Алгоритмы и структуры данных

Лекция 20
Числа.

Сергей Леонидович Бабичев

Длинная арифметика.

- Длинные числа представляются в виде полиномов вида

$$X = x_n \cdot R^n + x_{n-1} \cdot R^{n-1} + \cdots + x_1 \cdot R^1 + x_0$$

- Здесь R — основание системы счисления представления длинных чисел.
- R должно быть удобно хранить в единице хранения, *чанке*.
- Это обычно степень двойки или степень десятки.
- Операции над длинными числами — операции над полиномами.
- Будем называть (n)-числами те, что состоят из n чанков (имеют n цифр в системе счисления по основанию R). Это — степень соответствующего полинома.

- *Сложение и вычитание*. $(n)-$число и $(m)-$число складываются и вычитаются за $\Theta(\max(n, m))$.
- *Умножение*. Наивным образом (n)-число и (m)-число умножаются за $\Theta(n \cdot m)$. Алгоритмом Карацубы два n числа умножаются за $\Theta(n^{\log_2 3})$.
- Имеется более быстрый алгоритм умножения, основанный на комплексной арифметике.

Быстрое преобразование Фурье

Lemma (Восстановление коэффициентов полинома)

Пусть имеется полином

$$P(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$$

и имеются n *значений полинома* P *в попарно различных точках* $y_0 = P(x_0), y_1 = P(x_1), \ldots, y_{n-1} = P(x_{n-1})$.
Тогда можно восстановить коэффициенты полинома решением системы линейных уравнений

$$\begin{cases} a_0 + a_1 x_0 + \cdots + a_{n-1} x_0^{n-1} = y_0 \\ a_0 + a_1 x_1 + \cdots + a_{n-1} x_1^{n-1} = y_1 \\ \cdots \\ a_0 + a_1 x_{n-1} + \cdots + a_{n-1} x_{n-1}^{n-1} = y_{n-1} \end{cases} \tag{1}$$

относительно a_i.

Восстановление коэффициентов полинома

Заменим на матричное уравнение:

$$\begin{pmatrix} x_0^0 & x_0^1 & \dots & x_0^{n-1} \\ x_1^0 & x_1^1 & \dots & x_1^{n-1} \\ \dots & \dots & \dots & \dots \\ x_{n-1}^0 & x_{n-1}^1 & \dots & x_{n-1}^{n-1} \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ \dots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \dots \\ y_{n-1} \end{pmatrix} \tag{2}$$

- Слева — матрица Вандермонда. Из курса ЛА известно, что её определитель ненулевой, если $x_i \neq x_j$ для любых $i \neq j$.

Перемножение полиномов

- Требуется найти произведение $R = P \cdot Q$.
- Не умаляя общности возьмём $n = \deg P + \deg Q + 1 \geqslant \deg R + 1$.
- Найдём в n точках значения $P(x_i)$ и $Q(x_i)$.
- $R(x_i) = P(x_i) \cdot Q(x_i)$.
- Восстановим R по набору $R(x_i)$.
- Требуется: быстро находить значения полинома в n точках и быстро восстанавливать значения полинома по n значениям.
- Мы можем выбирать произвольные x_i. Выберем n комплексных корней степени n единицы.

Свойства пробных точек

- Такие корни есть точки на единичной окружности в комплексной плоскости.
- $x_k = e^{i\frac{2\pi k}{n}} = \cos\dfrac{2\pi k}{n} + i\sin\dfrac{2\pi k}{n}$
- Обозначим за ω первый корень $x_1 = \cos\dfrac{2\pi}{n} + i\sin\dfrac{2\pi}{n}$.
- Заметим, что $\omega^2 = x_2, \omega^3 = x_3, \ldots, \omega^n = x_0$.
- Мы получили мультипликативную группу степени n, а ω — примитивный корень группы.
- За ω можно взять любой из примитивных корней группы.

Перемножение полиномов

- Итак, нужно:

1. для ω_k найти значения полиномов $P(\omega_k)$ и $Q(\omega_k)$;
2. найти произведения значений полиномов $R(\omega_k)$;
3. восстановить коэффициенты полинома R.

Шаг 1. Вычисляем $P(\omega^0), P(\omega^1), \ldots, P(\omega^{n-1})$

- $P(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$.
- Введём два полинома:

$$\begin{aligned} P_0 = &\quad a_0 + a_2 x + a_4 x_2 + \ldots \\ P_1 = &\quad a_1 + a_3 x + a_5 x_2 + \ldots . \end{aligned}$$

Тогда

$$P(x) = P_0(x^2) + x \cdot P_1(x^2)$$

Шаг 1. Вычисляем значения вспомогательных полиномов

- Положим, что n — чётное число.
- Тогда для вычисления P_0 и P_1 достаточно взять их значения в чётных точках $\omega^0, \omega^2, \omega^4, \ldots$.
- Задача вычисления в n точках свелась к двум задачам вычисления значения в $\frac{n}{2}$ точках и консолидации результата за $\Theta(\frac{n}{2}$.

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n).$$

$$T(n) = O(n \log n).$$

Шаг 2. Вычисляем значения R

- Тривиальный шаг.

$$\begin{aligned} R(\omega^0) &= P(\omega^0) \cdot Q(\omega^0) \\ R(\omega^1) &= P(\omega^1) \cdot Q(\omega^1) \\ &\dots \\ R(\omega^{n-1}) &= P(\omega^{n-1}) \cdot Q(\omega^{n-1}) \end{aligned}$$

Время работы $\Theta(n)$.

Шаг 3. Восстановление коэффициентов R.

- Что мы сделали, вычисляя значения в точках:

$$\begin{pmatrix} P(\omega^0) \\ P(\omega^1) \\ \dots \\ P(\omega^{n-1}) \end{pmatrix} = \begin{pmatrix} (\omega^0)^0 & (\omega^0)^1 & \dots & (\omega^0)^{n-1} \\ (\omega^1)^0 & (\omega^1)^1 & \dots & (\omega^1)^{n-1} \\ \dots & \dots & \dots & \dots \\ (\omega^{n-1})^0 & (\omega^{n-1})^1 & \dots & (\omega^{n-1})^{n-1} \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ \dots \\ a_{n-1} \end{pmatrix} \tag{3}$$

- Свойство данной матрицы: $W_{ij} = \omega^{ij}$.
- Это произведение удалось найти за $n \log n$.
- Сейчас предстоит обратная задача: по левой части P и матрице W восстановить вектор a.
- Достаточно умножить левую часть на обратную матрицу W^{-1}.

Матрица W

- В матрице W элемент $W_{ij} = \omega^{ij}$.

$$W = \begin{pmatrix} \omega^0 & \omega^0 & \dots & \omega^0 \\ \omega^0 & \omega^1 & \dots & \omega^{n-1} \\ \dots & \dots & \dots & \dots \\ \omega^0 & \omega^{(n-1)} & \dots & \omega^{(n-1)(n-1)} \end{pmatrix} \tag{4}$$

Матрица V

- Введём матрицу V, элемент $V_{ij} = \omega^{-ij}$.

$$V = \begin{pmatrix} \omega^0 & \omega^0 & \dots & \omega^0 \\ \omega^0 & \omega^{-1} & \dots & \omega^{-(n-1)} \\ \dots & \dots & \dots & \dots \\ \omega^0 & \omega^{-(n-1)} & \dots & \omega^{-(n-1)(n-1)} \end{pmatrix} \tag{5}$$

Произведение $V \cdot W$

$$U = \begin{pmatrix} \omega^0 & \omega^0 & \dots & \omega^0 \\ \omega^0 & \omega^{-1} & \dots & \omega^{-(n-1)} \\ \dots & \dots & \dots & \dots \\ \omega^0 & \omega^{-(n-1)} & \dots & \omega^{-(n-1)(n-1)} \end{pmatrix} \cdot \begin{pmatrix} \omega^0 & \omega^0 & \dots & \omega^0 \\ \omega^0 & \omega^1 & \dots & \omega^{n-1} \\ \dots & \dots & \dots & \dots \\ \omega^0 & \omega^{(n-1)} & \dots & \omega^{(n-1)(n-1)} \end{pmatrix}$$

- Вычислим значение U_{ij}
- $U_{ij} = \sum\limits_{k=0}^{n} (\omega^{-1})^{ik} \cdot \omega^{kj} = \sum\limits_{k=0}^{n} (\omega^{j-i})^k$
- При $j = i$ каждый член суммы единица $T_{ij} = n,$ если $j = i.$
- При $j \neq i$

$$T_{ij} = \sum_{k=0}^{n} (\omega^{j-i})^k = (\omega^{j-i})^0 + (\omega^{j-i})^1 + \dots + (\omega^{j-i})^{n-1} = \frac{1 - (\omega^{j-i})^n}{1 - (\omega^{j-i})} = 0$$

Матрица $T = V \cdot W$

$$T = \begin{pmatrix} n & 0 & \dots & 0 \\ 0 & n & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & n \end{pmatrix} = n \cdot E.$$

$$W^{-1} = \frac{1}{n} V \tag{6}$$

Шаг 3. Восстановление коэффициентов R.

- Исполняем шаг 1 с заменой $\omega = \omega^{-1}$.
- В получившемся результате делим все коэффициенты на n.
- Шаг называется *обратным преобразованием Фурье*.
- $T = O(n \log n)$.

Проблемы с БФТ

1. Для перемножения полиномов требуется дополнение нулями до требуемой степени 2.
2. Потеря точности.

- Если перемножаем целочисленные многочлены, то при коэффициентах $|R_i| <\approx 10^{11}$ погрешность не превышает 0.5.

Общий алгоритм БПФ. Рекурсивная реализация.

1. На входе алгоритма FFT имеется массив $a[n = 2^k]$ и логический флаг обращения inv.
2. Создаём два массива размера $a[n/2]$, в $a0$ отправляем все чётные элементы, в $a1$ — все нечётные.
3. Рекурсивно вызываем $FFT(a0, inv)$ и $FFT(a1, inv)$.
4. Находим $\omega = (\cos(2\pi/n) + i\sin(2\pi/n))$. Если флаг inv установлен, то меняем знак мнимой части.
5. Первая половина массива a вычисляется как в цикле по k как

$$a_i = a0_i \cdot k + \omega^k \cdot a1_k.$$

6. Вторая половина массива a вычисляется как в цикле по k как

$$a0_{i+\frac{n}{2}} = a0_i \cdot k - \omega^k \cdot a1_k.$$

7. Если установлен флаг инверсии, то в цикле каждый присвоенный элемент делится на 2.

Алгоритм умножения полиномов $a[n]$ и $b[m]$

.

1. Устанавливаем размер выходного полинома как ближайшую степень двойки, большая или равная $2^k = n + m + 1$.
2. Создаём полиномы fa и fb как копии a и b и дополняем нулями до длины 2^k.
3. Проводим прямое преобразование Фурье для массивов fa и fb: $FFT(fa, false), FFT(fb, false)$.
4. Попарно перемножаем коэффициенты: $fa_i = fa_i \cdot fb_i$.
5. Проводим обратное преобразование Фурье: $FFT(fa, true)$.
6. Округляем результат и передаём его в возвращаемое значение.