Лекция 6

Сложность рекурсивных алгоритмов. Умножение матриц (над кольцом и булевых). Простой рекурсивный алгоритм для умножения целых чисел. Нахождение пары ближайших точек на плоскости

6.1 Сложность рекурсивных алгоритмов

Предположим, что алгоритм действует по схеме «разделяй и властвуй», т.е. сводит задачу к нескольким таким же задачам меньшего размера и решает их. Тогда время его работы можно оценить при помощи следующей теоремы (аналогично можно оценить и занимаемую память).

Теорема 6.1. *Пусть функция T задана соотношениями*

$$T(1) = 1,$$

$$T(n) \leq aT(\lceil n/c \rceil) + bn^d \text{ при } n>1,$$

где $a, b, c, d \geq 0$ — константы. Тогда

- *если $a < c^d$, то $T(n) = O(n^d)$;*
- *если $a = c^d$, то $T(n) = O(n^d \log n)$;*
- *если $a > c^d$, то $T(n) = O(n^{\log_c a})$.*

Замечание 6.1. При использовании этой теоремы n может быть любым параметром задачи, а не только размером входа.

Доказательство. Оценим $T(N)$ для N вида c^k; результат для n, не являющихся степенями c, будет простым следствием.

Раскрыв рекуррентное соотношение, получим

$$\begin{aligned} T(N) &\le bN^d + aT(N/c) \le bN^d + ab(N/c)^d + T(N/c^2) \le \dots \\ &\le bN^d \sum_{i=0}^{k} \left(\frac{a}{c^d}\right)^i . \end{aligned} \tag{6.1}$$

В случае $a < c^d$ эта $\sum_{i=0}^{k}$ ограничена $\sum_{i=0}^{+\infty}$, а та, в свою очередь, константой. В случае $a = c^d$ имеем сумму из $k = \log_c N$ единиц. Если же $a > c^d$, вычислим сумму как сумму геометрической прогрессии.

Наконец, покажем, что теорема верна и для произвольного n. Пусть $N = c^{\lceil \log_c n \rceil}$, для таких N мы теорему уже доказали. Тогда

$$T(n) \le^{?} T_*(N) = O(T_*(n)),$$

где T_* — наша оценка (с конкретной константой вместо $O(\dots)$). Неравенство $\le^{?}$ выполняется, так как каждый член более громоздко выглядящей суммы

$$T(n) \le bn^d + ab\lceil n/c \rceil^d + a^2 b(\lceil \lceil n/c \rceil /c \rceil)^d + \dots$$

мажорируется соответствующим членом оцененной нами суммы (6.1), которая в «раскрытом» виде выглядит как

$$T(N) \le bN^d + ab(N/c)^d + a^2 b(N/c^2)^d + \dots \ .$$

□

6.2 Умножение матриц

Задача: вычислить произведение $\mathbf{C}$ матриц $\mathbf{A}$ и $\mathbf{B}$ размера $n \times n$ над произвольным кольцом, используя лишь операции кольца. Будем подсчитывать количество этих операций. Как обычно, n можно считать степенью двойки (говоря формально, можно перемножить чуть бо́льшие матрицы; при написании реальной программы это, конечно, необязательно).

Очевидный способ. Поделим эти матрицы на четыре части, пополам по вертикали и горизонтали: например, $\mathbf{A} = \begin{pmatrix} \mathbf{A_{11}} & \mathbf{A_{12}} \\ \mathbf{A_{21}} & \mathbf{A_{22}} \end{pmatrix}$. 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 $\mathbf{A}_{ij}$, $\mathbf{B}_{ij}$ опять поделим на четыре равные части, и так далее, пока не сведем перемножение матриц к операциям перемножения элементов кольца.

Подсчитаем количество $T(n)$ операций с элементами матриц, выполняемых таким алгоритмом:

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

По теореме 6.1, $T(n) = O(n^3)$.

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 $\mathbf{M}_{kl}$:

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

Подсчитаем количество $T(n)$ операций с элементами матриц, выполняемых таким алгоритмом:

$$T(n) = 7T\left(\frac{n}{2}\right) + cn^2, \quad \text{где } c \text{ — некоторая константа.}$$

По теореме 6.1, $T(n) = O(n^{\log_2 7})$. Поскольку $\log_2 7 \approx 2.80735$, этот алгоритм лучше предыдущего и лучше тривиального алгоритма (через вычисление каждого элемента матрицы C по определению произведения матриц).

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

Упражнение 6.1. Где мы воспользовались принадлежностью *кольцу* элементов матриц?

Замечание 6.2. К умножению можно свести и обращение матриц (конечно, невырожденных и, к тому же, над полем). Для этого понадобится разложить матрицу в произведение матриц специального вида (нижнетреугольную, верхнетреугольную и матрицу перестановки). Если кому-то понадобится реализовать этот алгоритм, можно прочесть в книге Ахо, Хопкрофта и Ульмана или Кормена, Лейзерсона и Ривеста.

Упражнение 6.2. Мы выяснили количество операций над элементами кольца. За какое время можно перемножить на РАМ матрицы, состоящие из целых чисел? (Подсказка: за какое время Вы можете перемножить целые числа на РАМ?)

6.3 Умножение булевых матриц

Произведение (конъюнкция) булевых матриц (их элементами могут быть T (истина) и F (ложь)) определяется точно так же, как и произведение обычных матриц, но в качестве умножения элементов выступает конъюнкция $\wedge$, а в качестве сложения — дизъюнкция $\vee$. Мы не можем использовать наш быстрый алгоритм для перемножения булевых матриц, так как T и F с операциями $\vee$ и $\wedge$ не образуют кольца.

Пример 6.1. Пример перемножения булевых матриц:

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

Теорема 6.2. *Умножение булевых матриц можно выполнить за $O(n^{\log 7})$ арифметических операций по модулю $n+1$.*

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. □

Задача 6.1. Сформулируйте и докажите верхнюю оценку *времени* работы этого алгоритма на РАМ. Обратите внимание, что Вам предстоит производить операции по модулю $n+1$.

6.4 Простой рекурсивный алгоритм для умножения чисел

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 $O(n)$.

Начнем с простого алгоритма, время работы которого составляет $O(n^{\log_2 3})$. В дальнейшем мы изучим алгоритм, который умножает числа за время $O(n \cdot \log n \cdot \log\log n)$ (он, кстати, будет использовать наш простой рекурсивный алгоритм как подпрограмму).

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:

$$a_1 b_2 + a_2 b_1 = (a_1 + a_2)(b_1 + b_2) - a_1 b_1 - a_2 b_2.$$

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 —

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

Тогда по теореме 6.1 $T(n) = O(n^{\log_2 3})$.

Замечание 6.3. 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.

6.5 Нахождение пары ближайших точек на плоскости

Задача: на плоскости заданы координаты $n \geq 2$ точек (x_i, y_i). Найти две различные точки из числа заданных, находящиеся на минимально возможном расстоянии (и определить это расстояние).

Решение: построим рекурсивный алгоритм. Нам понадобится два упорядоченных двунаправленных списка номеров наших точек: список X будет упорядочен по возрастанию первой координаты, список Y — по возрастанию второй координаты. Будет также полезно, если в элементах первого списка будут храниться ссылки на соответствующие тем же точкам места второго списка.

Разделим наше множество точек на два приблизительно равных по мощности: первые $\lceil n/2 \rceil$ элементов списка X и оставшиеся. (Сделать это, используя наши списки, просто — получатся такие же пары списков, только в два раза короче.) Назовем эти множества S_1 и S_2; имеется значение x_0 первой координаты, которое разделяет элементы этих множеств. Рекурсивно применим наш алгоритм к S_1 и к S_2 — тем самым, найдем ближайшие пары точек для каждого из этих множеств.

Пусть наименьшее из полученных расстояний — δ. Для завершения вычислений нам остается проверить случай, когда ближайшая пара состоит из одной точки множества S_1 и одной точки множества S_2. Если это так, расстояние между ними менее

δ, а значит, обе они находятся в вертикальной полосе с координатами от $x_0 - \delta$ до $x_0 + \delta$ (множество таких точек легко выделить при помощи списка X).

Проверим расстояния от каждой из полученных точек до следующих семи точек в списке Y. Заметим, что этого достаточно: искомая пара точек находится внутри прямоугольника высоты δ, выделенного из нашей вертикальной полосы. Этот прямоугольник состоит из двух квадратов со стороной δ, в каждом из них может быть не более четырех точек, иначе в соответствующем множестве S_i были бы точки, расстояние между которыми было бы меньше δ (разделим этот квадрат на четыре одинаковых квадратика — в каждом из них может быть только одна точка).

Рекуррентное неравенство для количества операций, совершаемых рекурсивной частью нашего алгоритма, очевидно, $T(n) \leq 2T(\lceil n/2 \rceil) + O(n)$. По теореме 6.1, $T(n) = O(n \log n)$, и столько же операций используется на построение исходных списков (поскольку их надо отсортировать: это можно сделать, *например*, построив B^+-дерево[1]). (Заметим, что на перебор всех пар точек понадобилось бы $\Omega(n^2)$ операций.)

[1] В дальнейшем мы изучим и более разумные способы сортировки.