Разбор задачи «Нужно больше тренироваться»

Автор и разработчик: Ян Олеринский

Рассмотрим две последовательно сданные задачи во время t_i и t_{i+1}. Если часовой пояс k подходит под условие задачи, то данные задачи должны быть сданы либо в один день, либо в последовательные дни. То есть не должно существовать дня d, что:

$$\begin{cases} t_i < d \cdot m + k \\ t_{i+1} > (d+1) \cdot m + k - 1 \end{cases} \iff t_i + 1 \leqslant d \cdot m + k \leqslant t_{i+1} - m$$

Взяв обе части по модулю m, получаем, что если $t_{i+1} - t_i > m$, то все k лежащие по модулю от $((t_i + 1) \bmod m)$ до $(t_{i+1} \bmod m)$ удовлетворяют этому условию, поэтому не могут являться ответом. Здесь надо отдельно учесть случай, когда $t_{i+1} - t_i \geqslant 2m$, но это легко сделать, так как в этом случае ответ, очевидно, -1.

Таким образом, каждая пара соседних задач блокирует какой-то циклический отрезок часовых поясов. Для того, чтобы найти минимальный незаблокированный часовой пояс, можно использовать метод сканирующей прямой, что позволяет решить задачу за $\mathcal{O}(n \log n)$.

Разбор задачи «Опять запросы»

Автор и разработчик: Дмитрий Сосунов

Построим дерево отрезков на массиве a размера 2^n и на массиве b такого же размера, где $b_{i \oplus 2^k} = a_i$. Чем отличаются эти деревья? Утверждается, что второе дерево отрезков отличается от первого только порядком детей у каждой вершины на $(n-k)$-м уровне.

Отсюда легко следует, что чтобы получить дерево отрезков, построенное на массиве b, где $b_{i \oplus k} = a_i$, надо изменить порядок детей у каждой вершины на всех уровнях i, если $(n-i)$-й бит числа k равен 1.

Это замечание позволяет отвечать на нужные запросы, используя обычное дерево отрезков, построенное на массиве a, но изменяя порядок детей в каждой посещённой вершине, если это надо.

Таким образом, данная задача решается аналогично дереву отрезков с массовыми операциями за $\mathcal{O}(2^n + qn)$.

Разбор задачи «Задача для разминки ног»

Автор: Кирилл Кудряшов, разработчик: Игорь Маркелов

Для начала научимся решать задачу на дереве без запросов добавления и удаления рёбер, используя heavy-light декомпозицию. Для каждой вершины мы будем поддерживать количество путей и их суммарную длину, начинающихся в этой вершине. При том пути должны идти вниз по дереву, а их первое ребро должно быть лёгким.

Для каждого пути в декомпозиции посчитаем суммарную длину путей в исходном дереве, у которых самая высокая вершина принадлежит этому пути. Для этого напишем дерево отрезков. При слиянии двух вершин надо уметь находить суммарную длину путей, проходящих между ними. Для этого в вершине дерева отрезков надо для каждой стороны и каждого направления поддерживать суммарную длину и количество путей, идущих с той или иной стороны в том или ином направлении. Кроме этого, надо поддерживать ориентацию каждого ребра в дереве, что так же легко делается с помощью этой же heavy-light декомпозиции.

Такое дерево отрезков позволяет так же ориентировать какой-то отрезок рёбер в одну сторону, что нужно для решения задачи. Из тонкостей реализации стоит отметить, что, при изменении ориентации рёбер на каком-то пути в декомпозиции, суммарная длина и количество путей, которые надо поддерживать для каждой вершины могли измениться для некоторых вершин выше по дереву. Поэтому при обработке изменении ориентации рёбер надо рассмотреть не только пути, на которые

декомпозировался путь запроса, но и все пути выше по дереву. Такое решение можно реализовать за $\mathcal{O}(n \log n)$.

Чтобы научиться обрабатывать запросы добавления и удаления рёбер, надо вместо heavy-light декомпозиции использовать link-cut tree. В этом случае реализация будет немного отличаться от стандартной, так как в функции «expose» при перестроении путей надо аккуратно пересчитывать суммарную длину и количество путей идущих от самой нижней общей вершины двух путей вниз по дереву. Получаем решение за $\mathcal{O}(n \log n)$ с очень большой константой из-за большого количества информации, которую надо хранить в вершине link-cut tree.

Разбор задачи «Сложная задача»

Автор: Алексей Михненко, разработчик: Алексей Дацковский

Заметим, что если мы можем решить задачу a-й по счёту, а также можем решить её b-й, то мы можем переставить задачи так, чтобы решить данную задачу i-й по счёту для любого $a \leqslant i \leqslant b$. Это верно, так как, при обмене 2 соседних задач местами, положение задачи в порядке решения изменится максимум на 1. Тогда, нам достаточно найти для каждой задачи минимальный и максимальный номер, которым по счёту может быть решена данная задача.

Для достижения наименьшего номера нам выгодно, чтобы при каждом проходе по задачам Гена решал максимум 1 задачу, и он решил нашу задачу, как только она станет доступна. Для эмуляции этого процесса можно проходить по всем задачам в порядке убывания сложности и поддерживать текущий «буффер» задач (то есть задачи, которые уже доступны по сложности, но ещё не были решены). Тогда, Гена будет обязан решать одну задачу при каждом уменьшении сложности, а всё новые доступные задачи добавляются в буффер.

Для достижения наибольшего номера решения стратегия должна быть противоположной: несложно сделать так, чтобы все задачи с большей сложностью были решены раньше, но также нужно, чтобы перед выбранной задачей было решено как можно больше задач с меньшей сложностью. Для этого можно поддерживать все задачи легче текущей, доступные в данный момент.

Итого, для ответа на запрос нужно только проверить, попадает ли требуемая позиция в достижимый диапазон. Такое решение можно реализовать за $\mathcal{O}\left((n+q)\log n\right)$.

Разбор задачи «Разноцветный граф»

Автор: Александр Бабин, разработчик: Алексей Васильев

Для начала нужно проверить, что граф связный. Если это не так, то граф не может быть хорошим.

Теперь научимся решать задачу для конкретного графа. Посмотрим на какое-то подмножество цветов C и мысленно оставим в графе ребра только таких цветов. В этом графе найдем остовный лес, пусть в него вошло x_C ребер. Тогда если $x_C < |C|$, то граф не может быть хорошим, потому что как бы мы не выбрали по одному ребру каждого цвета из C, они будут образовывать цикл. Утверждается, что если для всех подмножеств ребер C выполнено условие $x_C >= |C|$, то граф хороший. Доказательство этого будет приведено далее.

Зафиксируем конкретное подмножество цветов C и научимся быстро проверять это условие. Пусть количество ребер в графе, состоящем только из ребер из C, равно e_C. Заметим, что если $e_C > \frac{|C|\cdot(|C|-1)}{2}$, то $x_C >= |C|$. Поэтому сначала проверим это условие, если оно не выполнилось то честно найдем размер остового леса за $\mathcal{O}(|C|^2)$. Таким образом, для фиксированного графа задачу можно решать за $\mathcal{O}\left(2^k \cdot k^2\right)$.

Чтобы решать задачу с обновлениями, достаточно для каждого цвета эффективно хранить множество рёбер этого цвета. Это можно реализовать с помощью std::set, что позволяет решать задачу за $\mathcal{O}\left(q \cdot \left(2^k \cdot k^2 + \log m\right)\right)$.

Теперь приведём доказательство того, что если $x_C >= |C|$ для всех C, то граф хороший. Необходимость очевидна и была доказана ранее. Осталось только доказать достаточность.

Посмотрим на произвольный остов всего графа. Если в этом остове есть ребро каждого цвета, то ответ уже найден. Иначе есть какой-то цвет c_1, который не встречается в остове. Посмотрим на произвольное ребро цвета c_1 и докажем, что существует остов, содержащий цвет c_1, а так же все цвета, которые уже содержаться в остове.

Заметим, что при при добавлении этого ребра в остов оно будет образовывать цикл. Если на этом цикле какой-то цвет встретился дважды в нашем остове, то мы можем удалить одно из ребер этого цвета и добавить ребро цвета c_1, что доказывает данный факт. Иначе пусть на этом цикле были цвета $c_2, \ldots, c_a$. Тогда по нашему условию остов из ребер только цвета $c_1, \ldots, c_a$ содержит хотя бы a ребер. Значит что-то где-то во вне нашего цикла есть ребро одного из цветов $c_1, \ldots, c_a$, которое не входит в остов. Тогда мы опять посмотрим на то, какой цикл оно образует и попробуем удалить одно из ребер цикла и добавить ребро c_1. Если этого сделать не получится, то сделаем то же самое и так далее. Заметим, что каждый раз мы увеличиваем множество рёбер, которые хоть раз попали на подобный цикл, а значит алгоритм конечен. А значит мы обязательно сможем найти остов, в котором будет рёбро цвета c_1, а так же все цвета, которые уже есть в остове. Что и требовалось доказать.

Алгоритм аналогичный доказательству позволяет решать задачу за $\mathcal{O}\left(q \cdot (poly(k) + \log m)\right)$, но это не требовалось в задаче. Так же, подобной асимптотики можно достигнуть при помощи алгоритма пересечения матроидов.

Разбор задачи «Турнир»

Разработчики: Антон Степанов и Филипп Грибов

Для начала запустим алгоритм поиска минимума. Для этого изначально скажем, что минимум равен 0. После этого будем перебирать вершины в любом порядке и каждый раз делать запрос про минимум и очередную вершину. Если ребро направлено в сторону минимума, то ничего не меняем, иначе присваиваем минимуму номер текущей вершины.

Суммарно на это мы потратим $n-1$ запрос, после чего получим структуру, в которой исходящая степень каждой вершины, кроме минимума, равна 1.

Теперь пройдёмся по всем вершинам кроме минимума в любом порядке. Если мы ещё не сделали запрос про минимум и очередную вершину, то сделаем его. Если ребро направлено в сторону минимума, то очередная вершина не может быть ответом, так как её исходящая степень точно больше 1. Если же ребро направлено от минимума и исходящая степень минимума уже точно больше 1, то закончим перебор.

Если исходящая степень минимума так и не превысила 1, то минимум, очевидно, и является ответом. Иначе, исходящая степень каждой вершины, которая потенциально может быть ответом, равна 1. Таким образом, если мы делаем запрос про две подобные вершины, то исходящая степень ровно одной из них превысит 1.

Есть одна проблема: про некоторые пары вершин мы уже спрашивали. Но если внимательно посмотреть на структуру запросов, которые мы сделали, можно заметить, что если убрать ориентацию рёбер и удалить все вершины, которые точно не могут быть ответом, то получится лес (множество деревьев). Таким образом, пока в лесу хотя бы 3 вершины найдутся два листа, про которые мы ещё не спрашивали. Спросим про них и удалим один из них.

После этого останется не больше двух кандидатов на ответ. До этого момента мы суммарно потратили не больше $2n-2$ запроса, так как после части решения с нахождением минимума каждый запрос, кроме возможно одного, удалял одну вершину из кандидатов.

Теперь можно проверить каждого кандидата наивно, используя на каждого не больше чем $n-1$ дополнительный запрос. Получили решение не больше чем за $4n-4$ запроса. Если немного аккуратнее оценить последнюю часть, то можно получить оценку в $4n-7$ запросов.

Разбор задачи «Космическое происшествие»

Автор: Алексей Михненко, разработчик: Виктор Романенко

Заметим, что, если возможно охладить реактор за k циклов, то и за любое количество циклов, которое больше, чем k, это также возможно. Поэтому можно использовать бинарный поиск для нахождения минимального количества циклов.

Осталось понять, как именно проверять, что конкретное число циклов подходит.

Пусть мы хотим проверить, что m циклов хватает. Разобьём задачу на 3 случая:

1. $A > B$
2. $A = B$
3. $A < B$

Давайте в любом из случаев сначала вычтем m раз из каждого элемента число B.

В первом случае посчитаем, сколько раз необходимо вычесть из каждого элемента $A - B$, чтобы этот элемент стал меньше 0. Если суммарное число вычитаний получается $\leqslant m$, то m циклов достаточно для охлаждения реактора.

Во втором случае достаточно проверить, что все элементы становятся отрицательными.

Третий случай схож с первым, но вместо вычитания из каждого элемента $A - B$, будем прибавлять $B - A$, пока элементы меньше 0. Если суммарное число вычитаний больше, чем m или какой-то элемент уже был больше либо равен 0, то m циклов недостаточно.

Заметим, что данные прибавления/вычитания в первом и третьем случаях для каждого элемента можно в выполнить формулой за $O(1)$. Итак, итоговая асимптотика решения составляет $O(n \log C)$, где C — максимальное значение t_i.

Разбор задачи «Обед»

Автор и разработчик: Степан Кузнецов

Разобьём многоугольник на n треугольников, i-й из них будет образован точкой C, i-й и $(i+1)$-й точками многоугольника. Пусть $S_i = 1$, если внутри i-го треугольника есть особая точка и $S_i = 0$ иначе. Тогда рассмотрим, какие ходы мы можем сделать:

- Если i-й и $(i+1)$-й треугольники ещё существуют (то есть i-я, $(i+1)$-я и $(i+2)$-я точки ещё не заменены на C), то точку $(i+1)$ можно заменить на C, если $S_i = 1$ или $S_{i+1} = 1$.
- Если i-й треугольник уже не существует, а $(i+1)$-й треугольник ещё существует и $S_{i+1} = 1$, то $(i+1)$-ю точку можно заменить на C.
- Если $(i+1)$-й треугольник уже не существует, а i-й треугольник ещё существует и $S_i = 1$, то $(i+1)$-ю точку можно заменить на C.

Таким образом, задача может быть перефразирована следующим образом:

- Дана строка S длины n, состоящая из нулей и единиц. Два игрока делают ходы по очереди, начиная с Алисы.
- За один ход можно выбрать два соседних символа строки (1-й и n-й символы считаются соседними) и, если хотя бы один из символов равен единице — заменить оба символа на нули.
- Проигрывает тот, кто не может сделать ход.

Если все символы строки равны 1, то не важно какой первый ход сделает Алиса, поэтому задача сводится к строке, в которой есть хотя бы один ноль.

Если в строке есть хотя бы один ноль, выделим все циклические отрезки из единиц. Заметим, что любой ход содержит хотя бы один символ из какого-то отрезка, при том каждый ход не может повлиять сразу на два отрезка из единиц. Поэтому игра разбивается на независимые игры на таких отрезках из единиц.

Для каждого отрезка из единиц легко посчитать функцию Шпрага-Гранди, так как если отрезок имеет длину m, то за один ход можно получить либо отрезок длины $m-1$, либо два новых отрезка с длинами k и $m-2-k$ (где $0 \leqslant k \leqslant m-2$). Таким образом для каждого m от 0 до n можно за $\mathcal{O}(n^2)$ насчитать функцию Шпрага-Гранди для отрезка из m единиц.

Для решения задачи осталось только заметить, что после добавления или удаления точки в бинарной строке меняется не больше одного символа, поэтому данные отрезки легко поддерживать, используя, например, std::set. Таким образом, можно поддерживать битовый ксор функций Гранди по всем отрезкам, что однозначно позволяет определить победителя игры.

Получили решение за $\mathcal{O}(n + (m+q)\log n)$.

Разбор задачи «Парные дороги»

Автор: Александр Бабин, разработчик: Тихон Евтеев

Для начала решим задачу за $\mathcal{O}(n^2)$, используя динамику по поддеревьям:
Будем считать $dp[u][cnt][flag]$, где

- u — номер вершины
- cnt — количество построенных пар дорог в поддереве
- $flag$ (0 или 1) — было ли взято ребро $u - p(u)$, хоть в одной из пар дорог.

Значение $dp[u][cnt][flag]$ — максимальная прибыль в поддереве, в описанных условиях.

Для пересчета динамики, нам понадобится ещё одна. Она позволяет пересчитать значение динамики в вершине u, через значения в сыновьях u.

Будем считать $dp_1[u][cnt][i][flag][parity]$, где

- u — номер текущей вершины
- cnt — количество построенных пар дорог в рассмотренной части поддерева u,
- i - номер первого нерассмотренного сына u
- $flag$ (0 или 1) — была ли вершина u взята в качестве центральной.
- $parity$ (0 или 1) — четность числа ребер, для которых u - центральная вершина.

Значение $dp_1[u][cnt][i][flag][parity]$ — опять же максимальная прибыль в рассмотренной части поддерева u, в описанных условиях.

Чуть подробнее про параметр $parity$: Мы хотим взять сколько-то пар ребер, с центральной вершиной u, однако мы не хотим при пересчете динамики каким-то образом выбирать обоих сыновей, образующих с u пару ребер одновременно. Вместо этого, мы пойдем по сыновьям поочереди (параметр i), и будем сохранять в параметре $parity$ — есть ли у нас лишнее ребро в вершину u, которому ещё нужно найти пару, чтобы получилась корректная пара ребер с центром в u.

Приведем пару примеров переходов в динамике (всего их 11: 8 — не берут ребро в сына, 2 — берут ребро в сына в качестве первого в пару, и 1 — в качестве второго в пару)

- $dp_1[u][cnt][i][1][0] \to dp_1[u][cnt + cnt_{son}][i+1][1][0]$, используя в сыне i значение $dp[son_i][cnt_{son}][1]$ — не берем ребро в i-го сына, оно уже было взято одим из пары с центром в son_i
- $dp_1[u][cnt][i][0][0] \to dp_1[u][cnt + cnt_{son} + 1][i+1][1][1]$, используя в сыне i значение $dp[son_i][cnt_{son}][0]$ — берем ребро в i-го сына в качестве первого в пару, при этом до этого мы не брали пары ребер с центром в u
- $dp_1[u][cnt][i][1][1] \to dp_1[u][cnt + cnt_{son}][i+1][1][0]$, используя в сыне i значение $dp[son_i][cnt_{son}][0]$ — берем ребро в i-го сына в качестве второго в пару ребер с центром в u.

Вы могли заметить, что состояние $flag = 0, parity = 1$ недостижимо, так что два этих параметра можно объеденить в коде, чтобы вместо 4-х состояний было всего 3, но это не обязательно.

После того, как мы прошлись по всем сыновьям, мы можем обновить значение $dp[u][cnt][0]$ через $dp_1[u][last][cnt][0][0]$ и $dp_1[u][last][cnt][1][0]$, а значение $dp[u][cnt][1]$ через $dp_1[u][last][cnt][1][1]$

В такой динамике всего $\mathcal{O}(n \cdot k)$ состояний, и если ограничить в поддереве u значение пакраметра cnt количеством ребер в поддереве u, деленым на 2 $(+ 1)$, то все пересчеты в сумме займут $\mathcal{O}(n^2)$ Времени. (а не $\mathcal{O}(n \cdot k^2)$, как может показаться на первй взгляд).

Перейдем наконец к полному решению:

Рассмотрим функцию $f(k)$ — ответ на задачу, в зависимости от требуемого количества пар ребер. Утверждается, что на отрезке $\left[0, \frac{(n-1)}{2}\right]$ эта функция нестрого выпукла. Это утверждение можно доказать по индукции по поддеревьям, но доказательство сильно сложнее всего остального решения задачи. Так же все значения вышеописанных динамик выпуклы по аргументу cnt (количеству взятых пар ребер).

Это позволяет применить к описанной динамике *lambda*-оптимизацию: Вместо того, чтобы хранить cnt в качестве параметра, мы будем считать $dp[u][flag]$ и $dp_1[u][i][flag][parity]$, значения которых: максимум величины [прибыль в рассмотренном пооддереве минус $cnt \cdot Cost$], для некоторого фиксированного значения $Cost$ — "стоимости" взятия каждой пары ребер. Для фиксированного значения $Cost$ такая динамика считается за $\mathcal{O}(n)$

Помимо самого значения максимума, будем хранить возможный отрезок значений cnt, на которых этот максимум достигается.

Бинарным поиском по $Cost$ найдем такое значение, при которм k, данное во входных данных, лежит на отрезке возможных значений cnt. Тогда ответ на задачу: значение $dp + k \cdot Cost$.

Чтобы восстановить ответ (непосредственно пары ребер, а не просто число), пройдемся в обратном порядке по пересчетам динамики, и если очередной пересчет дает максимум, и может дать требуемое значение cnt, то перейдем по нему, получив состояния, из которых можно восстановить ответ рекурсивно. Для этого потребуется дополнительно для каждого состояния динамики посчитать не только максимальную прибыль, но и минимальное и максимальное значение cnt при котором достигается максимум.

Итоговое время работы решения: $\mathcal{O}(n \cdot \log(C))$.