V(f, y)$.

Благодаря этой теореме можно проще доказывать неразрешимость некоторых языков.

Пример. $\text{HALT} = \{p \mid p(\varepsilon) \neq \perp\}$ неразрешим

Доказательство. Пусть h — разрешитель HALT .

```

1 p():
2     if h(p) == 1 :
3         while True: pass
4     else:
5         return 1

```

■

Пример (Теорема Успенского–Райса). $A \subseteq RE$, $A \neq \emptyset$, $A \neq RE$.
 $L(A) = \{p \mid L(p) \in A\}$ — неразрешим.

Доказательство. Пусть $\text{in}L_A$ — разрешитель.

Пусть $\underbrace{M}_{\text{in}M(x)} \in A$, $\underbrace{N}_{\text{in}N(x)} \notin A$.

```

1 f(x):
2     if (inL_A(getSource())):
3         return inN(x)
4     else:
5         return inM(x)

```

■

Замечание. Такая подростковая программа: послушаем родителей и сделаем наоборот.

Еще примеры невычислимых функций

Пример. $K(x)$ — Колмогоровская сложность

$K(x) = \min$ длина программы p , что $p() = x$.

К сожалению, нельзя написать программу, которая бы оптимально кодировала строку.

Утверждение 2.4.1. $K(x)$ — невычислима

Доказательство. Пусть $K(x)$ вычислима, напомним код.

```

1 p():
2     for x \in \Sigma*:
3         if k(x) > |getSource()|:
4             return x

```

Мы нашли программу, у которой колмогоровская сложность больше, чем сложность программы p , которая без проблем выводит эту строку. ■

Пример (Busy Beaver ($\text{BB}(n)$)). Эта функция принимает число n и возвращает максимальное число шагов, которое делают программы длины n (из n строчек или символов).

Утверждение 2.4.2. $BB(n)$ невычислима. Напишем программу, которая будет работать дольше.

```
1 f():  
2   for i = 0 .. BB(|getSource()|):  
3     pass  
4   return
```