Лекция 16

Алгоритм Шёнхаге-Штрассена

(Конспект: Т. Лазовская)

Сложение n-битовых чисел можно осуществить за $O(n)$ (во-первых, у нас имеется такая инструкция RAM-машины; во-вторых, ясно, что это действительно можно реализовать физически). Рассмотрим умножение n-битовых чисел (пусть n — степень двойки; можно считать так в любом случае, так как иначе время работы увеличится не более, чем в константу раз). Умножение «в столбик» даст, очевидно, время $\Omega(n^2)$. Хотелось бы построить более быстрый алгоритм.

Почему мы не включили операцию умножения в RAM-машину? Именно потому, что непонятно, как физически реализовать ее так, чтобы она работала время $O(n)$. Мы могли бы включить другие инструкции (и включим их, так как они нам понадобятся): умножение и деление на степень двойки, т.е. сдвиг двоичного представления:

$$\begin{array}{ll} \text{LSHIFT} & a, \\ \text{RSHIFT} & a \end{array}$$

(содержимое нулевого регистра сдвигается влево или вправо на a битов, т.е. умножается или делится на 2^a; RSHIFT — обобщение операции HALF). Ясно, что эту инструкцию совсем просто реализовать.

16.1 Простой алгоритм

Начнем с простого алгоритма, время работы которого составляет $O(n^{\log_2 3})$.

Имеем два n-битовых числа a и b. Разделим их в битовом представлении на $n/2$-битовые a_1, a_2 и b_1, b_2 (соответственно), а затем перемножим эти числа рекурсивно:

$$\begin{aligned} a &= a_1 \cdot 2^{n/2} + a_2, \\ b &= b_1 \cdot 2^{n/2} + b_2, \\ a \cdot b &= a_1 b_1 \cdot 2^n + (a_1 b_2 + a_2 b_1) \cdot 2^{n/2} + a_2 b_2; \end{aligned}$$

средний коэффициент можно вычислить, используя лишь одно умножение и остальные два коэффициента:

$$a_1b_2 + a_2b_1 = (a_1 + a_2)(b_1 + b_2) - a_1b_1 - a_2b_2.$$

Тем самым получаем, что нам достаточно трех умножений $n/2$-битовых чисел (ибо умножение на степень двойки — по существу, не умножение), т.е. рекуррентное уравнение для времени работы —

$$T(n) \leqslant 3T(n/2) + cn.$$

Тогда (в первой части курса была соответствующая теорема) $T(n) = O(n^{\log_2 3})$.

Замечание 16.1. Нюанс: строго говоря, мы перемножали не $n/2$-битные, а $(n/2+1)$-битные числа $x = a_1 + a_2$ и $y = b_1 + b_2$, но одно к другому сводится за линейное время. Действительно, пусть $x = 2A + x'$, $y = 2B + y'$, где A и B — $n/2$-битные числа, x' и y' — биты. Тогда $xy = 4AB + 2Ay' + 2Bx' + x'y'$. «Сложным» умножением здесь является только $A \cdot B$; остальные «умножения» реализуются за линейное время, поскольку это умножения на степени двойки или 0.

$O(n^{\log_2 3})$ — лучше, чем $O(n^2)$, но все еще много. Поэтому будем строить другой алгоритм.

16.2 Дискретное преобразование Фурье (ДПФ)

Пусть R — кольцо (в нем имеются $0, 1, +, \cdot$). Будем работать с векторами размерности n над R. Пусть в R имеется w — первообразный корень степени n из 1, и пусть существует n^{-1}.

Определение 16.1. Для вектора $a = (a_0, a_1, ..., a_{n-1})^t$ (в виде столбца) определим *преобразование Фурье* от него:

$$F(a) = Aa,$$

где $A_{ij}{=}w^{ij}$. *Обратное преобразование Фурье* — преобразование с матрицей $(A^{-1})_{ij} = n^{-1}w^{-ij}$.

Задача 16.1. Показать, что оно действительно обратное ($A^{-1}A = E$).

Использовать его мы будем примерно так. У нас векторы a и b задают числа (а можно считать — что многочлены). Сделаем ДПФ: $a \mapsto F(a)$, $b \mapsto F(b)$. После этого перемножим поэлементно $F(a)$ и $F(b)$. Получим что-то типа произведения значений многочленов в соответствующих точках; затем применим обратное ДПФ и получим искомый результат ab.

Сформулируем строгие утверждения.

Определение 16.2. Пусть $a = (a_0, \dots, a_{n-1})^t$, $b = (b_0, ..., b_{n-1})^t$. Определим новую операцию *свертка* ($\odot$); $a \odot b$ — это $2n$-мерный вектор с элементами

$$(a \odot b)_i = \sum_{j=0}^{n-1} a_j b_{i-j}$$

для $0 \leqslant i \leqslant 2n-1$; здесь и далее элементы, которые не определены (например, b_{-1}) считаются равными 0.

Замечание 16.2. Коэффициенты свертки равны коэффициентам произведения многочленов $\sum_i a_i x^i$ и $\sum_i b_i x^i$.

Лемма 16.1. *Дополним a и b до $2n$ элементов: $a'' = (a_0, \dots, a_{n-1}, 0, \dots, 0)^t$, $b'' = (b_0, \dots, b_{n-1}, 0, \dots, 0)^t$. Теперь применим ДПФ к каждому из них и перемножим результаты покомпонентно. Затем применим обратное ДПФ. Так мы получим вектор длины $2n$, равный свертке $a \odot b$.*

Упражнение 16.1. Доказать лемму 16.1.

Определение 16.3 (отрицательно обернутая свертка). Это n-мерный вектор $a \ominus b$ с элементами $(a \ominus b)_i = \sum_{j=0}^{i} a_j b_{i-j} - \sum_{j=i+1}^{n-1} a_j b_{n+i-j}$.

Лемма 16.2. *Пусть у нас есть не только w, но и ψ — корень из него ($\psi^2 = w$). Рассмотрим вектора*

$$\begin{aligned}
a' &= (a_0, a_1\psi, \dots, a_{n-1}\psi^{n-1}), \\
b' &= (b_0, b_1\psi, \dots, b_{n-1}\psi^{n-1}), \\
d' &= (d_0, d_1\psi, \dots, d_{n-1}\psi^{n-1}),
\end{aligned}$$

где d_i — коэффициенты отрицательно обернутой свертки ($d = a \ominus b$). Утверждение: $d' = F^{-1}(F(a') * F(b'))$, где $*$ — покомпонентное произведение двух векторов.

Доказательство. Достаточно показать, что $F(a') * F(b') = F(d')$.

$$\begin{aligned}
(F(a'))_i &= \sum_{j=0}^{n-1} w^{ij}\psi^j a_j, \\
(F(b'))_i &= \sum_{j'=0}^{n-1} w^{ij'}\psi^{j'} b_{j'}, \\
(F(d'))_i &= \sum_{j''=0}^{n-1} w^{ij''}\psi^{j''} \left(\sum_{k=0}^{j''} a_k b_{j''-k} - \sum_{k=j''+1}^{n-1} a_k b_{n+j''-k} \right).
\end{aligned}$$

Теперь перемножим:

$$(F(a'))_i * (F(b'))_i = \sum_{j=0}^{n-1} \sum_{j'=0}^{n-1} a_j b_{j'} \psi^{j+j'} w^{i(j+j')}.$$

Слагаемые, соответствующие $j + j' < n$, имеются и в $F(d')_i$ (при этом слагаемое с $\psi^{j+j'}$ соответствует слагаемому с $\psi^{j''}$). Заметим, что $\psi^n = -1$, $w^n = 1$. Поэтому слагаемые, соответствующие $j + j' \geqslant n$, могут быть преобразованы так:

$$a_j b_{j'} \psi^{j+j'} w^{i(j+j')} = -a_j b_{j'} \psi^{j+j'-n} w^{i(j+j'-n)},$$

но это как раз оставшиеся слагаемые из $F(d')_i$ (теперь слагаемое с $\psi^{j+j'-n}$ соответствует слагаемому с $\psi^{j''}$). $\square$

16.3 Алгоритм Шёнхаге-Штрассена[1]

Достаточно научиться перемножать n-битные числа по $\bmod\ (2^n + 1)$. (Чтобы перемножить точно, можно представить n-битные числа как $2n$-битные.) Не умаляя общности, можно считать, что $n = 2^k$.

Требуется перемножить два n-битных числа, u и v. Разделим их двоичные представления на блоки длиной l битов каждый, где $n = lb$, $b = 2^{k/2}$ или $b = 2^{(k-1)/2}$ (в зависимости от четности k). Очевидно, b — количество блоков.

$$\begin{aligned} u &= (u_0, u_1, \ldots, u_{b-1})^t, \\ v &= (v_0, v_1, \ldots, v_{b-1})^t, \\ w &= (w_0, w_1, \ldots, w_{b-1})^t, \\ w &= uv \bmod (2^n + 1) \text{ (как числа).} \end{aligned}$$

(Если задача тривиальна, т.е. один из сомножителей — 2^k, этот алгоритм не используют.) Нам надо найти w_i. Рассмотрим uv, еще не взятое по модулю $2^n + 1$; оно состоит из блоков

$$y_i = \sum_{j=0}^{b-1} u_j v_{i-j};$$

именно,

$$uv = y_0 + y_1 \cdot 2^l + \ldots + y_{b-1} \cdot 2^{l(b-1)} + y_b \cdot 2^{lb} + \ldots + y_{b+i} \cdot 2^{l(b+i)} + \ldots$$

Поскольку $2^{lb} \equiv -1 \pmod{2^n + 1}$, получаем $w_i = y_i - y_{b+i}$, т.е. w_i — коэффициенты отрицательно обернутой свертки.

Мы знаем для $i < b$, что $0 \leqslant y_i \leqslant (1 + i)2^{2l}$, т.к. в $y_i = \sum_{j=0}^{b-1} u_j v_{i-j}$ всего $1 + i$ ненулевых членов. Также $0 \leqslant y_i \leqslant (b - 1 - i)2^{2l}$ для $i \geqslant b$. Следовательно, w_i достаточно вычислить по модулю $\geqslant b2^{2l}$.

Пусть $w'_i = w_i \bmod b$, $w''_i = w_i \bmod (2^{2l}+1)$ (одно из них мы вычислим с помощью преобразования Фурье, другое — с помощью первого, «простого», алгоритма). Тогда можно найти и $w_i = (2^{2l}+1)[(w'_i - w''_i) \bmod b] + w''_i$.

[1] A. Schönhage; V. Strassen

Найдем w_i''. Рассмотрим кольцо остатков по $\bmod\ (2^{2l}+1)$. Пусть $\psi = 2^{2l/b}$. Убедимся, что он — корень из первообразного корня степени b: $\psi^b = 2^{2l} \equiv -1 \pmod{2^{2l}+1}$. Теперь, чтобы найти w_i'', можно воспользоваться леммой 16.2. Для этого надо уметь делить и умножать на степени ψ, но это легко, поскольку ψ — степень двойки. Кроме того, надо быстро уметь находить преобразование Фурье и обратное к нему (перемножать матрицу на вектор — слишком долго); это мы научимся делать на следующей лекции. И, наконец, надо поэлементно перемножить два вектора ($F(u')$ и $F(v')$), т.е. уметь умножать $2l$-битные числа; для этого мы рекурсивно воспользуемся нашим алгоритмом.

Найдем w_i'. Воспользуемся первым («простым») алгоритмом, но — для всех w_i' сразу. Рассмотрим два числа из $3b\log b$ битов:

$$\begin{aligned}&|0 \quad 0 \quad (u_0 \bmod b)|0 \quad 0 \quad (u_1 \bmod b)|\dots|,\\ &|0 \quad 0 \quad (v_0 \bmod b)|0 \quad 0 \quad (v_1 \bmod b)|\dots| \text{ —}\end{aligned}$$

дополнили наши числа нулями таким образом, чтобы получилось b блоков по $3\log b$ битов. Перемножим эти длинные числа «простым» алгоритмом и получим

$$|y_0 \bmod b|y_1 \bmod b|\dots|y_{b-1} \bmod b| \text{ —}$$

нули мы записывали так, чтобы после перемножения блоки не пересекались. Осталось попарно вычесть полученные числа: $w_i' = (y_i \bmod b - y_{b+i} \bmod b) \bmod b$.

Замечание 16.3. Сложение по модулю степени двойки делается просто.

Итак, нам осталось научиться быстро вычислять ДПФ (и обратное к нему) и оценить время работы всего алгоритма в целом.