1 De Bruijn Sequence

1.1 Задача

Есть бинарные строчки длины n, можно на них построить ориентированый граф G_1. Ребра есть из $a_0a_1\dots a_{n-1}$ в $a_1\dots a_{n-1}a_n$ (дописали цифру в конец, стерли из начала). Хотим найти Гамильтонов цикл в таком графе.

Равносильная переформулировка: построить циклическую бинарную строку длины 2^n такую, что каждая подстрока длины n в ней встречается ровно один раз.

1.2 Решение

Гамильтонов цикл искать сложно, поэтому давайте лучше научимся искать вместо него Эйлеров.

Давайте для этого определим аналогичный граф на 2^{n-1} вершине (переходы строим аналогично G_1). Каждому ребру будет взаимооднозначно соответствовать бинарная строка длины n (она равна строке длиной $N-1$, соответствующей вершине, плюс символ, написанный на ребре). Замечаем, что Гамильтонов цикл в G_1 — это то же самое, что и Эйлеров цикл в G_2 (потому что выписать последовательность рёбер G_2 — то же самое, что выписать последовательность вершин G_1). А искать Эйлеров цикл за линейное время мы уже умеем. Итого решили задачу за $O(2^n)$. Кстати, заметим, что Эйлеров цикл точно существует, потому наш G_2 связен и входящая/исходящая степени каждой вершины равны и равны двум.

1.3 Замечания к решению и обобщения

На самом деле, исходную задачу со строкой можно решать конструктивно.

А вот техника со сведением к Эйлеровому циклу помогает решать обобщения. Например, запретили строчки, в которых идут пять нулей подряд (и не рисуем соответствующие рёбра). Тогда можно сразу понять, существует ли ответ и найти его.

Другое обобщение: алфавит размером k, тогда из каждой вершины будет выходить/входить k рёбер. Без дополнительных ограничений цикл де Брёйна тоже существует и его можно найти.

2 Ориентация Графа

2.1 Задача

Задача: есть неориентированный граф, надо каждое ребро ориентировать (одним из двух способов) так, чтобы минимизировать максимальную исходящую степень. Никакой связности мы не требуем. Лучше, чтобы не было кратных рёбер (иначе потом оценка в $O(V^2E)$ сломается).

Подсказка для очень умных: потоков не надо, есть решение проще.

Эта задача — пример на локальные оптимизации.

2.2 Решение с локальными оптимизациями

Одно из решений: ориентируем как-нибудь, а потом попытаемся результат улучшить. Для улучшения выбираем произвольную вершину V, в которой достигается максимум исходящей степени $D = \deg V$ (помним, что граф сейчас уже ориентирован). Давайте рассмотрим какой-нибудь путь $V \to U$. Если перевернём ориентацию рёбер на этом пути, то исходящая степень вершины V уменьшится на один, а степень вершины U увеличится, для остальных вершин исходящая степень не поменяется. Если $\deg U \le D-2$, то у нас уменьшилось число вершин степени $\deg V$ и даже мог уменьшится максимум (если V была единственной с такой степенью). Теперь, собственно, алгоритм: пока существует путь $V \to U$ такой, что $\deg V = D \to \max, \deg U \le D-2$, инвертируем этот путь.

2.2.1 Оптимальность

Здесь и далее обозначим как D максимальную исходящую степень. Возьмём произвольную вершину $V \mid \deg V = D$. Запустим из неё dfs, он обойдёт множество A, из множества A рёбра наружу никуда не ведут. Исходящих рёбер внутри этого множества у нас хотя бы $|A| \cdot (D-1) + 1$ — это мы просто посчитали степень исходящих рёбер (потому что степень каждой вершины $\ge D-1$, а степень вершины V ровно D). Теперь заметим, что как бы мы не ориентировали рёбра, ведущие внутри

Егор Суворов
Всеволод Степанов

множества A, у нас точно будет вершина степени D (по принципу Дирихле: нельзя распределить хотя бы $|A| \cdot (D-1)+1$ единиц по $|A|$ вершинам так, чтобы у каждой было не более $D-1$ единицы). При этом рёбра из внешнего мира могут исходящую степень множества A только повысить. Значит, на всём графе тем более не получится.

2.2.2 Время работы

Оценим время работы. Прикольный пример: в ответе степень 10, нашли путь из вершины степени 100 в вершину степени 98, теперь они обе степени 99, это было довольно бессмысленно.

У нас есть X итераций, на каждой запускаем dfs за $O(E)$. За V итераций у нас ответ уменьшается хотя бы на единицу (а изначально он был D), поэтому время работы будет $O(D \cdot V \cdot E) \approx O(V^2 E)$, где D — текущая максимальная степень (так как нет кратных рёбер, то $D \le V$).

Еще пример: была вершина A степени 10 и вершина B степени 6. Провели пути $A \to B$, $A \to x_1, A \to x_2, \ldots, A \to x_6$. Теперь у вершины A степень 3, а у вершины B степень 7, проведём путь $B \to A$. Кажется, мы сначала увеличили степень B, а потом уменьшили обратно.

2.2.3 $O(E \log E)$ итераций

Хотим теперь оценить количество итераций в E (соображения: если все вершины примерно одной степени, то $D \approx \frac{E}{V}$, и $D \cdot V \approx E$). Серёже кажется, что это просто, но с ходу он умеет только $O(E \log E)$. Вот оно: пусть k — это число вершин степени D или $D-1$. Утверждается, что k всегда неуменьшается. Пусть z — количество рёбер, исходящих из этих k вершин. Пока k неувеличивается, z строго убывает. Теперь пусть у нас появилась новая вершина (и k как-то увеличилось), тогда у новой вершины степень не больше $D - 1 \le \frac{z}{k}$ (где $\frac{z}{k}$ — средняя степень вершин, бывших в k), то есть тогда z увеличилось не более, чем на $\frac{E}{k}$. Таким образом суммарных увеличений не более $\sum\limits_{k=1}^{n} \frac{E}{k} = O(E \log E)$. То есть итераций тоже не более, чем столько, потому что $z \ge 0$ и каждая итерация сначала уменьшает z на единицу (после чего, впрочем, может произойти увеличение в связи с уменьшением D).

2.2.4 $O(E)$ итераций

Оценка $O(E^2)$ операций, $O(E)$ итераций dfs-а. Рассмотрим вершину v, у которой мы на очередной итерации алгоритма уменьшаем степень. Заметим тогда, что всегда $deg_{out}(v) \ge max_u deg_{out}(u) - 1$, то есть мы этой вершине никогда не будем увеличивать степень. Такие вершины нам тогда дают в сумме $\sum v deg_{out}(v) \le E$ итераций dfs-а. Теперь рассмотрим вершину v, у которой на очередной итерации мы увеличили $deg_{out}(v)$ на 1. Заметим, что мы сначала будем только увеличивать степень этой вершины, потом, в какой-то момент, она станет максимальной, после чего мы ее будем только уменьшать. Итого, увеличение степеней дает не более $\sum v deg_{in}(v) \le E$ итераций, а последующее уменьшение, как мы уже выяснили, дает не более E итераций. Итого, мы делаем всего не более $3E$ итераций, то есть, асимптотика алгоритма $O(E^2)$

2.3 Другое решение

Другой алгоритм: бинарный поиск по ответу. Учимся проверять, что есть ответ с максимальной степенью не более x. Тогда у нас есть вершины со степенью $> x$ и $< x$, ищем путь из одних в другой. Тогда одни только уменьшаются, другие только увеличиваются, итого на каждой итерации бинпоиска будет $O(E^2)$ операций. То есть в сумме $O(E^2 \log V)$. Доказательство корректности проверки аналогично: если пути не нашлось, то найдётся компонента с кучей рёбер и противоречию по Дирихле.

2.4 Решение с потоками

Тем, кто знает про потоки: есть некое очевидное решение за $O(E^2)$.

Егор Суворов
Всеволод Степанов

3 Метод двух указателей

Задача: есть здание из n этажей, есть профессор и k яиц. Есть монотонная функция «разбилось ли яйцо при падении с некоторого этажа». Если яйцо не разбилось, можно переиспользовать. Надо найти границу функции, использовав минимальное число бросков.

Например, если $k \geq \log n$, то можно использовать бинарный поиск. Т.о. задача интересна лишь для малых k.

Динамика: есть k яиц и n этажей, про которые ничего не знаем (но знаем, что выше бьётся, ниже — не бьётся). Переход — перебрать, с какого этажа бросили и перебрать, разбилось потом или нет. Формула: $f_{n,k} = 1 + \min\limits_{i}(\max(f_{i-1,k+1}, f_{n-i,k}))$, — если нам сейчас не было ничего известно про n этажей, то после броска с i-го этажа яйцо либо разбивается, тогда нам не известно про $i-1$ этаж (будем искать последний этаж, при броске с которого не разбиваются яйца), и мы потратили одно яйцо, либо оно не разбивается, тогда яиц осталось столько же, а мы не знаем ничего про $n-i$ этаж. Итого решение за $O(n^2 \log n)$, где $k \leq \log n$.

Оптимизируем методом двух указателей до $O(n \log n)$. Замечание: $f_{n,k} \leq f_{n+1,k}$ (строгое доказательство от противного: если есть последовательность для $n+1$ этажа, то она подходит и для n этажей). Теперь зафиксируем состояние и посмотрим внутрь максимум на переходе, первая величина нестрого растёт, вторая нестрого убывает. Максимум из двух таких штук — это «галочка», надо найти «вершину» этой «галочки» (в ней минимум). Можно найти минимум не перебирая все варианты, а просто найдя такое i, что отношение между двумя величинами поменялось. Тогда надо будет посмотреть i и $i \pm 1$ (либо плюс, либо минус, в зависимости от того, что такое i). Пока еще ничего не соптимизировали. Обозначим это оптимальное i как $i_{n,k}$ — интересная позиция для (n,k).

Теперь замечание: $i_{n+1,k} \geq i_{n,k}$ (доказательство: в условии цикла `while`, который ищет нам i, первый операнд неувеличился, а второй неуменьшился, значит, итераций не меньше). Собственно, оптимизируем: начинаем искать $i_{n+1,k}$ начиная с $i_{n,k}$, а не с единицы. Можно, разумеется, не заводить отдельный двумерный массив. Получается амортизационный анализ цикла `while`. Итого решение за $O(n \log n)$, где $k \leq \log n$.

4 Количество различных чисел на отрезке $[L, R]$ в массиве длины n

Пусть есть m запросов, решаем в оффлайне. Хотим иллюстрировать метод двух указателей. Существует решение обычным (без персистентности и без двумерности) деревом отрезков за $O((n+m)\log n)$ (если добавить персистентность — онлайн).

4.1 $L_i \nearrow$, $R_i \nearrow$

То есть L_i и R_i неубывают. Научимся, зная количество различных чисел и зная в хэш-таблице для каждого числа, сколько раз оно встречалось, переходить от одного запроса к следующему. Для этого нужны две операции: `L++` (удаляет число с начала), `R++` (добавляет число в конец). Итого время обработки всех запросов будет $O(n+m)$ операций с хэш-таблицей.

А, еще нужно отсортировать запросы по какой-нибудь границе. Это можно делать подсчётом за линию.

4.2 Произвольные запросы

Добавим операцию `L` (`R` тоже можно, но нам не потребуется). Возьмём все запросы $0 \leq L_i < \sqrt{n}$, отсортируем по правому концу. Пусть их оказалось k, тогда по нашему алгоритму надо будет $O(n + k\sqrt{n})$ операций с хэш-таблицей (R пройдёт n, а L будет между соседями проходить не более $\sqrt{n}$).

Теперь у нас есть $\sqrt{n}$ промежутков: $[0, \sqrt{n}); [\sqrt{n}, 2\sqrt{n}); \ldots;$. Если в i-м отрезке оказалось k_i отрезков с таким левым концом, то, просуммировав по всем отрезкам, получим время работы $O((n+m)\sqrt{n})$.

Тут мы разбивали но отрезки длиной $\sqrt{n}$ в предположении, что $n \approx m$. В других соотношениях может быть иначе. Например, если $m \approx n^2$, то выгодно разбивать на отрезки длины 1.

Егор Суворов
Всеволод Степанов

4.3 Обобщение

Если есть запросы на отрезках $[L, R]$, при этом ответ для отрезка мы можем за время $O(t)$ чуть-чуть поменять границы, то умеем решать задачу в offline за время $O((n+m)t\sqrt{n})$.

5 Точки на прямой

Задача: есть n точек с весами на прямой, выбираем ровно k какие-нибудь точек (не обязательно из этих же), надо минимизировать функцию: сумма расстояний до ближайшей из k выбранных (расстояние от точки i учитывается с весом $w_i > 0$). Предположим, что точки уже отсортированы. Еще иногда можно услышать в формулировке «n домов и надо построить k почтовых отделений»

Заметим, что точки разделяются на k групп (каждая из которых является отрезком): для каких-то ближе первая выбранная, для каких-то ближе вторая выбранная и так далее. То есть при $k = 2$ можно перебрать границу разделения и попробовать решить по отдельности для каждой из половин.

5.1 Если точка одна

Вот есть выбранная точка. Рассмотрим сумму весов точек слева или под нами и сумму весов точек справа. Если будем двигать точку вправо, то ответ будет увеличиваться на сумму весов точек слева, уменьшаться на сумму весов точек справа. По смыслу посчитали производную. Заметим, что сумма весов слева увеличивается, сумма весов справа уменьшается, то есть можно найти момент, где они встречаются. Оптимально будет поставить именно в эту точку. Заметим, что достаточно будет лишь перебрать лишь данные точки, так как выбирать точку между какими-то двумя смысла нет, всегда можно подвинуть влево или вправо.

Теперь пусть мы посчитали оптимальное i_n для отрезка $[1 \dots n]$, тогда $i_n \le i_{n+1}$. Опять можно воспользоваться методом указателя, чтобы посчитать i_n для всех n. Итого линейное время.

5.2 Решение для двух точек

Посчитали на префиксах, посчитали на суффиксах аналогично. Перебрали точку разделения (середину отрезка, образованного двумя выбранными точками), для каждого такого положения ищем оптимальный ответ. Для этого заведем указатель, который будет указывать на оптимальный выбор точки в левой половине, заметим, что при движении точки разделения вправо он тоже будет двигаться вправо. Будем его двигать только в те позиции, когда либо слева либо справа отмеченная точка совпадает с одной из исходных, в итоге получим $O(n)$.

5.3 Много точек

Посчитаем динамику $f_{n,k}$ — для первых n точек выбрать k, $p_{n,k}$ — позиция разделения (не позиция почтового). Предподсчитаем для всех подотрезков оптимальное почтовое отделение за $O(n^2)$.

Если просто перебирать последний отрезок для последнего почтового отделения, то получим $O(n^2k)$. А теперь пишем фокус: $p_{n,k-1} \le p_{n,k} \le p_{n+1,k}$. Интуитивно вроде очевидно, а доказательство — в следующей серии, сегодня уже не успеваем.

Пока что алгоритм: перебираем $k \nearrow$, $n \searrow$, и тогда делаем переходы только из нужного промежутка, зная $p_{n,k-1}$ и $p_{n+1,k}$. И самое крутое: получился $O(n^2)$ (не куб, и не $O(nk)$!). Всё вместе работает за: $\sum\limits_{n,k}(p_{n+1,k} - p_{n,k-1} + 1) = nk + S$, где в S у нас почти все $p_{n,k}$ присутствуют с каждым из двух знаков (плюс и минус), а присутствуют только с одним знаком лишь линейное число слагаемых, каждое из которых не более n. То есть $S = O(n^2)$.

Егор Суворов
Всеволод Степанов