Третий курс, осенний семестр 2017/18
Конспект лекций по алгоритмам

Собрано 16 сентября 2017 г. в 21:07

Содержание

1. FFT и его друзья **1**
- 1.1. FFT . . . 1
  - 1.1.1. Прелюдия . . . 1
  - 1.1.2. Собственно идея FFT . . . 1
  - 1.1.3. Крутая реализация FFT . . . 1
  - 1.1.4. Обратное преобразование . . . 2
  - 1.1.5. Два в одном . . . 3
  - 1.1.6. Умножение чисел, оценка погрешности . . . 3
- 1.2. Разделяй и властвуй . . . 3
  - 1.2.1. Перевод между системами счисления . . . 3
  - 1.2.2. Деление многочленов с остатком . . . 3
  - 1.2.3. Вычисление значений в произвольных точках . . . 4
  - 1.2.4. Интерполяция . . . 4
  - 1.2.5. Извлечение корня . . . 4
- 1.3. Литература . . . 5

2. Деление многочленов **6**
- 2.1. Быстрое деление многочленов . . . 6
- 2.2. Быстрое деление чисел . . . 6
- 2.3. Быстрое извлечение корня для чисел . . . 6
- 2.4. Обоснование метода Ньютона . . . 7
- 2.5. Линейные рекуррентые соотношения . . . 7
  - 2.5.1. Матрица в степени . . . 7
  - 2.5.2. Деление многочленов . . . 7

2. Автоматы **7**
- 2.1. Определения, изоморфизм . . . 8
- 2.2. Минимальность, эквивалентность . . . 8
- 2.3. Алгоритм Хопкрофта . . . 8

3. Суффиксный автомат **9**
- 3.1. Введение, основные леммы . . . 9
- 3.2. Алгоритм построения за линейное время . . . 10
- 3.3. Реализация . . . 10
- 3.4. Линейность размера автомата, линейность времени построения . . . 10

Лекция #1: FFT и его друзья

5 сентября

1.1. FFT

1.1.1. Прелюдия

Пусть есть многочлены $A(x) = \sum a_i x^i$ и $B(x) = \sum b_i x^i$.
Посчитаем их значения в точках $x_1, x_2, \ldots, x_n$: $A(x_i) = fa_i, B(x_i) = fb_i$.
Значения $C(x) = A(x)B(x)$ в точках x_i можно получить за линейное время:

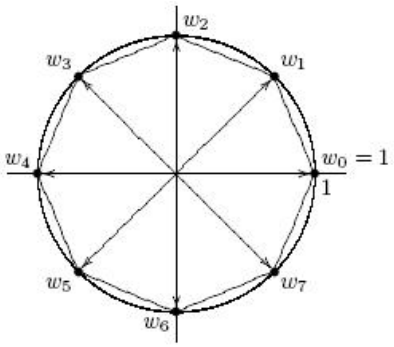
$$fc_i = C(x_i) = A(x_i)B(x_i) = fa_i fb_i$$

Схема быстрого умножения многочленов:

$$a_i, b_i \overset{\mathcal{O}(n\log n)}{\longrightarrow} fa_i, fb_i \overset{\mathcal{O}(n)}{\longrightarrow} fc_i = fa_i fb_i \overset{\mathcal{O}(n\log n)}{\longrightarrow} c_i$$

Осталось подобрать правильные точки x_i.

FFT расшифровывается Fast Fourier Transform и за $\mathcal{O}(n \log n)$ вычисляет значения многочлена в точках $w_j = e^{\frac{2\pi i j}{n}}$ для $n = 2^k$ (то есть, только для степеней двойки).



1.1.2. Собственно идея FFT

$A(x) = \sum a_i x^i = (a_0 + x^2 a_2 + x^4 a_4 + \ldots) + x(a_1 x + a_3 x^3 + a_5 x^5 + \ldots) = B(x^2) + xC(x^2)$ – обозначили все чётные коэффициенты A многочленом B, а нечётные соответственно C.

Заметим, что если все $w_j^2 = w_{n/2+j}^2$, поэтому многочлены B и C нужно считать уже не в n, а в $\frac{n}{2}$ точках. Итого алгоритм:

```
1 def FFT(a):
2   n = len(a)
3   if n == 1: return a[0] # посчитать значение A(x) = a[0] в точке 1
4   a ---> b, c
5   b, c = FFT(b), FFT(c)
6   w = exp(2*pi*i/n)
7   for i=0..n-1: a[i] = b[i % (n/2)] + w^i * c[i % (n/2)]
8   return a
```

Время работы $T(n) = 2T(n/2) + \mathcal{O}(n) = \mathcal{O}(n \log n)$.

1.1.3. Крутая реализация FFT

Чтобы преобразование работало быстро, нужно заранее предподсчитать все $w_j = e^{\frac{2\pi i j}{n}}$.
Заметим, что b и c можно хранить прямо в массиве a. Тогда получается, что на прямом ходу рекурсии мы просто переставляем местами элементы a, а уже на обратном делаем какие-то по-

лезные действия. Число a_i перейдёт на позицию $a_{r(i)}$, где $rev(i)$ – перевёрнутая битовая запись i. Кстати, $rev(i)$ мы уже умеем считать динамикой для всех i.
При реализации на C++ можно использовать стандартные комплексные числа `complex<double>`.

```
const int K = 20, N = 1 << K;
complex<double> root[N];
int rev[N];
void init() {
  for (int j = 0; j < N; j++) {
    root[j] = exp(2π*i*j/N); // cos(2πj/N), sin(2πj/N)
    rev[j] = rev[j >> 1] + ((j & 1) << (K - 1));
  }
}
```

Теперь, корни из 1 степени k хранятся в `root[j*N/k]`, $j \in [0, k)$. Доступ к памяти при этом не последовательный, проблемы с кешом. Чтобы корни, мы $2N$ раз вычисляли тригонометрические функции. Можно лучше (версия #2):

```
  for (int k = 1; k < N; k *= 2) {
    num tmp = exp(π/k);
    root[k] = {1, 0}; // в root[k..2k) хранятся первые k корней степени 2k
    for (int i = 1; i < k; i++)
      root[k + i] = (i & 1) ? root[(k + i) >> 1] * tmp : root[(k + i) >> 1];
  }
```

Теперь код собственно преобразования Фурье может выглядеть так:

```
void FFT(a, fa) { // a --> fa
  for (int i = 0; i < N; i++)
    fa[rev[i]] = a[i];
  for (int k = 1; k < N; k *= 2) // уже посчитаны FFT от кусков длины k
    for (int i = 0; i < N; i += 2 * k) // [i..i+k) [i+k..i+2k) --> [i..i+2k)
      for (int j = 0; j < k; j++) { // оптимально написанный стандартный цикл FFT
        num tmp = root[k + j] * fa[i + j + k]; // вторая версия root[]
        fa[i + j + k] = fa[i + j] - tmp;
        fa[i + j] = fa[i + j] + tmp;
      }
}
```

1.1.4. Обратное преобразование

Теперь имея при $w = e^{2\pi i/n}$:
$fa_0 = a_0 + a_1 + a_2 + a_3 + \dots$
$fa_1 = a_0 + a_1 w + a_2 w^2 + a_3 w^3 + \dots$
$fa_2 = a_0 + a_1 w^2 + a_2 w^4 + a_3 w^3 + \dots$
$\dots$
Нам нужно научиться восстанавливать коэффициенты $a_0, a_1, a_2, \dots$
Заметим, что $\forall j \neq 0 \ \sum k = 0^{n-1} w^{jk} = 0$ (сумма геометрической прогрессии).
И напротив при $j = 0$ получаем $\sum k = 0^{n-1} w^{jk} = n$.

Поэтому $fa_0 + fa_1 + fa_2 + \dots = a_0 n + a_1 \sum_k w^k + a_2 \sum_k w^{2k} + \dots = a_0 n$
Аналогично $fa_0 + fa_1 w^{-1} + fa_2 w^{-2} + \dots = \sum_k a_0 w^{-k} + a_1 n + a_2 \sum_k w^k + \dots = a_1 n$
И в общем случае $\sum_k fa_k w^{-jk} = a_j n$.
Заметим, что это ровно значение многочлена с коэффициентами fa_k в точке w^{-j}.

Осталось заметить, что множества чисел $\{w_j | j = 0..n-1\}$ и $\{w_{-j} | j = 0..n-1\}$ совпадают $\Rightarrow$

```
1 void FFT_inverse(fa, a) { // fa --> f
2   FFT(a, fa)
3   reverse(fa + 1, fa + N) // w^j <--> w^{-j}
4   for (int i = 0; i < N; i++) fa[i] /= N;
5 }
```

1.1.5. Два в одном

Часто коэффициенты многочленов – вещественные числа.
Если у нас есть многочлены $A(x), B(x) \in \mathbb{R}[x]$, возьмём числа $c_j = a_j + ib_j$ и посчитаем $fc = FFT(c)$. Тогда по fc за $\mathcal{O}(n)$ можно легко восстановить fa и fb.

Для этого вспомним про сопряжения комплексных чисел:
$\overline{x + iy} = x - iy, \overline{a \cdot b} = \overline{a} \cdot \overline{b}, w^{n-j} = w^{-j} = \overline{w^j} \Rightarrow \overline{fc_{n-j}} = \overline{C(w^{n-j})} = \overline{C}(w^j) \Rightarrow$
$fc_j + \overline{fc_{n-j}} = 2A(w^j) = 2fa_j$. Аналогично $fc_j - \overline{fc_{n-j}} = 2B(w^j) = 2fb_j$.

Итого для умножения двух многочленов можно использовать не 3 преобразования Фурье, а 2.

1.1.6. Умножение чисел, оценка погрешности

Число длины n. в системе счисления 10 можно за $\mathcal{O}(n)$ представить как число в системе счисления 10^k, а его как многочлен длины n/k. Умножения многочленов такой длины будет работать за $\frac{n}{k} \log \frac{n}{k}$. Отсюда возникает вопрос, какое максимальное k можно использовать? Коэффициенты многочлена-произведения будут целыми числами до $(10^k)^2 \frac{n}{k}$. Чтобы типе `double` целое число хранилось с погрешностью меньше 0.5 (тогда мы его сможем правильно округлить к целому), оно должно быть не более 10^{15}. Получаем при $n \leqslant 10^6$, что $(10^k)^2 10^6/k \leqslant 10^{15} \Rightarrow k \leqslant 4$. Аналогично для типа `long double` имеем $(10^k)^2 10^6/k \leqslant 10^{18} \Rightarrow k \leqslant 6$. Это оценка сверху, предполагающая, что само FFT погрешность не накапливает... на самом деле эта оценка очень близка к точной.

1.2. Разделяй и властвуй

1.2.1. Перевод между системами счисления

Чтобы перевести число X длины $n = 2^k$ в системе счисления с основанием a в систему счисления с основанием b, разобьём X на $n/2$ старших цифр – X_0 и $n/2$ младших цифр – X_1. Тогда $F(X) = F(X_0)F(a^{n/2}) + F(X_1)$, где $F(X)$ – число X в системе счисления b, а умножения и сложения выполняются в системе счисления b. Сложение $= \mathcal{O}(n)$, умножение $= \mathcal{O}(n \log n)$.
Предподсчёт $F(a^1), F(a^2), F(a^4), F(a^8), \ldots, F(a^n)$ займёт $\left(\sum_k \mathcal{O}(2^k k)\right) = \mathcal{O}(n \log n)$ времени.

Два рекурсивных вызова от задач размера $n/2$, итого $T(n) = 2T(n/2) + \mathcal{O}(n \log n) = \mathcal{O}(n \log^2 n)$.

1.2.2. Деление многочленов с остатком

Задача: даны $A(x), B(x) \in \mathbb{R}[x]$, найти $Q(x), R(x)$: $\deg R < \deg B \wedge A(x) = B(x)Q(x) + R(x)$.
Зная Q мы легко найдём R, как $A(x) - B(x)Q(x)$ за $\mathcal{O}(n \log n)$. Сосредоточимся на поиске Q.
Пусть $\deg A = \deg B = n$, тогда $Q(x) = \frac{a_n}{b_n}$. То есть, $Q(x)$ можно найти за $\mathcal{O}(1)$.
Из этого мы делаем вывод, что Q зависит не обязательно от всех коэффициентов A и B.

Lm 1.2.1. $\deg A = m, \deg B = n \Rightarrow \deg Q = m - n$, и Q зависит только от $m{-}n{+}1$ коэффициентов A и $m{-}n{+}1$ коэффициентов B.

Доказательство. У A и $Z = B \cdot Q$ должны совпадать $m{-}n{+}1$ старший коэффициент ($\deg R < n$). В этом сравнении участвуют только $m{-}n{+}1$ старших коэффициентов A. При домножение B на $x^{\deg Q}$, сравнятся как раз $m{-}n{+}1$ старших коэффициентов A и B. При домножении B на меньшие степени x, в сравнении будут участвовать лишь какие-то первые из этих $m{-}n{+}1$ коэффициентов. ■

Теперь будем решать задачу: даны n старших коэффициентов A и B, найти такой C из n коэффициентов, что у A и BC совпадает n старших коэффициентов. Давайте считать, что младшие коэффициенты лежат в первых ячейках массива.

```
1 void Div( int n, int *A, int *B)
2   C = Div(n/2, A + n/2, B + n/2) // нашли старших n/2 коэффициентов ответа
3   A' = Subtract(n, A, n + n/2 - 1, Multiply(C, B))
4   D = Div(n/2, A', B + n/2) // сейчас A' состоит из n/2 не нулей и n/2 нулей
5   return concatenate(D, C) // склеили массивы коэффициентов
```

Здесь `Subtract` – хитрая функция. Она знает длины многочленов, которые ей передали, и сдвигает вычитаемый многочлен так, чтобы старшие коэффициенты совместились.

1.2.3. Вычисление значений в произвольных точках

Дан многочлен $A(x), \deg A = n$ и точки $x_1, x_2, \dots x_n$. Мы хотим найти $A(x_1), A(x_2), \dots A(x_n)$. Заметим, что $A(x_i) = (A(x) \bmod \prod_{j=1..n}(x - x_j))(x_i)$. За $\mathcal{O}(1)$ делений с остатком можно разделить задачу на две независимых: посчитать значения $B(x) = (A(x) \bmod \prod_{j=1..n/2}(x - x_j))$ в точках $x_1, \dots, x_{n/2}$ и посчитать значения $C(x) = (A(x) \bmod \prod_{j=n/2+1..n}(x - x_j))$ в точках $x_{n/2+1}, \dots, x_n$.

Итого $T(n) = 2T(n/2) + \mathcal{O}(\texttt{div}(n)) = \mathcal{O}(\texttt{div}(n) \log n)$.

1.2.4. Интерполяция

Даны пары $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$. Задача – найти многочлен A степени $n - 1$, $\forall i$ дающий в точке x_i значение y_i. Будем решать задачу интерполяции по Ньютону методом разделяй и властвуй. Сперва найдём интерполяционный многочлен B для $(x_1, y_1), (x_2, y_2), \dots, (x_{n/2}, y_{n/2})$. $A = B + C \cdot \prod_{j=1..n/2}(x - x_j) = B + C \cdot D$. Чтобы у A были правильные значения в точках $x_{n/2+1}, \dots, x_n$, вычислим значения B и D в точках $x_{n/2+1}, \dots, x_n$ – b_j и d_j, и возьмём C, как интерполяционный многочлен от точек $(x_j, -b_j/d_j)$ при $j = n/2{+}1..n$.
Время работы $T(n) = 2T(n/2) + 2\mathcal{O}(\texttt{calcValue}(n/2)) = \mathcal{O}(\texttt{calcValue}(n) \log n)$.

1.2.5. Извлечение корня

Дан многочлен $A(x)$: $\deg A \equiv 0 \bmod 2$. Задача – найти $R(x)$: $\deg(A - R^2)$ минимальна.
Пусть мы уже нашли старшие k коэффициентов R, обозначим их R_k. Найдём $2k$ коэфф-тов: $R_{2k} = R_k x^k + X, R_{2k}^2 = R_k^2 x^{2k} + 2R_k X \cdot x^k + X^2$. Правильно подобрав X, мы можем "обнулить" k коэффициентов $A - R_{2k}^2$, для этого возьмём $X = (A - R_k^2 x^{2k})/(2R_k)$. В этом частном нам интересны только k старших коэффициентов, поэтому переход от R_k к R_{2k} происходит за $\mathcal{O}(\texttt{mul}(k) + \texttt{div}(k))$. Итого суммарное время на извлечение корня – $\mathcal{O}(\texttt{div}(n))$.

1.3. Литература

1. Слайды от sankowski по FFT и всем идеям разделяй и властвуй.
2. e-maxx про FFT и оптимизации к нему.
3. Задачи с codeforces на тему FFT.
4. Краткий конспект похожих идей в vk.

Лекция #2: Деление многочленов

12 сентября

2.1. Быстрое деление многочленов

Цель – научиться делить многочлены за $\mathcal{O}(n \log n)$.
Очень хочется считать частное многочленов $A(x)/B(x)$, как $A(x)B^{-1}(x)$. К сожалению, у многочленов нет обратных. Зато обратные есть у рядов, научимся сперва искать их.

• Обращение ряда

Задача. Дан ряд $A \in [[\mathbb{R}]], a_0 \neq 0$. Найти ряд B: $A(x)B(x) = 1 + x^n C(x)$.
Можно это сделать за $\mathcal{O}(n^2)$:
$b_0 = 1/a_0$
$b_1 = -(a_1 b_0)/a_0$
$b_2 = -(a_2 b_0 + a_1 b_1)/a_0$
...
А можно за $\mathcal{O}(n \log n)$. Для этого достаточно, зная $B_k(x)$: $A(x)B_k(x) = 1 + x^k C_k(x)$,
научиться за $\mathcal{O}(k \log k)$ вычислять $B_{2k}(x)$: $A(x)B_{2k}(x) = 1 + x^{2k} C_{2k}(x)$.
$B_{2k} = B_k + x^k Z \Rightarrow A \cdot B_{2k} = 1 + x^k C_k + x^k A \cdot Z = 1 + x^k (C_k - A \cdot Z)$.
Выберем $Z = B_k \cdot C_k \Rightarrow C_k - A \cdot Z = C_k - (1 + x^k C_k) C_k = x^k C_{2k}$.
Итого $B_{2k} = B_k + B_k(1 - A \cdot B_k) = B_k(2 - A \cdot B_k)$. Вычисляется за 2 умножения $= \mathcal{O}(k \log k)$.
Конечно, мы обрежем B_{2k}, оставив лишь $2k$ первых членов.

• Деление многочленов

Теперь, когда мы умеем обращать ряды, как это применить к делению многочленов?
Делим $A(x)$ степени m на $B(x)$ степени n. Перевёрнутый многочлен обозначим $A^r(x)$.
Заметим, что $(PQ)^r = P^r Q^r$. Поделим A на B с остатком: $A = BQ + R$.
Заметим, что у A и BQ совпадают $(m+1) - n$ старших коэффициентов $\Rightarrow$
$A^r - B^r Q^r \equiv 0 \bmod x^{m+1-n}$
Обозначим первые $m + 1 - n$ членов обратного ряда $(B^r)^{-1}$, как D.
Возьмём $Q^r = D \cdot A^r \Rightarrow B^r Q^r = (B^r D) A^r = (1 + x^{m+1-n} C) A^r \equiv A^r \bmod x^{m+1-n}$

2.2. Быстрое деление чисел

Для нахождение частного чисел, достаточно научиться с большой точностью считать обратно.
Рассмотрим метод Ньютона поиска корня функции $f(x)$:
$x_0 =$ `достаточно точное приближение корня`
$x_{i+1} = x_i - f(x_i)/f'(x_i)$
Решим с помощью него уравнение $f(x) = x^{-1} - a = 0$.
$x_0 =$ `обратное к старшей цифре` a
$x_{i+1} = x_i - (\frac{1}{x_i} - a)/(-\frac{1}{x_i^2}) = x_i + (x_i - a \cdot x_i^2) = x_i(2 - a x_i)$.
Любопытно, что очень похожую формулу мы видели при обращении формального ряда...
Утверждение: каждый шаг метода Ньютона удваивает число точных знаков x.
Итого, имея x_i с k точными знаками, мы научились за $\mathcal{O}(k \log k)$ получать x_{i+1} с $2k$ точными знаками. Суммарное время получения n точных знаков – $\mathcal{O}(n \log n)$.

2.3. Быстрое извлечение корня для чисел

Продолжаем пользоваться методом Ньютона.
$x_{i+1} = \frac{1}{2}(x_i + \frac{a}{x_i})$

2.4. Обоснование метода Ньютона

Цель: доказать, что каждый шаг удваивает число точных знаков x.

Сдалем замену переменных, чтобы было верно $f(0) = 0 \Rightarrow$ корень, который мы ищем, – 0.
Сейчас находимся в точке x_i. По Тейлору $f(0) = f(x_i) - x_i f'(x_i) + x_i^2 f''(\alpha)$ $(\alpha \in [0..x_i])$.
Получаем $\frac{f(x_i)}{f'(x_i)} = x_i + x_i^2 \frac{f''(\alpha)}{f'(x_i)}$. Передаём Ньютону $x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} = x_i - x_i - x_i^2 \frac{f''(\alpha)}{f'(x_i)}$.
Величина $\frac{f''(\alpha)}{f'(x_i)}$ ограничена сверху константой C.
Получаем, что если $x_i \leqslant 2^{-n}$, то $x_{i+1} \leqslant 2^{-2n+\log C}$.
То есть, число верных знаков почти удваивается.

2.5. Линейные рекуррентные соотношения

2.5.1. Матрица в степени

TODO

2.5.2. Деление многочленов

TODO

Лекция #2: Автоматы

12 сентября

2.1. Определения, изоморфизм

TODO

2.2. Минимальность, эквивалентность

TODO

2.3. Алгоритм Хопкрофта

TODO

Лекция #3: Суффиксный автомат

19 сентября

3.1. Введение, основные леммы

Def 3.1.1. *Суффиксный автомат (суффавтомат) строки s, $SA(s)$* – min *по числу вершин ДКА A*:

$$L(A) = \{\textit{суффиксы } s\}$$

Def 3.1.2. $R_s(u)$ – *правый контекст строки u относительно строки s.*

$$R_s(u) = \{x \mid ux \text{ – суффикс } s\}$$

Пример: $s = abacababa \Rightarrow R_s(ba) = \{cababa, ba, \epsilon\}$
Мы будем рассматривать правые контексты только от подстрок $s \Rightarrow R_s(v) \neq \varnothing$.

Def 3.1.3. $V_A = \{u \mid R_s(u) = A\}$ – *все строки с правым контекстом A.*

Замечание 3.1.4. Каждому классу V_A соответствует ровно одна вершина суффавтомата.

Следствие 3.1.5. Каждому правому контексту A соответствует ровно одна вершина суффавтомата. В дальнейшем мы будем отождествлять правые контексты и вершины автомата.
V_A – множество строк, заканчивающихся в вершине автомата A.

Def 3.1.6. $A(w)$ – *вершина автомата: $w \in V_A$ (вершина по строке).*

Следствие 3.1.7. Рёбра между вершинами проводятся однозначно:
$(\exists\, x, c\colon x \in V_A, xc \in V_B) \Leftrightarrow$ (между вершинами A и B есть ребро по символу "c").

Lm 3.1.8. $R_s(v) = R_s(u), |v| \leqslant |u| \Rightarrow v$ – суффикс u.

Lm 3.1.9. v – суффикс $u \Rightarrow R_s(u) \subseteq R_s(v)$ (у суффикса правый контекст шире).

Следствие 3.1.10. $\forall$ строки w несколько самых больших суффиксов w имеют такой же, как w, правый контекст, затем идёт суффикс со строго меньшим правым контекстом.

Def 3.1.11. `str[A]` *(строка вершины A) – максимальная по длине строка из V_A.*

Def 3.1.12. `len[A]` *(длина A) =* `|str[A]|`.

Def 3.1.13. `suf[A]` *(суффиксная ссылка A) – вершина автомата, соответствующая самому длинному суффиксу* `str[A]` *с отличным от A правым контекстом.*

Замечание 3.1.14. `suf[A]` корректно определена iff `len[A]` $\neq 0$.

Lm 3.1.15. Для суффавтомата строки s терминальными являются
вершины `A(s)`, `suf[A(s)]`, `suf[suf[A(s)]]`, ...

Lm 3.1.16. $\min_{w \in V_A} |w| =$ `len[suf[A]]` $+ 1$

3.2. Алгоритм построения за линейное время

Алгоритм будет онлайн наращивать строку s. Начинаем с пустой строки s. Осталось научиться, дописывая к s символ a, от $SA(s)$ переходить к $SA(sa)$.

Будем в каждый момент времени поддерживать:

(a) `start` – A(ϵ) (стартовая вершина)
(b) `last` – A(s) (последняя вершина)
(c) `suf[A]` – для каждой вершины автомата суффиксную ссылку (суфффссылку)
(d) `len[A]` – для каждой вершины автомата максимальную длину строки
(e) `next[A,c]` – рёбра автомата

База: $s = \epsilon, \texttt{start} = \texttt{last} = 1$.

Для того, чтобы понять, как меняется автомат, нужно понять, как меняются его вершины – правые контексты. Переход: $s \to sa \Rightarrow R_s(v) = \{z_1, \dots, z_k\} \to R_{sa}(v) = \{z_1 a, \dots, z_k a\}$ `+?` ϵ.

Lm 3.2.1. $(\epsilon \in R_{sa} v) \Leftrightarrow (v$ – суффикс $sa)$.

TODO

3.3. Реализация

TODO

3.4. Линейность размера автомата, линейность времени построения

TODO