Задача A. Распределение пряности

Автор задачи и разработчик: Даниил Орешников

Первые подгруппы расчитаны на частные решения.

- В **первой подгруппе** $n \leqslant 3$, то есть всего не более трех направлений; в таком случае ответ всегда не более 3, и можно просто в несколько условий перебрать все возможные исходы.
- Во **второй подгруппе** $k = 0$, то есть только одинаковые направления считаются похожими. В таком случае ответ равен максимальному количеству вхождений какого-либо числа в набор a_i: действительно, меньше нельзя, а для такого ответа легко строится пример.

В **третьей подгруппе** можно было написать полный перебор, всеми возможными способами распределяющий единицы пряности между студентами. Для такого перебора достаточно поддерживать, какие направления уже соответствуют каким студентам, и для каждого нового либо выдавать его студенту без нарушения правил, либо выдавать новому студенту. Если еще заранее упорядочить все a_i по неубыванию, такой перебор гарантированно проходил ограничения по времени.

В **четвертой подгруппе** работает решение через динамическое программирование, но мы сразу приведем решение для **четвертой и пятой подгрупп**, подводящее к полному решению. Упорядочим a_i по неубыванию и для каждого направления a_i найдем такое минимальное $j_i > i$, что $a_{j_i} - a_i > k$. Тогда все направления с i по $j_i - 1$ обязаны быть выданы разным студентам.

Утверждается, что ответ в таком случае равен $\max\limits_{i=1}^{n} j_i - i$, то есть максимальной длине такого отрезка. Опять же, оценка понятна: если в каждом таком отрезке все направления достались разным студентам, то количество студентов не меньше длины максимального из отрезков. Пример же строится жадно или конструктивно. Например, если мы определили, что студентов в ответе будет m, достаточно выдать направление a_i студенту с номером $i \bmod m + 1$. Тогда любые два направления у одного студента отличаются хотя бы на $a_{i+m} - a_i$, что больше k по нашему выбору m.

В четвертой подгруппе для вычисления такого m можно было просто суммировать $\texttt{cnt}[x] + \texttt{cnt}[x+1] + \texttt{cnt}[x+2]$, перебирая все возможные x (где `cnt` — количество вхождений числа в последовательность a_i). В пятой же было достаточно для каждого i от 1 до n находить j_i за время $\mathcal{O}(n)$.

Полное решение же отличается только применением методов двух указателей: если $i_1 \leqslant i_2$, то и $j_{i_1} \leqslant j_{i_2}$ по логике задачи. Тогда можно двумя указателями найти j_i для каждого i в сумме за $\mathcal{O}(n)$ времени.

Задача B. Добыча пряности

Автор задачи и разработчик: Даниил Орешников

Сразу отметим, что авторское решение в этой задаче довольно запутанно и заключается в переборе большого числа случаев. Есть также альтернативное решение, которое мы приведем сразу. Дальше для простоты будем считать, что время измеряется в минутах.

1. Сделаем двоичный поиск по ответу: очевидно, что если t минут хватает, то и $> t$ минут хватает.
2. Чтобы проверить, что хватает ровно t минут, заметим, что первый компонент можно производить непрерывно каждые t_1 минут, а сами харвестеры достаточно производить также непрерывно в последние $n \cdot t_3$ минут производства.
3. Остается только распределить, в какие моменты времени начинается производство вторых компонентов. Для этого необходимо, чтобы перед началом производства был доступный компонент типа I, а также чтобы перед началом производства очередного харвестера были свободные компоненты типа I и II.
4. Выпишем события вида «+1 компонент типа I» и требования вида «должно быть по 1 компоненту типов I и II» каждое в свой момент времени.

5. Если представлять количество компонентов типа I и II парой (c_1, c_2), то производство первого компонента — это событие вида $(+1, 0)$, производство второго — событие вида $(-1, +1)$ (-1 в начале интервала, $+1$ — в конце), а производство харвестера — событие вида $(-1, -1)$.

6. События первого и третьего вида уже расставлены. Остается расставить события второго вида так, чтобы ни c_1, ни c_2 не опускались ниже нуля. Утверждается, что если это можно сделать, это можно сделать жадным образом.

7. А именно, последнее производство компонента II поставим ровно перед производством последнего харвестера (позже производить нет смысла, а если можно произвести раньше, то можно и позже). Предпоследнее — сразу перед этим, и так далее.

8. Остается пройтись по событиям в порядке их следования во времени, и проверить, что ни c_1, ни c_2 не опускаются ниже нуля. Если это так, то за t минут можно все изготовить, иначе — нельзя.

Тем не менее, можно привести более конструктивное решение. В **первой подзадаче** достаточно произвести один харвестер, и это всегда можно сделать за $t_1 + \max(t_1, t_2) + t_3$ времени. В остальных подзадачах надо было смотреть на соотношения между t_1, t_2 и t_3.

Без строго доказательства (оставляем его читателям) отметим, что

- при $t_2 \geqslant 2t_1$ ответ всегда равен $t_1 + t_2 + t_3 + (n-1) \cdot \max(t_2, t_3)$, потому что компонентов первого типа всегда будет хватать, и все зависит только от скорости второй и третьей стадий производства;
- при $t_1 = t_2$ и $t_3 \geqslant 2t_1$ ответ всегда равен $2t_1 + nt_3$;
- при $t_1 = t_2$ и $t_3 \leqslant t_1$ ответ всегда равен $2nt_1 + t_3$;
- иначе, при $t_3 \geqslant t_1 + \max(t_1, t_2)$ ответ всегда равен $t_1 + \max(t_1, t_2) + nt_3$.

В оставшихся случаях можно показать, что $t_1 = 2$, $t_2 \in \{1, 3\}$ и $t_3 \leqslant 4$, то есть время изготовления всех n харвестеров будет $\leqslant c \cdot n$ для некоторой небольшой константы c. Тогда можно сделать динамическое программирование вида $\texttt{dp}[i][j]$ — минимальный момент времени, в котором «прогресс» производства компонентов типа II равен i, а «прогресс» производства харвестеров равен j.

Здесь за «прогресс» считаем время, потраченное на производства без учета простоя, то есть если i — прогресс второго этапа производства, произведено $\left\lfloor \frac{i}{t_2} \right\rfloor$ компонентов типа II, и на изготовление следующего уже потрачено $i \bmod t_2$ времени. Такое $\texttt{dp}$ можно посчитать за $\mathcal{O}(n^2)$. За подробностями можете обратиться к файлу с авторским решением.

Задача C. Предательство Юэ

Автор задачи и разработчик: Николай Ведерников

Подсчитаем, сколько существует половин кода длины n с суммой цифр s и произведением p. Пусть кодов длины n, у которых сумма и произведение равны (s, p), ровно a, пар с суммой и произведением цифр $(s-1, p)$, ровно b, и пар с $(s+1, p)$, ровно c. Тогда в ответ надо добавить $a \cdot (a+b+c)$.

Осталось научиться считать количество таких пар. Для этого воспользуемся динамическим программированием.

Пусть у нас для длины i подсчитано количество кодов с параметрами (s, p). Тогда на позицию $i+1$ мы можем поставить цифру d (от 1 до 9), и тогда для всех пар $(s+d, p \cdot d)$ мы должны добавить ответ для пары (s, p). Это достаточно стандартная динамика по цифрам. Обратите внимание, что проще написать динамику «вперед», чем «назад».

Корректная реализация такой динамики спокойно проходила тесты первой группы.

Далее стоит обратить внимание, что произведение и ответ может выйти за стандартные целочисленные типы данных, и поэтому надо воспользоваться длинной арифметикой или написать решение на языках, в которых она встроенная, например, на языке `Python`.

В зависимости от эффективности реализации можно набрать различное число баллов. Но для **полного решения** достаточно заметить, что ввод может принимать всего 42 различных значения, а тогда все ответы можно предподсчитать у себя на компьютере и отправить программу, которая будет выводить уже посчитанные ответы.

Задача D. Песчаная буря

Автор задачи и разработчик: Даниил Орешников

В **первой и второй подзадачах**, поскольку все запросы не пересекаются, суммарное количество изменений высот не превысит $\mathcal{O}(n)$. То есть можно каждый запрос обработать «наивным» образом, пройдясь по соответствующему отрезку и изменив значения высот. При изменении очередной высоты можно за $\mathcal{O}(1)$ пересчитать их сумму.

Тем не менее, в этих подзадачах можно было также отталкиваться и от специального вида входных данных, чтобы пересчитывать сумму еще проще. В первой подзадаче все высоты изначально равны, а тогда для каждого запроса либо изменение суммы будет нулевое, либо будет равно $(f_i - h) \cdot (r_i - l_i + 1)$. Во второй подзадаче можно было для поиска изменившихся значений использовать двоичный поиск.

В **третьей подзадаче** достаточно было так же за линейное время обработать каждый запрос, после чего заново за $\mathcal{O}(n)$ пересчитать сумму. Самые неэффективные реализации проходили только эту группу тестов, но любая аккуратная реализация проходила и **пятую подзадачу**. Аналогично, **четвертая подзадача** расчитана на аккуратную реализацию того же «наивного» решения. Просто в каждой из первых подзадач можно было, допустив ошибку в такой реализации, переписать решение на более частное.

Основную сложность составляло, конечно, **полное решение**. Здесь достаточно, отталкиваясь от необходимости делать изменения на отрезках, построить дерево отрезков, а точнее — Merge Sort Tree. В таком дереве в каждой вершине хранится отсортированная копия соответствующего отрезка исходного массива.

Помимо этого будем в каждой вершине хранить актуальный уровень песчаной бури, покрывающей соответствующий отрезок. Тогда для выполнения запроса отложим информацию о нем в соответствующих отрезках ДО, и, чтобы пересчитать сумму, в каждой затронутой вершине сделаем двоичный поиск: все меньшие значения высот не изменились, а все большие вошли в сумму как f_i, а не как исходная высота здания. Для быстрого пересчета достаточно было также хранить префиксные суммы на отсортированных в каждой вершине массивах.

Это решение можно было еще соптимизировать, если хранить данные немного по-другому и применять технику частичного каскадирования, однако и текущее решение за $\mathcal{O}(q \log^2 n)$ проходило ограничения по времени.