Лекция 7

Рисование планарного графа. Сложность рекурсивных алгоритмов. Умножение матриц (над кольцом и булевых). Нахождение пары ближайших точек на плоскости.

7.1 Рисование планарного графа

Планарным называется граф, который можно нарисовать на плоскости без самопересечений. Будем рассматривать неориентированные графы.

Ниже мы представим алгоритм, который работает с двусвязным графом, то есть с графом, в котором не существует вершины, удаление которой ведет к потере связности. (Заметим, что если граф не двусвязен, то в нем имеется некоторая вершина v, удаление которой приводит к разбиению графа на (V_1, E_1) и (V_2, E_2); тогда можно нарисовать подграфы, порожденные[1] $V_1 \cup \{v\}$ и $V_2 \cup \{v\}$, по отдельности и соединить полученные рисунки; чтобы отправить точку соединения на границу области, занимаемой графом, рисуем граф на сфере, после чего «раскрываем» сферу, проделав дырку рядом с вершиной.) Алгоритм будет рисовать граф на плоскости, если граф — планарный.

Определение 7.1. Пусть дан граф $G = (V, E)$, из которого уже на-

[1]Подграфом графа (V, E), *порожденным* подмножеством вершин $V' \subseteq V$, называется граф (V', E'), где $E' = \{\{x, y\} \in E \mid x \in V' \wedge y \in V'\}$.

рисовано некоторое подмножество вершин $W \subseteq V$. В этом разделе окрестностью множества вершин $S \subseteq V \setminus W$ будем называть множество $\Gamma(S) = \{v \in W \mid \exists e \in E : e = \{v, s\}, s \in S\}$.

На каждом шаге работы нашего алгоритма плоскость будет разбита уже нарисованными частями графа на *клетки* (таким образом, каждая из клеток K будет ограничена нарисованной *границей* ∂K — некоторым циклом исходного графа). Сам же граф будет уменьшаться и распадаться на компоненты связности (мы будем выкидывать уже нарисованные вершины). Очевидно, одна компонента может быть нарисована только целиком в одной клетке. Компонента *совместна* с клеткой, если все нарисованные вершины, бывшие в ее окрестности в исходном графе, принадлежат границе этой клетки.

Алгоритм 7.1.

1. Взять какой-нибудь цикл, нарисовать его и выкинуть из графа (у нас получилось две клетки).
2. Если в оставшемся графе существует компонента, совместная лишь с одной клеткой, то взять путь в этой компоненте, соединяющий (вместе с двумя соответствующими ребрами исходного графа) две вершины границы этой клетки, и нарисовать его в клетке (удалив путь из компоненты и разбив компоненту на несколько, если она развалилась).
3. Если каждая компонента согласована с несколькими клетками, то взять любую компоненту и вставить путь в клетку (удалив путь и т. д.) аналогично шагу 2.
4. Если еще не весь граф нарисован, вернуться к шагу 2.

$\square$

Заметим, что шаги 1 и 2 алгоритма являются вынужденными. Поэтому его корректность вытекает из следующей леммы.

Лемма 7.1. *Пусть в какой-то момент что-то уже нарисовано и алгоритм находится в шаге 3. Компонента C согласована с клетками K_1 и K_2. Если C можно вложить в K_1 (и успешно дорисовать граф до конца), то ее можно вложить и в K_2 (и успешно дорисовать).*

Доказательство. Рассмотрим правильный рисунок (всего графа), в котором C вложена в K_1. Построим правильный рисунок (всего графа), в котором C вложена в K_2. В дальнейшем под клетками и компонентами понимаются те клетки и компоненты, которые имелись в рассматриваемый момент времени.

Поменяем местами все компоненты, согласованные и с K_1, и с K_2 (назовем такие компоненты *активными*): те, что в исходном рисунке были нарисованы внутри K_1, отправим в K_2, и наоборот. Покажем, что их по-прежнему можно нарисовать без самопересечений.

Действительно, две активные компоненты всегда можно «развести» на плоскости: ведь прежде они были разведены. Так что конфликт может возникнуть только между активной и неактивной компонентами. Перебором случаев проверим, что таких конфликтов также не должно возникнуть: разобьем границы клеток K_1 и K_2 на участки, принадлежащие только K_1, только K_2, либо им вместе; разберем случаи, когда конфликтующий путь начинается на одном из участков, а заканчивается на другом.

Если окрестность неактивной компоненты целиком содержится в пути (в нарисованной части графа), внутренние (не первая и не последняя) вершины которого содержатся в $\partial K_i \setminus \partial K_{3-i}$, то конфликта возникнуть не может (эти внутренние вершины не могут входить в окрестность активной компоненты — ведь она согласована с обеими клетками!). Если же окрестность не содержится в таком пути, легко видеть (нарисуйте!), что неактивная компонента в рассматриваемый момент времени была согласована только с одной клеткой, а это противоречит условию шага 3. □

Упражнение 7.1. Важное упражнение на понимание: найдите, где используется двусвязность графа. □

Замечание 7.1. Существует алгоритм, позволяющий нарисовать планарный граф отрезками прямых за линейное число операций.

7.2 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 7.1. *Пусть функция T задана соотношениями*

$$T(1) = 1,$$

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

где $a, b, c, d \ge 0$ — 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 7.2. При использовании этой леммы n может быть любым параметром задачи, а не только размером входа.

Доказательство. Оценим $T(n)$ для n вида c^k; результат для других n будет простым следствием.

Раскрыв рекуррентное соотношение, получим

$$T(n) \leq bn^d + aT(n/c) \leq bn^d + ab(n/c)^d + T(n/c^2) \leq \ldots \leq bn^d \sum_{i=0}^{k} \left(\frac{a}{c^d}\right)^i.$$

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, для произвольного n

$$T(n) \leq T_*(c^{\lceil \log_c n \rceil}) = O(T_*(n)),$$

где T_* — наша оценка (с конкретной константой вместо $O(\ldots)$). □

7.3 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.

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 7.1, $T(n) = O(n^3)$.

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 7.1, $T(n) = O(n^{\log_2 7})$. 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 и даже программу, написанную на его основе, быстрее и лучше.

Упражнение 7.2. Где мы воспользовались принадлежностью *кольцу* элементов матриц?

Замечание 7.3. 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.

7.4 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 7.1. 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 7.2. *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. □

7.5 Проверка результата алгоритма умножения матриц

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

Возьмем случайный вектор $\mathbf{r}$, т.е. вектор, составленный из битов, принимающих значения 0 или 1 с вероятностью $\frac{1}{2}$ независимо друг от друга. У нас уже есть результат перемножения $n \times n$ матриц $\mathbf{A}$ и $\mathbf{B}$ — матрица $\mathbf{C}$, полученная при помощи алгоритма, в правильности которого мы не уверены. Будем проверять равенство

$$\mathbf{A} \cdot \mathbf{B} = \mathbf{C}. \tag{7.1}$$

Домножим обе части справа на случайный вектор $\mathbf{r}$. Вместо (7.1) проверим новое равенство

$$(\mathbf{AB}) \cdot \mathbf{r} = \mathbf{C} \cdot \mathbf{r}$$

и выдадим ответ, соответствующий результату этой проверки. На такую проверку уйдет лишь $O(n^2)$ операций с элементами матриц — это меньше, чем в алгоритме Штрассена (и в любом другом известном алгоритме для *умножения* матриц). Докажем, что этот алгоритм действительно проверяет результат перемножения.

Теорема 7.3. $\forall$ *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* $\frac{1}{2}$.

Доказательство. Пункт a) очевиден, рассмотрим пункт b).
Известно, что $(\mathbf{AB} - \mathbf{C})\mathbf{r} \neq \mathbf{0}$. В каком случае алгоритм ошибется? Если скажет, что $\mathbf{AB} = \mathbf{C}$, то есть если $(\mathbf{AB} - \mathbf{C})\mathbf{r} = \mathbf{0}$. Возьмем строчку матрицы $\mathbf{X} = \mathbf{AB} - \mathbf{C}$, не равную $\mathbf{0}$ (она есть, поскольку $\mathbf{AB} \neq \mathbf{C}$). Пусть x_{kl} — ненулевой элемент этой строчки. Тогда произведение k-ой строки на $\mathbf{r}$ выглядит так:

$$\sum_{i \in \{1,2,\ldots,\widehat{l},\ldots\}} x_{ki} r_i + x_{kl} r_l = 0, \quad \text{где } x_{kl} \neq 0. \tag{7.2}$$

Обозначим

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

С какой вероятностью $r_l = c$? С вероятностью 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 *методом отпечатков пальцев* (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? На первый взгляд, вероятность ошибки велика. Но если этот алгоритм повторить 100 раз (100 — это лишь константа!), то вероятность ошибки станет $\frac{1}{2^{100}}$, а это уже меньше вероятности отказа вычислительной техники.

Замечание 7.4. Мы не пользовались тем, что матрицы — квадратные.

7.6 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 нашим алгоритмом, очевидно, $T(n) \leq 2T(\lceil n/2 \rceil) + O(n)$. По теореме 7.1, $T(n) = O(n \log n)$, и столько же операций используется на построение исходных списков (поскольку их надо отсортировать). (Заметим, что на перебор всех пар точек понадобилось бы $\Omega(n^2)$ операций.)