Задача A. Фрирен и гримуары

Автор задачи и разработчик: Даниил Голов

Первая подгруппа, как это обычно бывает, расчитана на произвольные корректные решения за любую достаточно адекватную асимптотику времени работы. Мы в разборе не будем сильно фокусироваться на частичных решениях и быстро перейдем к полному, только вкратце упомянув отдельные частные случаи.

Во **второй подгруппе**, в случае, когда все a_i одинаковы, можно заметить, что оба критерия выбора совпадают: в обоих случаях надо из еще не рассмотренных гримуаров выбрать тот, у которого максимально b_i, а при равенстве — минимально i. Таким образом, для решения достаточно было отсортировать все гримуары по убыванию b_i и возрастанию i, после чего ровно такой порядок вывести в ответ.

В **четвертой подгруппе** достаточно было n раз проходить циклом по всем доступным гримуарам и выбирать оптимальный из еще не просмотренных тот, который требуется. Для этого достаточно было хранить массив пометок «прочитан гримуар или нет» и рассматривать только те, у которых эта пометка равна `false`. После вывода очередного гримуара в ответ следовало у него заменить пометку на `true`. Общее время работы такого решения равно $\mathcal{O}(n^2)$.

Для **полного решения** стоило начать с применения той же идеи дважды. Отсортируем все гримуары по убыванию a_i (и затем по остальным критериям при равенстве), и сложим их в таком порядке в массив A. Теперь отдельно отсортируем их в порядке убывания b_i и сложим их в массив B. Таким образом, мы получаем два различных порядка на гримуарах, соответствующих их «оптимальности» по двум критериям.

Напишем следующее решение: если нас просят найти самый сложный гримуар, пройдемся по массиву A и выберем максимальный из еще не использованных, иначе — пройдемся по массиву B. Если каждый раз проходиться от начала соответствующего массива, получится уже описанное решение за $\mathcal{O}(n^2)$, поэтому надо как-то отслеживать уже взятые гримуары и пропускать их.

Это можно было делать по-разному в третьей, пятой и шестой подгруппах, но общая идея заключается в методе двух указателей: будем поддерживать позицию последнего взятого гримуара из A и последнего взятого из B, и при выборе очередного будем просто двигать соответствующий указатель вперед по массиву (так как поддерживается инвариант, что все предыдущие уже были прочитаны и выведены в ответ). Помимо этого достаточно поддерживать уже описанные выше пометки того, прочитан каждый гримуар или нет. Тогда, если, двигаясь вперед по A соответствующим указателем, мы встречаем уже прочитанный гримуар, это означает, что он был прочитан когда-то раньше, пока мы двигались по B. В таком случае его надо просто пропустить. В сумме оба указателя будут сдвинуты не более $2n$ раз, поэтому время работы решения равно $\mathcal{O}(\texttt{sort} + n) = \mathcal{O}(n \log n)$.

Пятая подгруппа была скорее расчитана на другое аналогичное решение, в котором вместо массивов A и B можно было завести два дерева поиска или кучи (`set` или `priority_queue`, он же `heap`), упорядочивающих элементы по тем же критериям, а при удалении очередного гримуара из одного из них удалять и из второго. В таком случае в этой подгруппе можно было не беспокоиться о том, что пары (a_i, b_i) могут повторяться, и спокойно использовать `set` или `map` вместо `multiset`.

Задача B. Фрирен и барьер

Автор задачи: Михаил Иванов, разработчик: Даниил Орешников

Первая подгруппа подразумевала достаточно прямолинейное решение за $\mathcal{O}(n^3 \log n)$: переберем все t от 1 до k, для каждого вычислим все неполные частные от деления a_i на t, после чего найдем `mex` стандартным алгоритмом. Под стандартным алгоритмом мы здесь имеем в виду сортировку полученной последовательности и линейный поиск первого целого числа, которое в ней отсутствует. Либо, например, можно сложить все ее элементы в `set` и также проверить, какого минимального целого числа в нем нет.

В **четвертой подгруппе** достаточно было немного ускорить описанное выше решение, заметив, что порядок на a_i не меняется, если каждое из них поделить на одно и то же t. Поэтому можно заранее отсортировать a_i, и после искать `mex` для каждой последовательности за $\mathcal{O}(n)$, что дает нам решение за $\mathcal{O}(n^2)$ в сумме.

Вторая и третья подгруппы аналогичны друг другу в том наблюдении, что `mex` в таком случае никогда не может быть больше 11. Действительно, в случае, когда $a_i \leqslant 10$, числа 11 в принципе быть в наших последовательностях не может. Тогда как при $n \leqslant 10$, если в последовательности есть числа от 0 до 9, то для 10 места не остается.

Во второй группе достаточно заметить, что при $t > 10$ всегда будет выполняться $b_t = 1$, так как в последовательности частных есть 0, но нет 1. Более того, повторяющиеся a_i можно не рассматривать, поэтому для $t \leqslant 10$ можно было найти ответ наивным решением, перебирающим не больше 10 различных чисел..

В третьей группе нужен уже другой подход. Чтобы выполнялось $b_t = m$, должно выполняться следующее:

- для всех $q < m$ существует такое a_i, что $\left\lfloor \frac{a_i}{t} \right\rfloor = q$;
- не существует такого a_i, что $\left\lfloor \frac{a_i}{t} \right\rfloor = m$.

Если немного переписать эти условия, получится, что для всех $q < m$ должен существовать a_i между $q \cdot t$ и $(q+1) \cdot t - 1$ включительно. В третьей группе это можно проверять для каждых потенциальных t и m за $\mathcal{O}(n)$.

Последние две подгруппы расчитаны на полное решение и некоторые его вариации (например, в пятой подгруппе ожидалось, что кто-то может придумать решение с использованием корневых эвристик или просто менее оптимальную реализацию полного решения). Здесь же просто приведем полное решение.

Воспользуемся рассуждениями, описанными выше для третьей группы и поймем, как за $\mathcal{O}(\frac{n}{t})$ находить b_t. Мы уже выяснили, что `mex` неполных частных от деления a_i на t — это такое минимальное m, что не существует $a_i \in [mt, (m+1)t - 1]$. Построим массив `flag`, что `flag`$[x] = 1$, если существует $a_i = x$, и 0 иначе. Если на таком массиве посчитать префиксные суммы, мы сможем за $\mathcal{O}(1)$ получать информацию о том, существует ли хотя бы один a_i со значением в произвольном интересующем нас интервале.

Переберем все t от 1 до k, и для каждого переберем m от 0 до $\left\lceil \frac{\max(a)}{t} \right\rceil$. Если для очередного m не существует a_i между mt и $(m+1)t - 1$, то m и будет ответом для этого t. В сумме такое решение работает за $\sum\limits_{t=1}^{k} \frac{n}{k} = \mathcal{O}(n \log n)$, так как это известное равенство, вытекающее из суммы гармонического ряда.

Задача C. Фрирен и интересные вопросы

Автор задачи и разработчик: Михаил Тихонов

Переформулируем задачу в более удобном для понимания виде, из чего напрямую будет следовать базовое решение. Создадим граф на n вершинах, соответствующих числам от 1 до n, и будем представлять a-интересные и ab-интересные пары как ребра в этом графе.

Тогда определение взаимно интересной пары просто указывает на пару вершин, между которыми в таком графе есть путь. Поскольку ребра неориентированные, взаимно интересными будут числа, лежащие в одной компоненте связности. Если построить граф в явном виде, и на каждый запрос запускать `dfs`, получится решение за $\mathcal{O}(qn^2)$, проходящее **первую подгруппу**.

Во **второй подгруппе**, в случае, когда все $b_i = 0$, из-за того, что все $a_i \geqslant 1$, никакие два года не являются ab-интересными. Это немного упрощает задачу, так как останется рассмотреть только a-интересные года. В остальном решение не отличается от полного.

В **третьей подгруппе** все a_i простые, значит два года являются a-интересными тогда и только тогда, когда $a_i = a_j$. Таким образом a-интересные или ab-интересные пары индексов — это пары индексов, на которых в a или в a и b стоят одинаковые значения. Аналогично предыдущей группе, дальше решение не отличается от полного, только ребра графа строить сильно проще.

Для решения **четвертой и пятой подгрупп** достаточно реализовать наивное построение графа за $\mathcal{O}(n^2)$, определить в нем компненты связаности с помощью какого-нибудь обхода (`dfs`, `bfs`), а затем отвечать за $\mathcal{O}(1)$, определяя интересность годов по компонентам связности, в которых эти года лежат (взаимно интересные если в одной компоненте связности, и не интересные, если в разных).

Отличие между группами заключается в том, что в четвертой подгруппе можно себе позволить при ответе на каждый запрос делать обход графа заново. Это позволяло получить баллы за эту подгруппу, даже не приходя к представлению годов и связей в виде графа: достаточно было реализовать алгоритм, аналогичный обходу, не строя граф в явном виде.

Для **полного решения** воспользуемся идеей из пятой подзадачи (построением графа) и научимся строить граф и соединять вершинки в одну компоненту связности быстрее. Так как мы свели ответ на запрос к задаче определения принадлежности вершин одной компоненте связности, применим подходящую структуру данных: *dsu* (система непересекающихся множеств), способную объединять вершины и отвечать на запрос «лежат ли они в одной компоненте связности» за $\mathcal{O}(\alpha(n))$, что принято считать близким к $\mathcal{O}(1)$.

Однако мы строим граф все еще за $\mathcal{O}(n^2)$. Для быстрого определения *a-интересных* лет сделаем следующее: заведем массив `idx`, хранящий для каждого уникального значения в массиве a любую из его позиций. Иными словами, сделаем присвоение $\texttt{idx}[a_i] \leftarrow i$ для всех i. Заметим, что по мере заполнения этого массива можно в СНМ объединять индексы, на которых стоят одинаковые числа.

Теперь воспользуемся идеей из решета Эратосфена, работающего за $\mathcal{O}(A \log A)$: для каждого $1 \leqslant t \leqslant A$, если в массиве $\texttt{idx}[t] \neq \varnothing$ (т.е. в массиве a существует $a_i = t$), пройдемся по всем числам p, которые тоже есть в a и для которых t является делителем, и объединим в СНМ $\texttt{idx}[t]$ и $\texttt{idx}[p]$. Так мы свяжем все a-интересные года за $\mathcal{O}(A \log A \cdot \alpha(n))$, где $A = \max(a)$.

Осталось пройтись по массиву b, и все i связать в СНМ с $\texttt{idx}[b_i]$. Так как до этого мы уже объединили все a-интересные числа в одну КС, нам достаточно добавить только ребра, задающие ab-интересные года, а для этого как раз надо связать для одинаковых чисел его произвольную позицию вхождения в a с каждым его индексом в b. Осталось только ответить на все q запросов с помощью СНМ.

Итоговая сложность решения: $\mathcal{O}((A \log A + n + q) \cdot \alpha(n))$.

Задача D. Фрирен и память о героях

Авторы задачи: Даниил Орешников и Владимир Рябчун, разработчики: Константин Бац и Даниил Орешников

Решение **первой подгруппы** заключалось в прямолинейной реализации всего процесса. Для каждого запроса типа '`?`' просто запустим обход дерева (`dfs` или `bfs`) из соответствующей вершины, и, приходя в какую-либо другую вершину, пройдемся по всем запросам, предшествующим данному, чтобы посчитать, в течение какого времени в соответствующем городе уровень памяти уменьшался. В качестве ответа надо просто вывести максимальную из встреченных величин.

Решение **второй и третьей подгрупп** могло быть получено из описанного выше следующими оптимизациями:

1. во-первых, можно было не проходить по всем запросам в каждой вершине, а сгруппировать запросы по вершинам, на которые они влияют — тогда решение будет работать не за $\mathcal{O}(nq^2)$, а за $\mathcal{O}(q(n+q))$;

2. также можно было подвесить дерево за произвольную вершину и сделать предподсчет, позволяющий находить `lca` за $\mathcal{O}(1)$, например, разреженную таблицу на эйлеровом обходе — тогда не будет необходимости каждый раз запускать `dfs`, расстояние между двумя вершинами можно будет находить за $\mathcal{O}(1)$;

3. ну и наконец, вместо того, чтобы при ответе на каждый запрос типа '`?`' перебирать все предшествующие запросы, можно по мере их обработки поддерживать информацию об уровне памяти в городах: последний год изменения состояния, уровень памяти на тот момент, и уменьшается ли он сейчас; по такой тройке величин всегда можно восстановить текущий уровень памяти для любого более позднего года.

В зависимости от эффективности реализаций этих оптимизаций, решение проходило вторую или вторую и третью подгруппы. Вторая оптимизация при этом не дает большого выигрыша в данном случае, но имеет пользу в одном из полных решений.

Оставшиеся подгруппы решались теми или иными вариациями двух известных авторам решений: одно основано на дереве отрезков на эйлеровом обходе дерева, а другое — на центроидной декомпозиции. Мы здесь подробно приведем только первое решение, а про второе скажем только вкратце.

Заметим, что если подвесить дерево за вершину 1, то, стартуя из вершины 1, мы придем в вершину v в момент времени $\texttt{depth}(v)$, где $\texttt{depth}$ — глубина вершины. Если уровень памяти изначально равен s_v и постоянно уменьшается, то в этот момент времени он будет равен $\max(0, s_v - \texttt{depth}(v))$. Аналогично, если стартовать из вершины 1 в момент времени t, уровень памяти в вершине v к моменту прихода в нее будет равен $\max(0, s_v - \texttt{depth}(v) - t)$.

Тогда для решения **четвертой подгруппы** остается только заметить, что при изменении вершины старта на соседнюю (а точнее, на одного из ее детей в подвешенном дереве), расстояние до всех вершин в поддереве уменьшается на 1 (соответственно, уровни памяти к моменту прихода в них увеличиваются на 1), а до всех остальных вершин — наоборот, увеличивается на 1.

Сделаем следующее: построим эйлеров обход дерева, и для каждой вершины выпишем $s_v - \texttt{depth}(v)$. Построим на таких величинах дерево отрезков, тогда ответом для вершины 1 при старте в момент времени 0 будет просто максимум на всем дереве отрезков. После обойдем все дерево, и при переходе в вершину u будем вычитать 1 на всем дереве, после чего прибавлять 2 на отрезке, соответствующем поддереву u. Эти два действия соответствуют пересчету уровней памяти, описанному выше. Затем ответ для вершины u так же находим как максимум на всем дереве отрезков.

Понятно, что при возвращении из нижних вершин в верхние описанные операции надо откатывать, но это не повлияет на асимптотику: за время $\mathcal{O}(n \log n)$ мы сможем посчитать ответы для всех вершин при условии, что старт пути происходит в момент времени 0. Для запроса со стартом в момент времени t надо просто уменьшить предподсчитанный ответ на t.

Пятая подгруппа отличалась от предыдущей тем, что в некоторые моменты уровень памяти мог перестать уменьшаться (после чего фиксировался и больше не менялся). Любое решение предыдущей подгруппы не очень сложно переделывается:

1. найдем уже описанным алгоритмом ответ, считая, что уровень памяти все так же уменьшался для всех вершин с самого начала (тогда мы можем получить максимум в городе, в котором на самом деле память не уменьшалась, но в таком случае мы обновим ответ через заведомо большую величину на втором шаге, см. ниже);

2. учтем те города, в которых уровень памяти остановился на фиксированной величине — для этого достаточно просто поддерживать максимум из уровней памяти в тех городах, для которых уменьшение остановилось, это делается просто одной лишней переменной.

Для **полного решения** авторы использовали корневую декомпозицию по запросам. Будем для каждого города поддерживать информацию о последнем запросе изменения в нем и уровне памяти на тот момент (см. решение второй и третьей группы). А также разобьем все запросы изменений на блоки ($Q \approx 6000$), и для каждого блока выделим:

- города, в которых память уменьшалась в течение всего блока;
- города, в которых память не изменялась в течение всего блока;
- города, для которых в течение блока случались запросы изменений.

Для городов первого типа просто применим уже описанное выше решение. Для городов второго типа воспользуемся идеей из пятой группы: просто найдем максимум из значений уровня памяти в них, и будем всегда через них обновлять ответ. А городов третьего типа может быть не больше Q: для каждого из них просто найдем расстояние между ними и текущей вершиной, и обновим ответ. Для поиска расстояния можно было воспользоваться разреженной таблицей для поиска $\texttt{lca}$ за $\mathcal{O}(1)$.

Общее время работы такого решения равно $\mathcal{O}(\frac{q}{Q} \cdot n \log n + q \cdot Q)$. Последняя подгруппа была расчитана на подбор оптимального значения для размера блока и на другие оптимизации корневой декомпозиции.

Альтернативное решение через центроидную декомпозицию получало полный балл с запасом по времени. Для этого достаточно было построить центроидную декомпозицию дерева, после чего отдельно учитывать все города, в которых уровень памяти не уменьшается, как и было описано выше, и отдельно хранить для каждого центроида c в его «компоненте» максимуму величины $\texttt{opt}_c = s_v - \texttt{dict}(c, v)$. Тогда ответ на запрос типа '?' из вершины v — это просто проход по всем центроидам, в компонентах которых лежит стартовая вершина, и обновление ответа через $\texttt{opt}_c + \texttt{dist}(c, v)$.