СПб ВШЭ, 3-й курс, 3-й модуль 2023/24
Конспект лекций по алгоритмам

Собрано 23 января 2024 г. в 00:09

Содержание

Лекция #1: Действия над многочленами

8 апреля 2019

1.1. Над $\mathbb{F}_2$

Умножение/деление/gcd можно делать битовым сжатием за $\approx \frac{nm}{w}$, где w – word size.

• Хранение

$A(x) = a_0 + a_1 x + \cdots + a_n x^n \ \rightarrow$ `N = n + 1; bitset<N> a`
Обобзначения: $n = \deg A$, $m = \deg B$, $N = \deg A + 1$, $M = \deg B + 1$.
Многочлен степени n = массив коэффициентов длины N.

• Умножение

```
1 for i=0..n
2     if a[i]
3         c ^= b << i
```

Время работы: $\mathcal{O}(n \cdot \lceil \frac{m}{w} \rceil)$. Можно n заменить на «*число не нулей в* `a`».

• Деление

```
1 for i=n..m
2     if a[i]
3         a ^= b << (i-m), c[i-m] = 1
```

Результат: в «a» лежит остаток, в «c» частное.
Время работы: $\mathcal{O}((n-m) \cdot \lceil \frac{m}{w} \rceil)$.

• gcd

Запускаем Евклида. Один шаг Евклида – деление. Деление работает за $\mathcal{O}((n-m) \cdot \lceil \frac{m}{w} \rceil)$ и уменьшает n на $n-m \Rightarrow$ суммарно все деления отработают за $\mathcal{O}((n-m) \cdot \lceil \frac{m}{w} \rceil + \frac{m^2}{w}) = \mathcal{O}(\frac{nm}{w})$.

1.2. Умножение многочленов

Над произвольным кольцом умеем за $\mathcal{O}(nm)$.
Точнее за «(*число не нулей в* `a`)$\cdot$(*число не нулей в* `b`)».

• Карацуба

Пусть $N = 2^k$. Если нет, дополним оба массива нулями (нулевые старшие коэффициенты).
Делим многочлены на две части: $A = A_0 + x^{N/2} A_1,\ B = B_0 + x^{N/2} B_1$

$$A \cdot B = A_0 B_0 + x^N A_1 B_1 + x^{N/2}(A_0 B_1 + A_1 B_0) = C_0 + x^N C_1 + x^{N/2}((A_0+B_0)(A_1+B_1) - C_0 - C_1)$$

Умножение многочленов длины N – сложение, вычитание и 3 умножения многочленов длины $\frac{N}{2}$.

$$T(n) = 3T(\tfrac{n}{2}) + n = \boxed{\Theta(n^{\log 3})}$$

Такой способ умножения работает над произвольным кольцом.

• Оптимальное над $\mathbb{F}_2$

У нас уже есть Карацуба и битовое сжатие. Соединим. Внутри Карацубы реализуем сложение, вычитание и разделение многочлена на две части за $\lceil \frac{n}{w} \rceil$. Казалось бы мы ускорили всё в w раз, но нет, время работы равно числу листьев рекурсии.

$$T(n) = 3T(\tfrac{n}{2}) + \lceil \tfrac{n}{w} \rceil = \Theta(n^{\log 3})$$

Асимптотическая оптимизация: в рекурсии при $N \leqslant w$ будем умножать за $\mathcal{O}(n)$. Улучшили $w^{\log 3}$ до $w \Rightarrow$ новое время работы в $w^{(\log 3)-1}$ раз меньше.

• **Над $\mathbb{Z}$, над $\mathbb{R}$, над $\mathbb{C}$**

Фурье за $\mathcal{O}(n \log n)$. Смотри главу про Фурье (FFT).

• **Над конечным полем**

Все конечные поля изоморфны $\Rightarrow$ умножить над $\mathbb{F}_p \Leftrightarrow$ умножить над $\mathbb{Z}/p\mathbb{Z}$.
Умножим в $\mathbb{Z}$ (Карацуба или FFT), затем возьмём по модулю.
Для некоторых p, например $p = 3 \cdot 2^{18} + 1$, можно напрямую применить «Фурье по модулю».

Лекция #2: FFT и его друзья

5 сентября

2.1. FFT

2.1.1. Прелюдия

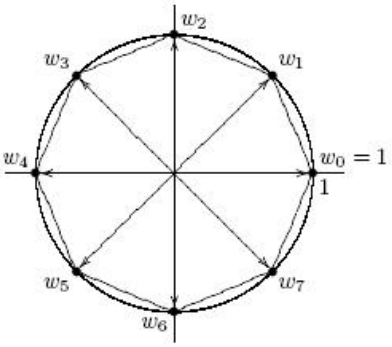
Пусть есть многочлены $A(x) = \sum a_i x^i$ и $B(x) = \sum b_i x^i$.
Посчитаем их значения в точках $x_1, x_2, \ldots, x_n$: $A(x_i) = fa_i, B(x_i) = fb_i$.
Значения $C(x) = A(x)B(x)$ в точках x_i можно получить за линейное время:

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

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 комплексных точках $w_j = e^{\frac{2\pi i j}{n}}$ для $n = 2^k$ (то есть, только для степеней двойки).

Что нужно помнить про комплексные числа?
При умножении комплексных чисел углы складываются, длины перемножаются.
В частности, если обозначить $w = e^{\frac{2\pi i}{n}} = \cos\frac{2\pi i}{n} + i\sin\frac{2\pi i}{n}$, то $w_j = w^j$ (все корни из единицы – это степени главного корня, и они образуют циклическую группу). Также $w^{-j} = w^{n-j}$.

2.1.2. Собственно 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 многочлена A многочленом B, а нечётные соответственно C.

Посчитаем рекурсивно $B(w_j)$ и $C(w_j)$, зная их, за $\mathcal{O}(n)$ посчитаем $A(w_j) = B(w_j) + w_j C(w_j)$.

Заметим, что $\forall j\ w_j = w_{j \bmod n} \Rightarrow \forall j\ w_j^2 = w_{n/2+j}^2 \Rightarrow B$ и C нужно считать только в $\frac{n}{2}$ точках.

Итого алгоритм:

```
1 def FFT(a):
2     n = len(a)
3     if n == 1: return a[0] # посчитать значение многочлена A(x) ≡ a[0] в точке x = 1
4     for j=0..n-1: (j%2 ? c : b)[j / 2] = a[j]
5     b, c = FFT(b), FFT(c) # самое важное - две ветки рекурсии
6     for j=0..n-1: a[j] = b[j % (n/2)] + exp(2πi*j/n) * c[j % (n/2)]
7     return a
```

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

2.1.3. Крутая нерекурсивная реализация FFT

Чтобы преобразование работало быстро, нужно заранее предподсчитать все $w_j = e^{\frac{2\pi ij}{n}}$.

Заметим, что b и c можно хранить прямо в массиве a. Тогда получается, что на прямом ходу рекурсии мы просто переставляем местами элементы a, на обратном ходу рекурсии делаем какие-то полезные действия. Число a_i перейдёт на позицию $a_{rev(i)}$, 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>`, но свои рукописные будут работать немного быстрее.

```
const int K = 20, N = 1 << K; // N - ограничение на длину результата умножения многочленов
complex<double> root[N];
int rev[N];
void init():
    for (int j = 0; j < N; j++):
        root[j] = exp(2πi*j/N); // cos(2πj/N), sin(2πj/N)
        rev[j] = rev[j >> 1] + ((j & 1) << (K - 1));
```

Теперь, корни из единицы степени k хранятся в `root[j*N/k]`, $j \in [0, k)$.
Доступ к памяти при этом не последовательный, проблемы с кешом.
Чтобы посчитать все корни, мы $2N$ раз вычисляли тригонометрические функции.

- **Улучшенная версия вычисления корней**

```
for (int k = 1; k < N; k *= 2):
    num tmp = exp(πi/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:

```
FFT(a): // a → f = FFT(a)
    vector<complex> f(N);
    for (int i = 0; i < N; i++) // прямой ход рекурсии превратился в один for =)
        f[rev[i]] = a[i];
    for (int k = 1; k < N; k *= 2) // пусть 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 главный цикл FFT
                num tmp = root[k + j] * f[i + j + k]; // root[] из «улучшенной версии»
                f[i + j + k] = f[i + j] - tmp; // w_{j+k} = -w_j при n = 2k
                f[i + j] = f[i + j] + tmp;
    return f;
```

2.1.4. Обратное преобразование

Обозначим $w = e^{2\pi i/n}$. Нам нужно из
$f_0 = a_0 + a_1 + a_2 + a_3 + \dots$
$f_1 = a_0 + a_1 w + a_2 w^2 + a_3 w^3 + \dots$
$f_2 = a_0 + a_1 w^2 + a_2 w^4 + a_3 w^3 + \dots$

научиться восстанавливать коэффициенты $a_0, a_1, a_2, \dots$

Заметим, что $\forall j \neq 0 \sum\limits_{k=0}^{n-1} w^{jk} = 0$ (сумма геометрической прогрессии).
И напротив при $j = 0$ получаем $\sum\limits_{k=0}^{n-1} w^{jk} = \sum 1 = n$.
Поэтому $f_0 + f_1 + f_2 + \dots = a_0 n + a_1 \sum_k w^k + a_2 \sum_k w^{2k} + \dots = \boxed{a_0 n}$
Аналогично $f_0 + f_1 w^{-1} + f_2 w^{-2} + \dots = \sum_k a_0 w^{-k} + a_1 n + a_2 \sum_k w^k + \dots = \boxed{a_1 n}$
И в общем случае $\sum_k f_k w^{-jk} = \boxed{a_j n}$
Рассмотрим $F(x) = f_0 + x f_1, x^2 f_2 + \dots \Rightarrow F(w^{-j}) = a_j n$, похоже на `FFT(f)`.
Осталось заметить, что множества чисел $w^{-j} = w^{n-j} \Rightarrow$

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

2.1.5. Два в одном

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

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

Итого для умножения двух многочленов можно использовать не 3 вызова FFT, а 2.

2.1.6. Умножение чисел, оценка погрешности

Число длины n в системе счисления 10 $\rightarrow$ система счисления 10^k $\rightarrow$ 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 $(10^k)^2 \cdot \frac{n}{k}$.
Чтобы в 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 $(10^k)^2 \cdot 10^6 / k \leqslant 10^{15} \Rightarrow k \leqslant 4$.
Аналогично для типа `long double` имеем $(10^k)^2 \cdot 10^6 / k \leqslant 10^{18} \Rightarrow k \leqslant 6$.
Это оценка сверху, предполагающая, что само FFT погрешность не накапливает... на самом деле эта оценка очень близка к точной.

2.2. Разделяй и властвуй

2.2.1. Перевод между системами счисления

Задача: перевести число X длины $n = 2^k$ из a-ичной системы счисления в b-ичную.

Разобьём число X на $\frac{n}{2}$ старших цифр и $\frac{n}{2}$ младших цифр: $X = X_0 \cdot a^{n/2} + X_1 \Rightarrow$

$$F(X) = F(X_0)F(a^{n/2}) + F(X_1)$$

Умножение за $\mathcal{O}(n \log n)$ и сложение за $\mathcal{O}(n)$ выполняются в системе счисления b.
Предподсчёт $F(a^1), F(a^2), F(a^4), F(a^8), \ldots, F(a^n)$ займёт $\sum_k \mathcal{O}(2^k k) = \mathcal{O}(n \log n)$ времени.
Итого $T(n) = 2T(n/2) + \mathcal{O}(n \log n) = \mathcal{O}(n \log^2 n)$.

2.2.2. 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 2.2.1. $\deg A = m, \deg B = n \Rightarrow \deg Q = m - n$, и Q зависит только от $m{-}n{+}1$ коэффициентов A и $m{-}n{+}1$ коэффициентов B.

Доказательство. У A и $B \cdot Q$ 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 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.

2.2.3. Вычисление значений в произвольных точках

Задача. Дан многочлен $A(x)$, $\deg A = n$ и точки $x_1, x_2, \ldots x_n$. Найти $A(x_1), A(x_2), \ldots A(x_n)$.

Вспомним теорему Безу: $A(w) = A(x) \bmod (x - w)$.
Обобщение: $B(x) = A(x) \bmod (x - x_1)(x - x_2)\ldots(x - x_n) \ \Rightarrow \ \forall j \ B(x_j) = A(x_j)$

```
1 def Evaluate(n, A, x[]): # n = 2^k
2     if n == 1: return list(A[0])
3     return Evaluate(n/2, A mod (x-x_1)·...·(x-x_{n/2}), [x_1, ... , x_{n/2}]) +
4         Evaluate(n/2, A mod (x-x_{n/2+1})·...·(x-x_n), [x_{n/2+1}, ... , x_n])
```

Итого $T(n) = 2T(n/2)+\mathcal{O}(\texttt{div}(n))$. Если деление реализовано за $\mathcal{O}(n \log n)$, получим $\mathcal{O}(n \log^2 n)$.

2.2.4. Интерполяция

Задача. Даны пары $(x_1, y_1), \ldots, (x_n, y_n)$. Найти многочлен A: $\deg A = n-1$, $\forall i\ A(x_i) = y_i$.

Сделаем интерполяцию по Ньютону методом разделяй и властвуй.
Сперва найдём интерполяционный многочлен B для $(x_1, y_1), (x_2, y_2), \ldots, (x_{n/2}, y_{n/2})$.

$$A = B + C \cdot D, \text{ где } D = \prod_{j=1..\frac{n}{2}} (x - x_j), \text{ а } C \text{ нужно найти}$$

Подгоним правильные значения в точках $x_{n/2+1}, \ldots, x_n$, вычислим b_j, d_j – значения B и D в точках $x_{n/2+1}, \ldots, x_n \Rightarrow C$ – интерполяционный многочлен точек $(x_j, -\frac{b_j}{d_j})$ при $j = \frac{n}{2}+1 \ldots n$.

Итого $T(n) = 2T(n/2) + 2\mathcal{O}(\texttt{evaluate}(n/2))$. При $\texttt{evaluate}(n) = \mathcal{O}(n \log^2 n)$ имеем $\mathcal{O}(n \log^3 n)$.

2.2.5. 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 – $\mathcal{O}(\texttt{div}(n))$.

2.3. Литература

[sankowski]. Слайды по FFT и всем идеям разделяй и властвуй.
[e-maxx]. Про FFT и оптимизации к нему.
[codeforces]. Задачи на тему FFT.
[vk]. Краткий конспект похожих идей от Александра Кулькова.

Лекция #3: Деление многочленов

12 сентября

3.1. 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: $A(x)B(x) = 1$.
Первые n коэффициентов B можно найти 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 $\mathcal{O}(n \log n)$.
Обозначим $B_k(x) = b_0 + b_1 x + \cdots + b_{k-1}x^{k-1}$. Заметим, что $\forall k\ A(x)B_k(x) = 1 + x^k C_k(x)$.
$B_1 = b_0 = 1/a_0$. Научимся делать переход $B_k \to B_{2k}$ за $\mathcal{O}(k \log k)$.
$B_{2k} = B_k + x^k Z \Rightarrow A \cdot B_{2k} = 1 + x^k C_k + x^k A \cdot Z = 1 + x^k(C_k - A \cdot Z)$.
Выберем $Z = B_k \cdot C_k \Rightarrow C_k - A \cdot Z = C_k - C_k(A \cdot B_k) = C_k - C_k(1 + x^k C_k) = -x^k C_k^2$.
Итого $B_{2k} = B_k + B_k(x^k C_k) = B_k + B_k(1 - A \cdot B_k) \Rightarrow \boxed{B_{2k} = B_k(2 - A \cdot B_k)}$

Два умножения = $\mathcal{O}(k \log k)$. Общее время работы $n \log n + \frac{n}{2} \log \frac{n}{2} + \frac{n}{4} \log \frac{n}{4} + \cdots = \mathcal{O}(n \log n)$.
Конечно, мы обрежем B_{2k}, оставив лишь $2k$ первых членов.

• Деление многочленов

A^R – reverse многочлена. $a_0 + a_1 x + \cdots + a_n x^n \to a_n + a_{n-1}x + \cdots + a_0 x^n$.
Умножение: $A^R B^R = (AB)^R$ (доказательство: в c_{ij} += $a_i b_j$ поменяли индексы на $n-i$ и $m-j$).
Новое определение деления: по A, B хотим C: $A^R \equiv (BC)^R \bmod x^n$.
Здесь n – число коэффициентов у A, у B ровно столько же.

Обращение ряда нам даёт умение по многочлену Z: $z_0 \neq 0$ строить Z^{-1}: $Z \cdot Z^{-1} \equiv 1 \bmod x^n$.

$$C^R = (B^R)^{-1} A^R$$

Время работы: обращение ряда + умножение = $\mathcal{O}(n \log n)$.
Над кольцом делить странно, а вот над произвольным полем Фурье может не работать, тогда деление работает за $\mathcal{O}(\texttt{mul}(n) + \texttt{mul}(\frac{n}{2}) + \dots) = \mathcal{O}(\texttt{mul}(n))$ для $\texttt{mul}(n) = \Omega(n)$.

3.2. (*) 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
$x_{i+1} = x_i - (\frac{1}{x_i} - a)/(-\frac{1}{x_i^2}) = x_i + (x_i - a \cdot x_i^2) = x_i(2 - ax_i)$.
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 $\mathcal{O}(n \log n)$.

3.3. (*) Быстрое извлечение корня для чисел

Продолжаем пользоваться методом Ньютона.

$$x_{i+1} = \frac{1}{2}(x_i + \frac{a}{x_i})$$

Если у x_i k_i верных знаков, то $k_{i+1} = k_i + \Theta(1)$, а
$x_i \to x_{i+1}$ вычисляется одним делением многочленов длины k_i за $\mathcal{O}(k_i \log k_i)$.

3.4. (*) 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.

3.5. Линейные рекуррентые соотношения

Задача. Дана последовательность $f_0, f_1, \ldots, f_{k-1}, \forall n \geqslant k\ f_n = f_{n-1}a_1 + \cdots + f_{n-k}a_k$, найти f_n.

Вычисления "в лоб" можно произвести за $\mathcal{O}(nk)$.

3.5.1. Через матрицу в степени

$$A = \begin{bmatrix} a_1 & a_2 & \ldots & a_k \\ 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots \end{bmatrix},\ \forall i\ A \cdot \begin{bmatrix} f_{i-1} \\ f_{i-2} \\ \ldots \\ f_{i-k} \end{bmatrix} = \begin{bmatrix} f_i \\ f_{i-1} \\ \ldots \\ f_{i-k+1} \end{bmatrix} \Rightarrow\ A^n \cdot \begin{bmatrix} f_{k-1} \\ f_{k-2} \\ \ldots \\ f_0 \end{bmatrix} = \begin{bmatrix} f_{n+k-1} \\ f_{n+k-2} \\ \ldots \\ f_n \end{bmatrix}$$

Умножать матрицы умножаем за $\mathcal{O}(k^3) \Rightarrow$ общее время работы $\mathcal{O}(k^3 \log n)$.

3.5.2. Через умножение многочленов

На самом деле можно возводить в степень не матрицу, а многочлен по модулю...
Умножение матриц за $\mathcal{O}(k^3)$ заменяется на умножение многочленов по модулю за $\mathcal{O}(k \log k)$.

Будем выражать f_n через $f_i \colon i < n$. В каждый момент времени $f_n = \sum_j f_j b_j$.

Изначально $f_n = f_n \cdot 1$. Пока $\exists j \geqslant k \colon b_j \neq 0$, меняем $f_j b_j$ на $\left(\sum\limits_{i=1}^{k} f_{j-i}a_i\right) \cdot b_j$. (*)

Посмотрим, как меняются коэффициенты b_j. Пусть $B(x) = \sum b_j x^j$, $A(x) = x^k - \sum\limits_{i=1}^{k} x^{k-i}a_i$.

Тогда (*) – переход от $B(x)$ к $B(x) - A(x)x^{j-k}b_j$.
Изначально $B(x) = x^n \Rightarrow$ наш алгоритм – вычисление $x^n \bmod A(x)$.
Возведение многочлена x в степень n по модулю $A(x)$ – $\mathcal{O}(\log n)$ умножений и взятий по модулю.

Итого: $\mathcal{O}(k \log k \log n)$.

Лекция #4: Применение умножения многочленов

8 апреля 2019

4.1. Факторизация целых чисел

• **Вычисление** $n! \bmod m$

Возьмём $k = \lfloor\sqrt{n}\rfloor$, рассмотрим $P(x) = x(x+k)(x+2k)...(x+k(k-1))$.
$P(1)P(2)P(3)...P(k) = (k^2)!$, чтобы дополнить до $n!$, сделаем руками $\mathcal{O}(k)$ умножений.
Посчитать $P(x)$ мы можем методом разделяй и властвуй за $O(k \log^2 k)$.

• **FFT над** $\mathbb{Z}/m\mathbb{Z}$

Умножим в $\mathbb{Z}$ (то же, что $\mathbb{R}$, что $\mathbb{C}$), в конце возьмём по модулю m.
Если не хватает точности обычного вещественного типа ($m = 10^9 \Rightarrow$ `long double` уже слишком мал), то представим многочлен с коэффициентами до m, как сумму двух многочленов с коэффициентами до $k = \lceil m^{1/2} \rceil$: $P(x) = P_1(x) + k \cdot P_2(x), Q(x) = Q_1(x) + k \cdot Q_2(x)$.
Сделаем 3 умножения, как в Карацубе:

$$P(x)Q(x) = P_1Q_1 + k^2P_2Q_2 + k((P_1{+}Q_1) \cdot (P_2{+}Q_2) - P_1Q_1 - P_2Q_2)$$

• **Факторизация**

Найдём $\min d$: $gcd(d!, n) \neq 1$. Тогда d – минимальный делитель n. Можно искать бинпоиском, можно "двоичными подъёмами", тогда время $= \mathcal{O}(calc(n^{1/2}!)) = \mathcal{O}(n^{1/4} \log^2 n)$.

4.2. CRC-32

Один из вариантов хеширования последовательности бит $a_0, a_1, \ldots, a_{n-1}$ – взять остаток от деления многочлена $A(x) \cdot x^k$ на $G(x)$, где $G(x)$ – специальный многочлен, а $k = \deg G + 1$.

В *CRC-32-IEEE-802.3* (в v.42, mpeg-2, png, cksum) $k = 32$, $G(x) =$ 0xEDB88320 (32 бита).
Если размер машинного слова $\geqslant k$, CRC вычисляется за $\mathcal{O}(n)$, в общем случаем за $\mathcal{O}(n \cdot \lceil \frac{k}{w} \rceil)$.

```
1 for i = n-1..k:
2     if a[i] != 0:
3         a[i..i-k+1] ^= G
```

Упражнение 4.2.1. `CRC(A ^ B)` = `CRC(A) ^ CRC(B)`, `CRC(concat(A, 1))` = (`CRC(A)` $\cdot 2 + 1$) $\bmod G$

4.3. Кодирование бит с одной ошибкой

Контекст. По каналу хотим передать n бит.
В канале может произойти не более 1 ошибки вида "замена бита".

Детектирование ошибки. Передадим $a_1, a_2, \ldots, a_n$ и $b =$ `XOR`$(a_1, a_2, \ldots, a_n)$.
Если b' равно `XOR`$(a'_1, a'_2, \ldots, a'_n)$, ошибки при передачи не было.

Исправление ошибки.
Удвоение бит не работает – и из 0, и из 1 в результате одной ошибки может получиться 01.
Работает утроение бит ($3n$) или "удвоение бит и дополнительно передать `XOR`" ($2n{+}1$).
Раскодирование для $2n{+}1$: $00 \to 0,\ 11 \to 1,\ 01/10 \Rightarrow$ подгоняем, чтобы сошёлся `XOR`.

Исправление ошибки за $\lceil \log_2 n \rceil + 1$ дополнительных бит.
Передадим два раза $\texttt{XOR}(a_1, a_2, \ldots, a_n)$. Теперь мы знаем, есть ли ошибка среди $a_1, a_2, \ldots, a_n$.
Если ошибка есть, её нужно исправить. $\forall b \in [0, \lceil \log_2 n \rceil)$ передадим $\texttt{XOR}_{i:\, bit(i,b)=0}(a_i)$.
В итоге мы знаем все $\lceil \log_2 n \rceil$ бит позиции ошибки.

4.4. Коды Рида-Соломона

Задача. Кодируем n элементов конечного поля $\mathbb{F}_q$.
Канал допускает k ошибок. Хотим научиться исправлять ошибки после передачи.

Замечание 4.4.1. Конченые поля имеют размер p^k ($p \in Prime, k \in \mathbb{N}$). Поле размера p – $\mathbb{Z}/p\mathbb{Z}$.
Поле размера p^k – остатки по модулю неприводимого многочлена над $\mathbb{F}_p$ степени k.
Для кодирования битовых строк удобно использовать $q = 2^k$ при $k \in \{32, 64, 4096\}$.

Lm 4.4.2. Если обозначить расстояние Хэмминга как $D(s,t)$, а $f(s)$ – код строки s, канал передачи допускает k ошибок, то исправить ошибки можно iff $\forall s \neq t \;\; D(f(s), f(t)) \geqslant 2k{+}1$.

Доказательство. С учётом ошибок строка s может перейти в любую точку шара радиуса k с центром в $f(s)$. Если такие шары пересекаются, $\forall$ точку пересечения не раскодировать. ■

Код Рида-Соломона. Данные $a_0, \ldots a_{n-1}$ задают многочлен $A(x) = \sum a_i x^i$. Передадим значения многочлена в произвольных $n{+}2k{+}1$ различных точках (здесь мы требуем $p \geqslant n$).

Теорема 4.4.3. Корректность кодов Рида-Соломона.

Доказательство. $A(x) \neq B(x) \Rightarrow (A-B)(x)$ имеет не более n корней $\Rightarrow A(x)$ и $B(x)$ имеют не более n общих значений $\Rightarrow$ хотя бы $2k{+}1$ различных $\Rightarrow D(f(A), f(B)) \geqslant 2k{+}1$. ■

Выбор точек и q: хочется применить FFT. На практике можно отправлять данные такими порциями, что $n{+}2k{+}1 = 2^b$. Нужно q вида $a \cdot 2^b + 1$ и x: $ord(x) = 2^b$, точка $w_i = x^i, \;\; i \in [0, 2^b)$.

• Декодирование

Мы доказали возможность однозначного декодирования, осталось предъявить алгоритм.
Имели $A(x)$, $\deg A = n{-}1$, передали $A(w_0), A(w_1), \ldots A(w_{n+2k})$. Получили на выходе данные с ошибками $A'(w_0), A'(w_1), \ldots A'(w_{n+2k})$. Посчитали интерполяционный многочлен $A'(x)$.

Утверждение 4.4.4. Если у $A'(x)$ старшие $2k{+}1$ коэффициентов нули, ошибок не было.

Итого, если ошибок нет, декодирование при желании можно сделать за $\mathcal{O}(n \log n)$ через FFT.

Обозначим позиции ошибок $e_1, e_2, \ldots, e_k$.
Может быть, ошибок меньше. Главное, что в позициях кроме e_i ошибок точно нет.
Многочлен ошибок $E(x) = (x - w_{e_1})(x - w_{e_2}) \ldots (x - w_{e_k})$, $B(x) = A(x)E(x)$, $B'(x) = A'(x)E(x)$.
$\forall i \notin \{e_j\}\; A(w_i) = A'(w_i) \Rightarrow \forall i \; B(w_i) = B'(w_i)$. Запишем это, как СЛАУ $\forall i \; B(w_i) = A'(w_i)E(w_i)$, где неизвестные – коэффициенты многочленов B ($n{+}k{+}1$ штук) и E (k штук).

Теорема 4.4.5. Для записанной СЛАУ $\exists!$ решение.

Следствие 4.4.6. Декодирование: решим СЛАУ, найдём $A(x) = \frac{B(x)}{E(x)}$.

Асимптотика декодирования: Гаусс $\mathcal{O}((n+k)^3)$. Бэрликэмп-Мэсси $\mathcal{O}((n+k)^2)$.
Ссылка на более крутые алгоритмы декодирования в разд. 4.6.

4.5. Применения fft в комбинаторике

• Возведение в степень

2^n бинпарным возведением в степень вычисляется за $T(n) = T(\frac{n}{2}) + n \log n = \mathcal{O}(n \log n)$.

• Умножение многочленов от нескольких переменных

Посчитать $C(x, y) = A(x, y) \cdot B(x, y)$, пусть $\deg_x \leqslant n, \deg_y \leqslant m$, тогда возьмём $y = x^{2n+1}$ и вычислим $C'(x) = A'(x) \cdot B'(x)$, мономы вида $ax^{(2n+1)i+j}$, $j \leqslant 2n$ заменим на $ax^j y^i$. $\mathcal{O}(nm \log nm)$.

4.5.1. Покраска вершин графа в k цветов

Предподсчитаем за $\mathcal{O}(2^n)$ все независимые множества I (те, что можно покрасить в один цвет).

Рассмотрим любую корректную покраску в k цветов $I_1, I_2, \dots I_k$: $\sqcup I_j = V$,
заметим $\sum I_j = 2^n - 1, \sum |I_j| = n$.
Рассмотрим $P(x, y) = \sum_I x^I y^{|I|}$, внимательно посмотрим на $P^k(x, y)$:

Теорема 4.5.1. Коэффициент в $P^k(x, y)$ монома $x^{2^n-1} y^n$ – это число покрасок в k цветов.

Lm 4.5.2. $A \cap B = \varnothing \Leftrightarrow |A| + |B| = |A \cup B|$

Lm 4.5.3. $A \cap B \neq \varnothing \Leftrightarrow A + B > A \cup B$
(сложение/сравнение множеств – операции с битовыми масками)

Алгоритм – возведение многочлена степени $2^n n$ в степень k.
Одно умножение работает за $\mathcal{O}(2^n n^2)$, возведение в степень за $\mathcal{O}(2^n n^2 \log k)$.

4.5.2. Счастливые билеты

Задача. Массив из $2n$ цифр из множества $d_1, d_2, \dots, d_k$ называется счастливым, если $\sum$ первых n цифр совпадает с $\sum$ последних n. Найти число счастливых массивов из $2n$ цифр.

Рассмотрим $P(x) = (x^{d_1} + x^{d_2} + \dots + x^{d_k})^n$. Ответ – сумма квадратов коэффициентов P.
Алгоритм = возведение в степень. Время возведения в степень – $\mathcal{O}(\log n)$ умножений.
Точнее $\mathcal{O}(mul(N) + mul(\frac{N}{2}) + mul(\frac{N}{4}) + \dots) = \mathcal{O}(N \log N)$, где $\deg P = N = n \cdot \max d_i$.

4.5.3. 3-SUM

Задача. Даны n чисел $a_i \in \mathbb{Z} \cap [S]$, найти i, j, k: $a_i + a_j + a_k = S$.
Решение #1. Сортировка, далее $\forall i$ два указателя для j, k. $\mathcal{O}(n^2)$.
Решение #2. Возьмём $P(x) = \sum_i x^{a_i}$, рассмотрим P^3, возьмём коэффициент при x^S. $\mathcal{O}(S \log S)$.

4.5.4. Применение к задаче о рюкзаке

Простая задача. Subsetsum. Даны n целых $a_i > 0$, выбрать подмножество: $\sum_j a_j = S$.
Посчитаем $P(x) = \prod_i (1 + x^{a_i})$, возьмём коэффициент при x^S. n умножений $\Rightarrow \mathcal{O}(nS \log S)$.

Алгоритм 4.5.4. Сложная задача. *То же, но выбрать подмножество размера ровно k.*

Посчитаем $P(x, y) = \prod_i (1 + yx^{a_i})$, возьмём коэффициент при $x^S y^k$.
Будем вычислять разделяйкой: $T(n) = 2T(\frac{n}{2}) + nS \log nS$ (nS – степень многочлена).
Получаем $\mathcal{O}(nS \log nS \log n)$, что лучше базовой динамики за $\mathcal{O}(n^2 S)$ (`dp[i,size,sum]`).

4.5.5. (*) Сверхбыстрый рюкзак за $\tilde{O}(\sqrt{n}S)$

Решаем subsetsum. Пусть $A = \{a_i\}$. Если мы разобьём $A = A_1 \sqcup \cdots \sqcup A_k$ и для каждого A_i насчитаем массив f_{ij} = можем ли мы набрать вес j, используя предметы из A_i, то останется только за $\mathcal{O}(kS \log S)$ перемножить $F_1(x) \cdot \ldots \cdot \cdots \cdot F_k(x)$, где $F_i(x) = \sum f_{ij} x^j$.

Возьмём $k \approx \sqrt{n}$, $A_i = \{x \in A \colon x \bmod k = i\}$. $x \in A_i \Rightarrow x = ky + i \Rightarrow$
рассмотрим $B_i = \{y \colon ky + i \in A_i\}$ и для B_i воспользуемся 4.5.4, который насчитает
$\sum f_{ijt} x^j y^t$, где f_{ijt} = число способов набрать сумму ровно j, используя ровно t предметов из B_i,
при этом $jk + it \leqslant S \Rightarrow j \leqslant \lfloor \frac{S}{k} \rfloor \Rightarrow$ 4.5.4 отработает за $\mathcal{O}(\frac{S}{k} n_i \log n_i \log nS)$, где $n_i = |A_i|$.

Итого: решили subsetsum за $\mathcal{O}(kS \log S) + \sum n_i \mathcal{O}(\frac{S}{k} n_i \log n_i \log nS) =$
$\mathcal{O}(kS \log S) + \mathcal{O}(\frac{S}{k} n \log n \log S) \Rightarrow$ оптимальное $k = \sqrt{n \log n}$, subsetsum за $\mathcal{O}(\sqrt{n \log n} \cdot S \log S)$.
$\tilde{O} f = \mathcal{O}(f \cdot poly(\log))$.

4.5.6. (*) Сверхбыстрый рюкзак за $\tilde{O}(n + S)$

Будем решать задачу $f(A)$**:** определить $\forall s \in [0,S]$, можно ли набрать вес s.

Если есть ответы $f(A)$ и $f(B)$, то `fft` за $\mathcal{O}(S \log S)$ даёт ответ для $A+B$. Назовём это свёрткой.
Если мы разделим множество предметов A на $A = A_1 \sqcup A_2 \sqcup \cdots \sqcup A_k$, мы можем для каждой части A_i решить задачу и свёртками за $\mathcal{O}(kS \log S)$ получить ответ для A.

Хорошее разделение: $m = \lceil \log n \rceil$, $A_i = A \cap (\frac{S}{2^i}, \frac{S}{2^{i-1}}], i \in [1, m]$, $A_{m+1} = A \cap [1, \frac{S}{2^m}]$.
Для A_{m+1} делаем разделяйку с `fft` даёт $T(n) = 2T(\frac{n}{2}) + \texttt{fft}(n\frac{S}{2^m}) = \mathcal{O}(S \cdot n \log n \cdot \log)$.
Для A_i рассмотрим множество-ответ X_i, заметим $|X_i| < 2^i$. Обозначим $k = 2^i$.
Поделим A_i случайным образом на k множеств: $A_i = \sqcup A_{ij}$, $\mathsf{E}(\max_j(X_i \cap A_{ij})) = \mathcal{O}(\log k) = \varepsilon$,
чтобы решить задачу для A_{ij} разделим его на ε^2 случайных множеств A_{ijt}.

$Pr[\forall t \ |A_{ijt} \cap X_i| \leqslant 1] \geqslant \frac{1}{2} \Rightarrow$ ответ для A_{ijt} тривиален: мы можем взять $\leqslant 1$ предмета $\Rightarrow$ можем набрать только суммы $s \in A_{ijt}$. Ответ для $A_{ij} = \varepsilon^2$ свёрток, при этому в ответе нам нужны суммы не до S, а до $\frac{S}{2^i}\varepsilon$, так как $\max A_{ij} \leqslant \frac{S}{2^i}$ и $|A_{ij} \cap X_i| \leqslant \varepsilon$. Осталось свернуть ответы для A_{ij} в ответ для A_i – разделяйка длины 2^i, где в листьях `fft` от длины $\frac{S}{2^i}\varepsilon \Rightarrow$ время на свёртки для A_i:
$2^i \log(2^i) M \log M$, где $M = \frac{S}{2^i}\varepsilon \Rightarrow \mathcal{O}(S \log 2^i \log M\varepsilon) = \mathcal{O}(S \log^3)$. Итак $\forall i \Rightarrow \mathcal{O}(S \log^4) = \tilde{O}(S)$.

4.6. Литература

[Shuhong Gao'2002]. Декодирование Рида-Соломона через расширенного Евклида.
[Koiliaris Xu'2018]. Subset Sum in $\tilde{\mathcal{O}}(\sqrt{n}S)$.