Лекция 1

Умножение матриц и его проверка. Обращение матриц. Сравнение строк на расстоянии и поиск подстроки.

(Конспект: Ф. Александров)

1.1 Умножение матриц

Будем учиться как можно быстрее перемножать квадратные матрицы с элементами из кольца. Пусть у нас есть две матрицы $\mathbf{A}$ и $\mathbf{B}$ размера $n \times n$. Обозначим за $\mathbf{C}$ их произведение:

$$\mathbf{A} * \mathbf{B} = \mathbf{C}. \tag{1.1}$$

1. Простой способ. Пусть $n = 2^k, k \in \mathbb{Z}$. Поделим каждую из матриц на 4 равные части. Например,

$$\mathbf{A} = \begin{pmatrix} \mathbf{A_{11}} & \mathbf{A_{12}} \\ \mathbf{A_{21}} & \mathbf{A_{22}} \end{pmatrix}. \tag{1.2}$$

Каждая из матриц разбиения будет иметь размерность $\frac{n}{2} \times \frac{n}{2}$. Сведем перемножение матриц размера $n \times n$ к перемножению матриц размера

$\frac{n}{2} \times \frac{n}{2}$:

$$\begin{aligned}
\mathbf{C}_{11} &= \mathbf{A}_{11}\mathbf{B}_{11} + \mathbf{A}_{12}\mathbf{B}_{21}, \\
\mathbf{C}_{12} &= \mathbf{A}_{11}\mathbf{B}_{12} + \mathbf{A}_{12}\mathbf{B}_{22}, \\
\mathbf{C}_{21} &= \mathbf{A}_{21}\mathbf{B}_{11} + \mathbf{A}_{22}\mathbf{B}_{21}, \\
\mathbf{C}_{22} &= \mathbf{A}_{21}\mathbf{B}_{12} + \mathbf{A}_{22}\mathbf{B}_{22}.
\end{aligned}$$

Далее каждую из матриц $\mathbf{C}_{ij}$ опять поделим на четыре равные части, и так далее, пока не сведем перемножение матриц к операциям перемножения элементов кольца.

Подсчитаем время работы $T(n)$ такого алгоритма. Здесь и далее за единицу времени примем время операции с элементом матрицы.

$$T(n) = 8T\left(\frac{n}{2}\right) + cn^2, \quad \text{где } c \text{ – некоторая константа.} \tag{1.3}$$

Теорема 1.1. *Пусть* $T(n) = 8T\left(\frac{n}{2}\right) + cn^2$, $n = 2^k$, $k \in \mathbb{Z}$. *Тогда* $T(n) = O(n^3)$.

Доказательство. $T(n) = 8T(\frac{n}{2}) + cn^2 = 8^2T(\frac{n}{4}) + 8c(\frac{n}{2})^2 + cn^2 = \cdots = cn^2 + 8c(\frac{n}{2})^2 + 8^2c(\frac{n}{2^2})^2 + \cdots + 8^{k-1}c(\frac{n}{2^{k-1}})^2 + 8^k = cn^2\left(1 + \frac{8}{2} + \frac{8^2}{2^2} + \cdots + \frac{8^{k-1}}{2^{2k-2}}\right) + 8^k = cn^2\left(\frac{2^k-1}{3}\right) = c'n^2(n-1) = O(n^3)$ $\square$

Из теоремы 1.1 следует, что время работы нашего алгоритма равно $O(n^3)$. То есть сам алгоритм не лучше алгоритма перемножения матриц "по определению", но этот же подход мы сейчас реализуем в алгоритме, трудоемкость которого будет уже меньше $O(n^3)$. Для этого нам потребуется чуть более общая теорема:

Теорема 1.2. *Пусть* $n = b^k$, $k \in \mathbb{Z}$, $T(n) = aT(\frac{n}{b}) + cn^2$, $a < b^2$. *Тогда* $T(n) = O(n^{log_b a})$.

Доказательство. Аналогично доказательству теоремы 1.1. $\square$

Упражнение 1.1. Докажите теорему 1.2 для произвольного n.

Упражнение 1.2. Докажите теорему 1.2 при $a = b^2$.

Упражнение 1.3. Докажите теорему 1.2 при $a > b^2$.

2. Способ похитрее (*Алгоритм Штрассена.* Опять рассмотрим такое же разбиение матриц и введем новые матрицы

$$\begin{aligned}
\mathbf{M}_1 &= (\mathbf{A}_{12} - \mathbf{A}_{22})(\mathbf{B}_{21} + \mathbf{B}_{22}),\\
\mathbf{M}_2 &= (\mathbf{A}_{11} + \mathbf{A}_{22})(\mathbf{B}_{11} + \mathbf{B}_{22}),\\
\mathbf{M}_3 &= (\mathbf{A}_{11} - \mathbf{A}_{21})(\mathbf{B}_{11} + \mathbf{B}_{12}),\\
\mathbf{M}_4 &= (\mathbf{A}_{11} + \mathbf{A}_{12})\mathbf{B}_{22},\\
\mathbf{M}_5 &= \mathbf{A}_{11}(\mathbf{B}_{12} - \mathbf{B}_{22}),\\
\mathbf{M}_6 &= \mathbf{A}_{22}(\mathbf{B}_{21} - \mathbf{B}_{11}),\\
\mathbf{M}_7 &= (\mathbf{A}_{21} + \mathbf{A}_{22})\mathbf{B}_{11}.
\end{aligned}$$

Тогда $\mathbf{C}_{ij}$ можно выразить через $\mathbf{M}_{kl}$:

$$\begin{aligned}
\mathbf{C}_{11} &= \mathbf{M}_1 + \mathbf{M}_2 - \mathbf{M}_4 + \mathbf{M}_6,\\
\mathbf{C}_{12} &= \mathbf{M}_4 + \mathbf{M}_5,\\
\mathbf{C}_{21} &= \mathbf{M}_6 + \mathbf{M}_7,\\
\mathbf{C}_{22} &= \mathbf{M}_2 - \mathbf{M}_3 + \mathbf{M}_5 - \mathbf{M}_7.
\end{aligned}$$

Пользуясь теоремой 1.2, находим время работы алгоритма: $T(n) = O(n^{log_2 7})$. Поскольку $log_2 7 \approx 2.80735$, этот алгоритм лучше предыдущего и стандартного алгоритма (через вычисление каждого элемента результирующей матрицы по определению умножения матриц).

Как можно проверить, что алгоритм действительно находит нам произведение матриц? Этот алгоритм прост, и убедиться в его правильности можно простой подстановкой. Далее мы научимся проверять произвольный алгоритм и даже программу, написанную на его основе, быстрее и лучше.

Упражнение 1.4. Где мы воспользовались принадлежностью кольцу элементов матриц?

1.2 Уножение булевых матриц

Мы не можем использовать наш быстрый алгоритм для перемножения булевых матриц, так как T (истина) и F (ложь) с операциями $\vee$ (дизъюнкция) и $\wedge$ (конъюнкция) не образуют кольца.

Пример 1.1. Пример перемножения булевых матриц, для которого не работает представленный быстрый алгоритм:

$$\begin{pmatrix} T & F \\ T & F \end{pmatrix} \wedge \begin{pmatrix} F & T \\ T & T \end{pmatrix} = \begin{pmatrix} F & T \\ F & T \end{pmatrix}.$$

Теорема 1.3. *Умножение булевых матриц можно выполнить за* $O(n^{\log 7})$ *арифметических операций над числами от* 0 *до* n.

Доказательство. Чтобы воспользоваться нашим быстрым алгоритмом, будем вместо булевых операций $\vee$ и $\wedge$ использовать операции сложения и умножения в кольце $\mathbb{Z}_{n+1}$, где n – размер матрицы. Легко показать, что элемент произведения, вычисленного таким образом, отличен от нуля тогда и только тогда, когда соответствующий элемент произведения булевых матриц истинен. $\square$

1.3 Проверка результата алгоритма перемножения матриц

Итак, мы знаем уже несколько алгоритмов перемножения матриц, но у нас нет хорошего способа проверки таких алгоритмов (и реализующих их программ). Рассмотрим вероятностный алгоритм, который даст нам возможность проверять результат быстрее, чем считать произведение.

Возьмем случайный вектор $\mathbf{r}$, составленный из случайных битов (принимают и 1, и 0 с равными вероятностями). У нас уже есть результат перемножения матриц $\mathbf{A}$ и $\mathbf{B}$ – матрица $\mathbf{C}$, полученная нами. Будем проверять равенство:

$$\mathbf{A} \times \mathbf{B} = \mathbf{C}. \tag{1.4}$$

Домножим обе части справа на случайный вектор $\mathbf{r}$. Теперь проверим новое равенство

$$\mathbf{AB} \times \mathbf{r} = \mathbf{C} \times \mathbf{r}. \tag{1.5}$$

На проверку его уйдет меньше времени, трудоемкость такой проверки равна $O(m * n)$, где n – длина вектора $\mathbf{r}$, m – количество строк матрицы $\mathbf{C}$. Если $\mathbf{C}$ квадратная, то трудоемкость выражается проще – $O(n^2)$. Вспомним, что трудоемкость при перемножении матриц была близка к $O(n^{2.8})$. Докажем, что этот алгоритм дествительно проверяет результат перемножения.

Теорема 1.4. $\forall$ *матриц* $\mathbf{A}, \mathbf{B}, \mathbf{C}$
a) $\mathbf{AB} = \mathbf{C} \Rightarrow$ *алгоритм проверки не ошибается,*
b) $\mathbf{AB} \neq \mathbf{C} \Rightarrow$ *алгоритм ошибается с вероятностью не более чем* $\frac{1}{2}$.

Доказательство. Пункт a) очевиден, рассмотрим пункт b).
Известно, что $(\mathbf{AB} - \mathbf{C})\mathbf{r} \neq \emptyset$. В каком случае алгоритм ошибется? Если

скажет, что $\mathbf{AB} = \mathbf{C}$, то есть если $(\mathbf{AB} - \mathbf{C})\mathbf{r} = \emptyset$. Возьмем строчку матрицы $\mathbf{X} = \mathbf{AB} - \mathbf{C}$, не равную $\emptyset$ (помним, что сейчас у нас $\mathbf{AB} \neq \mathbf{C}$). Пусть x_{kl} – ненулевой элемент этой строчки. Тогда произведение k-ой строки на $\mathbf{r}$ выглядит так:

$$\sum_{i\in\{1,2,\dots,\widehat{l},\dots\}} x_{ki}r_i + x_{kl}r_l = 0, \quad \text{где } x_{kl} \neq 0. \tag{1.6}$$

Обозначим

$$c := -\frac{1}{x_{kl}} \sum_{i\in\{1,2,\dots,\widehat{l},\dots\}} x_{ki}r_i. \tag{1.7}$$

С какой вероятностью $r_l = c$? С вероятность выбрать бит r_l равным биту c, то есть с вероятностью $\frac{1}{2}$. Следовательно, алгоритм ошибается с вероятностью не более $\frac{1}{2}$. $\square$

Такой метод проверки называется *fingerprinting*.

Итак, наш алгоритм правильно решает задачу (т.е. говорит, что данная ему программа верно вычисляет произведение $\mathbf{A}$ и $\mathbf{B}$), если $\mathbf{AB} = \mathbf{C}$, и ошибается (говорит “верно”, хотя на самом деле “неверно”) с вероятностью не более $\frac{1}{2}$, если $\mathbf{AB} \neq \mathbf{C}$. Алгоритмы такого типа называются вероятностными алгоритмами с *односторонней ограниченной вероятностью ошибки (one-sided bounded error)*. Какова реальная польза от такого алгоритма? Ведь вероятность ошибки очень велика. Но если этот алгоритм применить 10 раз, то вероятность ошибки станет $\left(\frac{1}{2}\right)^{10}$, а это уже менее 0.001.

1.4 Обращение матриц

Конечно же, мы хотим применить наш (и любой другой) быстрый алгоритм перемножения матриц для обращения матрицы. (Естественно, над произвольным кольцом обратить матрицу не выйдет, ибо понадобится обратный элемент.)

1. Обращение треугольных матриц. Рассмотрим квадратную матрицу $\mathbf{A}$:

$$\mathbf{A} = \begin{pmatrix} \mathbf{A_{11}} & \emptyset \\ \mathbf{A_{21}} & \mathbf{A_{22}} \end{pmatrix}, \quad \text{где } \mathbf{A_{21}} \text{ – квадратная матрица,}$$
$$\text{а } \mathbf{A_{11}}, \mathbf{A_{22}} \text{ – квадратные треугольные.}$$

Тогда обратной к матрице $\mathbf{A}$ будет матрица такого вида

$$\mathbf{A}^{-1} = \begin{pmatrix} \mathbf{A}_{11}^{-1} & \emptyset \\ -\mathbf{A}_{22}^{-1}\mathbf{A}_{21}\mathbf{A}_{11}^{-1} & \mathbf{A}_{22}^{-1} \end{pmatrix} \tag{1.8}$$

Как видно, получили рекурсивный алгоритм обращения треугольных матриц.

Упражнение 1.5. Посчитать трудоемкость алгоритма, зная трудоемкость $M(n) = O(n^{2+\varepsilon})$ умножения $n \times n$-матриц (для некоторого $\varepsilon > 0$).

2. Обращение произвольных матриц.

Теорема 1.5. $\forall \mathbf{A}$ – *невырожденная матрица – раскладывается в произведение*

$$\mathbf{A} = \mathbf{LUP}, \quad \text{где} \quad \begin{aligned} \mathbf{L} &\text{ – нижняя треугольная матрица,} \\ \mathbf{U} &\text{ – верхняя треугольная матрица,} \\ \mathbf{P} &\text{ – матрица перестановок.} \end{aligned}$$

Причем раскладывается достаточно быстро.

Доказательство. Докажем, предъявив алгоритм. Пусть $\mathbf{A}$ – $m \times p$ матрица, для которой хотим найти обратную. Алгоритм наш будет рекурсивным, с уменьшением порядка матрицы на каждом шаге рекурсии. Рекурсия остановится на тривиальном случае.

В алгоритмическом плане это выглядит так, будто мы умеем находить искомое разложение для матриц меньшего, чем у $\mathbf{A}$, размера и хотим свести задачу к разложениям таких матриц.

Такой (рекурсивный) алгоритм разложения матрицы $\mathbf{A}$ размером $m \times p$ назовем $factor(\mathbf{A}, \mathbf{m}, \mathbf{p})$. Он возвращает три необходимые матрицы $\mathbf{L}, \mathbf{U}, \mathbf{P}$. Предъявим шаг рекурсии для размера $m \times p$. Разделим $\mathbf{A}$ на две матрицы:

$$\mathbf{A} = \begin{pmatrix} \mathbf{B} \\ \mathbf{C} \end{pmatrix}$$

где $\mathbf{B}$ и $\mathbf{C}$ – матрицы размера $m/2 \times p$.

Матрицы $\mathbf{L_1}, \mathbf{U_1}, \mathbf{P_1}$ – результат $factor(\mathbf{B}, m/2, p)$ (то есть $\mathbf{B} = \mathbf{L_1 U_1 P_1}$).

Введем новые матрицы:

$$\begin{aligned} \mathbf{D} &= \mathbf{CP_1^{-1}} \\ \mathbf{G} &= \mathbf{D} - \mathbf{FE^{-1}U_1} \\ \mathbf{H} &= \mathbf{U_1P_3^{-1}} \end{aligned}$$

Пусть $\mathbf{G'}$ – матрица, образованная $p - \frac{m}{2}$ последними столбцами матрицы $\mathbf{G}$. Получим $\mathbf{L_2}, \mathbf{U_2}, \mathbf{P_2}$, как результат $factor(\mathbf{G'}, \frac{m}{2}, p - \frac{m}{2})$. Далее, за $\mathbf{E}$ обозначим первые $m/2$ столбцов матрицы $\mathbf{U_1}$, а за $\mathbf{F}$ – первые $m/2$ столбцов матрицы $\mathbf{D}$, За $\mathbf{P_3}$ примем матрицу, у которой нижним правым блоком является $\mathbf{P_2}$, верхним левым – единичная размера $m/2 \times m/2$, дополненную также нулями.

Теперь получим наши искомые матрицы $\mathbf{L}, \mathbf{U}, \mathbf{P}$:

Матрицу $\mathbf{L}$ составим из матриц $\mathbf{L_1}$ (верхний левый блок), $\mathbf{FE^{-1}}$ (нижний левый), $\mathbf{L_2}$ (нижний правый). Оставшимися элементами запишем нули.

$\mathbf{U}$ построим с помощью $\mathbf{H}, \mathbf{U_2}$: верхняя ее часть – это H, нижняя же состоит из нулевой матрицы слева и U_2 справа.

$\mathbf{P} = \mathbf{P_3 P_1}$. □

Очевидно, что для $\mathbf{A} = \mathbf{LUP}$ $\mathbf{A^{-1}} = \mathbf{P^{-1}U^{-1}L^{-1}}$. Искать обратные для треугольных матриц $\mathbf{L}, \mathbf{U}$ уже умеем, а для перестановочной матрицы $\mathbf{P}$ обратной будет $\mathbf{P}^{\mathrm{T}}$.

Упражнение 1.6. Посчитайте трудоемкость описанного выше алгоритма разложения матрицы в терминах упражнения 1.5.

Теорема 1.6. *В условиях упражнения 1.5 можно выполнить обращение произвольной невырожденной $n \times n$ матрицы за время $M(n)$.*

Упражнение 1.7. Доказать теорему 1.6. *Указание:* воспользоваться упражнениями 1.5 и 1.6; обращение матрицы перестановки можно быстро реализовать, представив ее в виде одномерного массива.

1.5 Метод fingerprinting в применении к задачам со строками

1. Сравнение на равенство двух строк. Есть две строки a, b (можно считать их битовыми), которые необходимо сравнить на совпадение, затратив как можно меньше информации на это сравнение. Например, надо сравнить файлы по сети.

Идея алгоритма – сравнивать не сами строки, а функции от них. Пусть длина $\sharp a$ строки a составляет n бит.

Пусть $p \in \mathbb{P}$, $\mathbb{P}$ – множество простых чисел. В качестве функции-хэша возьмем $\mod p$. То есть будем сравнивать уже

$$a \mod p \quad \text{и} \quad b \mod p \tag{1.9}$$

Для такого сравнения достаточно передать $\log p$ битов (здесь и далее по-умолчанию берется двоичный логарифм, log_2) и еще столько же битов понадобится для передачи числа p.

Будем брать случайное простое число p из интервала $[2..\tau]$ для некоторого τ, которое определим позже. Плохими p для нас будут такие, которые будут давать равенство в (1.9) при неравенстве исходных строк a, b. Количество таких чисел равно

$$\sharp\{p \in P : (a-b) \vdots p\} \tag{1.10}$$

Задача 1.1. Как выбрать простое число из заданного интервала случайным образом с равномерным распределением?

Лемма 1.1. *$c \leq 2^n \Rightarrow c$ имеет не более n различных простых делителей.*

Доказательство. Очевидно. □

Пусть $\tau = n^2 \log n^2$. Тогда вероятность ошибки при сравнении остатков $\mod p$ можно оценить

$$P_{\text{ошибки}} \leq \frac{n}{\frac{\tau}{\log \tau}} = \frac{n(\log n^2 + \log\log n^2)}{n^2 \log n^2} = O\left(\frac{1}{n}\right). \tag{1.11}$$

Таким образом, привели вероятностный алгоритм с односторонней ошибкой с вероятностью ошибки $O(\frac{1}{n})$, требующий передачи всего $O(\log n)$ битов.

Определение 1.1. *Расстоянием* между двумя строками a, b будем считать количество несовпадающих у них битов.

Задача 1.2. Придумать алгоритм для нахождения расстояния d между двумя строками a, b длины n. Посчитать количество передаваемых битов, как функцию $T(n, d)$.

Определение 1.2. Под *editing distance* между a, b будем понимать минимальное количество операций редактирования, необходимых для преобразования строки a в строку b. Операциями редактирования считаем:

1. вставку бита
2. замену бита
3. уничтожение бита
4. перемещение сплошного блока битов

Задача 1.3. Придумать алгоритм для нахождения editing distance между двумя строками a, b длины n. Посчитать трудоемкость $T(n, d)$.

2. Поиск вхождения строки a в строку b. Займемся теперь такой задачей. Нужно определить, входит ли строка a в строку b. Длины строк a и b равны, соответственно, m и n. Пусть $m \leq n$. Ясно, что решая ее в лоб, получим сложность почти $O(mn)$.

Предъявим вероятностный алгоритм с линейной сложностью $O(m+n)$.

Замечание 1.1. Существует детерминированный алгоритм поиска подстроки в строке за время $O(m+n)$, но он гораздо сложнее.

Определим

$$b(i) = b_i b_{i+1}..b_{i+m-1}. \tag{1.12}$$

Необходимо провести $n-m+1$ сравнений строк на равенство, а это мы уже умеем делать: будем сравнивать

$$(a \mod p) \quad \text{и} \quad (b(i) \mod p). \tag{1.13}$$

Но можно упростить задачу, вычисляя $(b(i) \mod p)$ через $(b(i-1) \mod p)$.

$$\begin{aligned} b(i) &= b_i + 2b_{i+1} + .. + 2^{m-1}b_{i+m-1} \\ b(i-1) &= 2b_i + 2^2 b_{i+1} + .. + 2^{m-1}b_{i+m-2} + b_{i-1} \\ \Rightarrow b(i) &= \frac{b(i-1) - b_{i-1}}{2} + 2^{m-1}b_{i+m-1} \end{aligned}$$

Опять будем брать простое число $p \in [2, \tau]$. $\tau = n^2 m \log n^2 m$,

$$P_{\text{ошибки}} \leq \frac{m}{\frac{\tau}{\log \tau}} \leq \frac{2m \log n^2 m}{n^2 m \log n^2 m} = O\left(\frac{1}{n^2}\right) \tag{1.14}$$

Можно вообще избавить этот алгоритм от необходимости ошибаться. Если

$$a \mod p = b(i) \mod p,$$

то честно проверим равенство $a = b$. Этот алгоритм уже не ошибается. Какова его сложность? Математическое ожидание времени его работы $T(n, m)$ легко посчитать:

$$\mathbf{E}T(n, m) \leq mn * \frac{1}{n} + (m+n)(1 - \frac{1}{n}) = O(m+n). \tag{1.15}$$

То есть предъявили искомый линейно-быстрый алгоритм поиска подстроки в строке, причем это *вероятностный алгоритм с нулевой ошибкой.*

Упражнение 1.8. Исследовать работу (вероятность ошибки и сложность) такого алгоритма: если $(a - b(i)) \vdots p$, то выбираем новое простое p' и меняем p на p'.

Задача 1.4. Обобщить алгоритмы на случай двумерного пространства. То есть реализовать поиск блока в матрице.