Лекция 6

Минимальное сечение, минимальное остовное дерево, детерминированный поиск подстроки

(Конспект: А. Бережной)

6.1 Минимальное сечение (MIN-CUT)

Пускай нужно построить минимальное сечение графа с n вершинами и m ребрами (о том, что такое сечение, см. лекцию 5 в первой части курса). Будем считать, что все веса в графе равны единице.

Поступаем так: берем случайное ребро в графе и стягиваем две вершины в одну, получая граф с кратными ребрами (но без циклов). После некоторого количества таких операций у нас останется две вершины; ребра между ними соответствуют какому-то (быть может, не минимальному) сечению в исходном графе. Посчитаем вероятность ошибки, т.е. того, что сечение – не минимально. Пусть k — это вес минимального сечения (зафиксируем конкретное минимальное сечение M); тогда степень любой вершины не превосходит k. Ошибка возникнет, если на каком-то шаге мы стянем в точку ребро из M. Пусть p — вероятность этого (на каждом конкретном шаге),

$$p \leq \frac{k}{\frac{nk}{2}} \leq \frac{2}{n}.$$

Тогда

$$P\{\text{успеха}\} \geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\dots\left(1 - \frac{2}{3}\right) = \frac{2(n-2)!}{n!} = \frac{2}{n(n-1)}$$

Это плохо, т.к. придется повторять процедуру $O(n^2)$ раз, чтобы получить вероятность ошибки, ограниченную константой. Вместо этого будем производить стягивание ребер только до тех пор, пока не останется $\lceil \frac{n}{\sqrt{2}} + 1 \rceil$ вершин. Сделаем для данного графа H это дважды; так мы получим два графа: H_1 и H_2. Применим этот алгоритм рекурсивно к обоим и вернем то сечение, которое окажется меньше. База рекурсии тривиальна (граф, состоящий из двух вершин).

Пусть t — остающаяся глубина рекурсии. Пусть $P_i^{(t)}$ — вероятность того, что ни одно ребро из выбранного нами выше минимального сечения M не было стянуто при переходе от графа H (полученного при данном рекурсивном вызове) к графу H_i (передаваемому далее по рекурсии),

$$P_i^{(t)} \geq \left(1 - \frac{2}{n}\right)\dots\left(1 - \frac{2}{\lceil \frac{n}{\sqrt{2}} + 2 \rceil}\right) = \frac{2}{n(n-1)} \frac{\lceil \frac{n}{\sqrt{2}} + 1 \rceil \lceil \frac{n}{\sqrt{2}} \rceil}{2} \geq \frac{1}{2}.$$

Пусть $P^{(t)}$ — вероятность того, что, получив на вход мультиграф, в котором сечение M еще есть, алгоритм вернет именно это сечение (при условии, что будет достаточно рекурсивных вызовов глубины t),

$$P^{(t)} \geq 1 - \prod_{i=1,2}\left(1 - P_i^{(t-1)} P^{(t-1)}\right) \geq 1 - \left(1 - \frac{1}{2}P^{(t-1)}\right)^2 = P^{(t-1)} - \frac{(P^{(t-1)})^2}{4}.$$

Возьмем q_t так, что $P^{(t)} = \frac{4}{q_t+1}$. Тогда

$$\frac{4}{q_t + 1} = \frac{4}{q_{t-1} + 1} - \frac{4}{(q_{t-1} + 1)^2} = \frac{4q_{t-1}}{(q_{t-1} + 1)^2},$$

$$q_t = q_{t-1} + 1 + \frac{1}{q_{t-1}}.$$

Далее, так как $P^{(0)} = 1$, то $q_0 = 3$. Отсюда $t < q_t < 3 + t + H_t$, где $H_t = 1 + \frac{1}{2} + \ldots + \frac{1}{t}$. Значит, $P^{(t)} = \Omega(\frac{1}{t})$; поскольку глубина рекурсии $O(\log n)$, имеем $P^{(t)} = \Omega(\frac{1}{\log n})$, т.е. достаточно $O(\log n)$ повторов для того, чтобы вероятность ошибки стала ограничена константой.

Теперь оценим трудоемкость алгоритма. На стягивание ребра тратится время $O(n^2)$. Оно повторяется $O(\log^2 n)$ раз (один логарифм от рекурсии, а другой — чтобы получить вероятность ошибки, ограниченную константой). Итого: время работы алгоритма составляет $O(n^2 \log^2 n)$.

6.2 Минимальное остовное дерево

Остовное дерево — это дерево, состоящее из некоторых ребер графа и содержащее все его вершины. Требуется построить остовное дерево с минимальным суммарным весом ребер. (Считаем, что все ребра разного веса: этого можно добиться, если к равным ребрам добавить достаточно малые ε_i.)

Сперва рассмотрим простой детерминированный алгоритм: **алгоритм Borůvka**. Выберем у каждой вершины ребро наименьшего веса. После этого в полученном графе каждую компоненту связности стянем в вершину. При каждой такой итерации число вершин уменьшается вдвое. Так что сложность этого алгоритма $O(m \log n)$. Фактически, этот алгоритм мало чем отличается от алгоритма Краскала, но его можно улучшить следующим образом:

Алгоритм 6.1. По графу G_1 с n вершинами и m ребрами строим минимальный остовный лес (ибо граф может быть несвязен) для него.

1. Применим к G_1 3 шага алгоритма Borůvka - получим граф G_2 и некое множество ребер S, которые принадлежат остовному дереву. В графе G_2 максимум $\frac{n}{8}$ вершин.

2. Из G_2 выкинем примерно половину ребер (именно, каждое ребро выкинем с вероятностью $\frac{1}{2}$) — это будет граф $G_2(\frac{1}{2})$ (в нем $\leq \frac{n}{8}$ вершин, а мат. ожидание количества ребер $\leq \frac{m}{2}$.

3. Для $G_2(\frac{1}{2})$ применяем алгоритм рекурсивно; получим F — минимальный остовный лес для этого графа.

   Далее требуется определение. Пусть в графе H есть остовное дерево (или лес) T. Ребро, соединяющее вершины v и w называется **тяжелым в H относительно T**, если ее вес больше веса максимального ребра на пути из v в w в T. Иначе ребро называется **легким**. Понятно, что тяжелые ребра нас не интересуют.

   Задача 6.1. Найти все легкие ребра за линейное время.

4. Пусть V_2 — множество ребер G_2, легких относительно F. Возьмем только их — получим граф G_3.

5. Рекурсивно обрабатываем G_3 и выдаем полученный результат.

□

Пусть $T(n,m)$ — мат. ожидание времени работы нашего алгоритма на графе с n вершинами и m ребрами (более строго, максимум этого мат. ожидания по всем графам). На рекурсивный вызов от $G_2(\frac{1}{2})$ тратится время $T(\frac{n}{8}, \frac{m}{2})$. Покажем, что на рекурсивный вызов от G_3 тратится время $T(\frac{n}{8}, \frac{n}{4})$.

Лемма 6.1. *Пусть $G(p)$ — граф, полученный из какого-то графа G выкидыванием каждого ребра с вероятностью $1-p$, и F — минимальный остовный лес в $G(p)$. Тогда мат. ожидание числа легких ребер в G относительно F не превосходит $\frac{n}{p}$, где n — число вершин.*

Доказательство. Лес F будем строить одновременно с графом $G(p)$. Упорядочим ребра G по возрастанию весов. Очередное ребро с вероятностью p добавляем в $G(p)$. Если добавленное ребро соединяет в F разные компоненты связности, то добавляем его в F (как в алгоритме Краскала). Таким образом, в F попадет $n-1$ ребро. Легкими в G будут эти $n-1 \leq n$, а также те, которые попали бы в F, если бы их случайно не отбросили. Так как в среднем "отсев проходит" каждое $\frac{1}{p}$-е ребро, то всего легких ребер будет в среднем не более, чем $\frac{n}{p}$. □

Упражнение 6.1. Доказать лемму 6.1 более формально.

Лемма 6.2. *Мат. ожидание количества ребер в графе G_2, легких относительно леса F, не превосходит $n/4$.*

Доказательство. В этом графе $\frac{n}{8}$ вершин, а $p = \frac{1}{2}$. Применяем предыдущую лемму. □

Таким образом, для мат. ожидания времени работы нашего алгоритма, очевидно, верно соотношение:

$$T(n,m) \leq T\left(\frac{n}{8}, \frac{m}{2}\right) + T\left(\frac{n}{8}, \frac{n}{4}\right) + c(m+n).$$

Упражнение 6.2. Доказать, что $T(n,m) = O(n+m)$.

6.3 Алгоритм Морриса–Пратта для поиска образца в строке

Пусть есть две строки: a и b. Требуется определить, содержит ли строка a строку b в качестве подстроки. Мы уже знаем простой вероятностный алгоритм для этой задачи (см. лекцию 1 в первой части курса), работающий время $O(m+n)$, где n и m – длины a и b. "Обычный" детерминированный алгоритм имеет трудоемкость $O(mn)$. Это все потому, что каждый раз, найдя различие, он начинает поиск заново, со следующей позиции в a, откатываясь в b на самое начало. А это совсем не обязательно, если предварительно обработать b и получить полезную информацию. Именно, нужно построить для b функцию откатов, которая будет говорить, откуда нужно возобновить поиск. То есть, если найдено различие в i-й позиции, она укажет, сколько позиций все же совпало. Эту функцию можно определить следующим образом:

$$f(i) = \max\{s \in [0..s-1] \mid b_1 \dots b_s = b_{i-s+1} \dots b_i\}.$$

Когда она найдена, то поиск требует линейного количества операций. Остается ее найти. Для этого есть такой алгоритм:

Алгоритм 6.2 (Вычисление функции откатов).

1. $f(1)$:=0;
2. for j:=2 to $length(b)$ do
3. begin
4. $i := f(j-1)$;
5. while $(b_j \neq b_{i+1})$ and $(i > 0)$ do $i := f(i)$;
6. if $(i > 0)$ or $(b_j = b_{i+1})$
7. then $f(j) := i + 1$ (*нашли*)
8. else $f(j) := 0$ (*не нашли*)
9. end;

Этот алгоритм работает за время $O(length(b))$, так как во время его работы i увеличится не более $length(b)$ раз (и всего на единицу — в строке 7), а все остальное время будет уменьшаться (при каждой итерации цикла while — хотя бы на единицу).

Итого, алгоритм Морриса–Пратта — линейный.