Задача A. Алиса, Боб и два массива

Автор и разработчик задачи: Антон Степанов

Запишем оба массива в явном виде (избавимся от записи массивов в виде отрезков). В $dp[x][y]$ будем хранить выигрышность игры, если из a мы удалили первые x символов, а из b первые y символов.

Переберем символ, который поставит первый из игроков, пересчитаем dp. Это решение работает за $O(NMk)$.

Теперь нам нужно избавиться от перебора нового символа. Будем перебирать по убыванию сначала x, затем y. Заметим, что когда мы уменьшаем y на 1 изменяется переход только по 1 цвету, поэтому мы можем просто поддерживать количество цветов, по которым есть переходы в проигрышные позиции. Это позволяет получить решение за $O(NM)$.

Будем нумеровать индексы массивов с единицы.

Заметим, что если мы сделали ход, то мы окажемся в позиции (x, y), что $a_x = b_y$. Давайте считать dp только по таким позициям.

Зафиксируем отрезок одинаковых символов, в котором лежит x, и отрезок одинаковых, в котором лежит y. Пусть это отрезки $[lx, rx]$, $[ly, ry]$. Оба этих отрезка состоят из одинаковых символов, назовем этот символ t.

Мы хотим узнавать dp для всех пар (x, y), где x лежит в отрезке $[lx, rx]$, а y лежит в отрезке $[ly, ry]$. Заметим, что для всех этих пар, переходы по символам кроме t ведут в одни и те же позиции. Если среди этих переходов есть проигрышная позиция, то все интересные для нас позиции выигрышные, так как из всех них есть переход в проигрышную.

Иначе, если нет перехода в проигрышную позицию, то нет смысла переходить по символам, кроме t. При этом мы можем спокойно делать переходы по символу t, пока не окажемся в новом блоке (т. е. пока не выйдем за отрезки $[lx, rx], [ly, ry]$).

Последнее замечание состоит в том, что если мы переходим по новому символу (т. е. не по тому, в котором сейчас стоим), то мы окажемся в начале блока. Т. е. если мы оказались в позиции (x', y'), то x' находится в начале отрезка одинаковых символов, и y' находится в начале отрезка одинаковых. Поэтому давайте хранить dp только для таких позиций (т. е. таких, которые находятся в начале блока).

1. Используя эти замечания мы будем поступать так.
2. Храним dp только для позиций в началах блоков.
3. dp для этих позиций будем насчитывать ответы, перебирая их с конца в начало.
4. Чтобы найти ответ для других (x, y), переберем переходы по «другим» символам.
5. Если есть переход в проигрышную позицию, то позиция (x, y) выигрышная.
6. Иначе переходим по символу a_x, пока не окажемся в новом блоке, после возвращаемся к шагу 4.

Это работает за $O(n^3k)$.

Используя ту же оптимизацию, что и для решения за $O(NM)$, мы можем избавится от перебора символа и получим решение за $O(n^3)$.

Решение за $O(n^3)$ при должной оптимизации уже может заходить на 100 баллов, но у авторов есть решение за $O(n^2log)$. Объясним общую идею решения.

Рассмотрим все пары (x, y) с цветом t. Если мы будем рассматривать переходы только по символу t, то все пары разобьются на «диагональки», так что при переходе по символу t происходит переход в следующую пару на диагональке.

Мы хотим для каждого символа уметь прыгать по его диагональке, пока не окажемся в блоке, из которого есть переход в проигрышную позицию (переход по другому символу). Для этого просто для каждой диагонильки будем хранить последнюю пару на этой диагонильке, из которой есть переход в проигрышную по другому символу.

Это можно делать неявной ДО за $O(n^2log)$, однако такое решение имеет слишком большую константу.

Заметим, что количество интересных для нас диагоналей равно $O(nm+q)$, поэтому мы можем просто сжать координаты и написать ДО снизу, такое решение наберет 100 баллов.

Задача B. Арифметическое упражнение

Автор и разработчик задачи: Андрей Пархаев

Подгруппа 1**.**

Пусть все x_i равны x. В первой подгруппе предлагается на каждом шаге применять изменение к нулевым элементам массива, если это возможно. Если же нулевых ячеек нет, будем применять изменения к самому последнему элементу массива. Тогда в последней ячейке будет чередоваться значение $x, -x, x, ...$

Подгруппа 2**.**

Существует 2^m различных способа сделать последовательность изменений, т.к. каждая из m операций применятся либо к первому, либо ко второму элементу a. Переберем всевозможные последовательности изменений и посчитаем сумму после их применения. Среди всех таких сумм возьмем максимальную. Такое решение работает за $O(m \cdot 2^m)$.

Подгруппа 3**.**

Воспользуемся динамическим программированием. Пусть $dp[pref][i][j] = true/false$ — можно ли обработать первые $pref$ изменений, чтобы выполнялось $a_1 = i, a_2 = j$. Обратите внимание, что i и j могут быть отрицательными. Понятно, что a_1 и a_2 не могут по модулю превысить сумму значений x_i. Сумма всех x_i не превосходит $10m$, поэтому мы можем хранить значения в диапазоне от -500 до 500.

Подгруппа 4.

Улучшим динамику из предыдущей группы. Заметим, что при фиксированном значении a_1 на префиксе, нам имеет смысл пересчитываться только через минимальный или максимальный доступный a_2. Поэтому будем хранить явное значение только одного из элементов массива. Заведем $dp_{min}[pref][i]$ и $dp_{max}[pref][i]$, которые хранят минимальное и максимальное достижимое значение a_2 на префиксе, если $a_1 = i$.

Замечание.

Сделаем замечание, которое поможет нам существенно продвинуться к решению. Понятно, что каждый элемент последовательности x_i войдет в итоговую сумму либо со знаком «+», либо со знаком «−». Давайте разобьем все m элементов на n последовательностей, в зависимости от того, к какой позиции массива a было применено изменение. Некоторые из последовательностей могут быть пустыми. В каждой последовательности знаки элементов чередуются, и последний элемент всегда имеет знак «+». А потому, если заменить знаки на $+1$ и -1 соответственно, то сумма на каждом суффиксе таких массивов будет равна либо 1, либо 0. Совместим все последовательности обратно в одну и посмотрим на сумму знаков на произвольном суффиксе. Она будет находится в диапазоне $[0; n]$. Теперь мы можем использовать это, как критерий для проверки последовательности знаков на правильность.

Подгруппа 5.

Переберем знак для каждого x_i и проверим, что на каждом суффиксе выполняется описанный критерий. Решение работает за $O(m \cdot 2^m)$.

Подгруппа 6 **и** 7.

Снова воспользуемся динамикой. Пусть $dp[i][j]$ — максимальная сумма, которую можно получить на суффиксе i, если текущий баланс знаков равен j. Тогда $dp[i][j] = max(dp[i+1][j-1] + x_i, dp[i+1][j+1] - x_i)$. Такое решение работает за $O(nm)$.

Подгруппа 8.

В этой подгруппе предлагается действовать жадно, соблюдая критерий баланса. Например, можно идти с конца последовательности x_i. На каждом шаге будем пытаться взять новый элемент и, при надобности, выкидывать какой-то элемент, который был взят ранее. Т.к. различных значений не более двух, выкидывать имеет смысл только минимум. Будем поддерживать в очереди минимумы, которые были взяты со знаком «+» и которые еще можно выкинуть.

Подгруппа 9 **и** 10.

Существует несколько альтернативных подходов. Рассмотрим несколько из них.

1. Улучшим решение для предыдущей подгруппы. Будем идти с конца, храня текущую расстановку знаков для элементов суффикса. В дереве отрезков будем поддерживать элементы, которым мы можем изменить знак на противоположный. Когда переходим к новому элементу, пытаемся поставить ему знак «+» (в случае, если это невозможно, меняем знак минимального элемента со знаком «+», для которого это можно сделать) и смотрим, как меняется сумма от этого действия. Аналогично, пробуем поставить знак «−» и смотрим, как меняется сумма. Среди двух вариантов, выбираем тот, в котором сумма будет наибольшей.

2. Отсортируем последовательность x_i по убыванию математического модуля. Будем идти по отсортированным значениям и на каждом шаге пытаться поставить элементу нужный знак (если число положительное — «+», иначе «−»), если это возможно. Иначе будем ставить тот, знак, который мы вынуждены поставить. Чтобы понимать, можем ли мы поставить тот или иной знак, будем хранить дерево отрезков, в каждой позиции которого хранится текущий баланс соответствующего суффикса. Если мы хотим поставить очередной знак, надо посмотреть на минимальный и максимальный баланс который уже достигается перед ним и проверить, можно ли расставить оставшиеся знаки так, чтобы критерий баланса не нарушался.

3. Вспомним решение для подгупп 6 и 7, использующее динамическое программирование. Заметим, что функция выпуклая отдельно по двум четностям, поэтому мы можем воспользоваться slope trick.

Задача C. Мечтать не вредно

Авторы задачи: Константин Амеличев, Тимофей Федосеев
Разработчик задачи: Тимофей Федосеев

Сначала сделаем общее замечание, которое поможет нам в процессе решения всех подзадач. Рассмотрим порядок, в котором вершины будут удалены в исходном дереве, и дальше для простоты будем называть его порядком удаления.

Рассмотрим произвольную вершину v и обозначим за S_v множество вершин, идущих перед v в порядке удаления. Теперь определим оптимальную вершину для обнуления. Заметим, что бессмысленно обнулять вершину на пути от v до корня, так как это может только ухудшить позицию v в порядке удаления. При обнулении вершины c, не лежащей на пути до корня, позиция v в порядке удаления уменьшится на количество вершин из S_v в поддереве вершины c.

Таким образом, требуется найти поддерево, не содержащее вершину v, с наибольшим количеством вершин, идущих перед v в порядке удаления.

В дальнейшем под суммой в поддереве мы будем понимать количество вершин из S_v в этом поддереве.

- **Подзадача 1. Не более двух бамбуков (10 баллов)**
  В этой подзадаче дерево состоит не более чем из двух бамбуков, подвешенных к корню. В этом случае найти порядок удаления можно методом двух указателей. Оптимальной вершиной для обнуления является корень одного из бамбуков. Для ответа на запросы достаточно пройти по порядку удаления и поддерживать сумму в обоих бамбуках.

- **Подзадача 2. Произвольное количество бамбуков (6 баллов)**
  Решение аналогично подзадаче 1. Для нахождения порядка удаления нужно эффективно склеить несколько списков, это можно сделать с помощью структуры данных, например очереди с приоритетом или множества (set). Теперь при прохождении по порядку удаления будем поддерживать сумму в каждом бамбуке и дополнительно отслеживать два бамбука с максимальной суммой. Для ответа на запрос выберем лучший из них, который не содержит вершину запроса.

Теперь научимся проводить симуляцию процесса в общем случае. Будем поддерживать множество детей корня в структуре данных. На очередной итерации удалим из него вершину с максимальным значением и добавим её детей. В зависимости от выбранной структуры данных асимптотика

составит $O(n \log n)$ или $O(n^2)$. Для эффективной реализации подойдет очередь с приоритетом или множество (set).

- **Подзадача 3. Любая симуляция (8 баллов)**
  В этой подзадаче требуется написать любую корректную симуляцию. Для ответа на запрос переберем вершину для обнуления и запустим симуляцию. Получим решение не хуже $O(n^4)$.

- **Подзадача 4.** $O(n^2 \log n)$ **(13 баллов)**
  Переберем вершину для обнуления, запустим симуляцию за $O(n \log n)$ и обновим ответ для каждой вершины ее номером в получившемся порядке удаления.

- **Подзадача 5.** $O(n \log n + nq)$ **(11 баллов)**
  Будем идти по порядку удаления и поддерживать множество просмотренных вершин. Для ответа на запрос посчитаем суммы в поддеревьях и выберем максимальное, не содержащее вершину запроса.

- **Подзадача 6. Сбалансированное двоичное дерево (9 баллов)**
  Будем идти по порядку удаления и поддерживать сумму в каждом поддереве. Эту информацию можно пересчитывать за $O(\log n)$ при рассмотрении очередной вершины. Заметим, что оптимальная вершина для обнуления смежна с путём от вершины запроса до корня. Для ответа на запрос рассмотрим все такие вершины и получим решение за $O(n \log n)$.

- **Подзадача 7.** $O(n \log n + nh)$ **(11 баллов)**
  Будем действовать аналогично подзадаче 6. Помимо суммы в поддеревьях будем также для каждой вершины поддерживать двух детей с максимальной суммой. Пересчитывать такую информацию при добавлении вершины и находить поддерево, смежное с путём до корня и с максимальной суммой, можно за $O(h)$.

- **Подзадача 8. Куча на минимум (14 баллов)**
  Можно заметить, что в этом случае каждое поддерево, смежное с путём от вершины запроса до корня, идёт в порядке удаления либо целиком до вершины запроса, либо целиком после.

  Воспользуемся этим и предварительно посчитаем размеры поддеревьев, затем будем обходить дерево в глубину. Будем поддерживать размер оптимального поддерева, смежного с путём. Находясь в очередной вершине, отсортируем её детей по значению. Дети с большим значением будут целиком идти в порядке удаления перед детьми с меньшим значением. Запустим из них по очереди обход в глубину, обновив значение оптимального поддерева размером детей перед ними.

  В результате такого обхода можно также выписать порядок удаления за линейное время. Получим решение за $O(n)$.

- **Подзадача 9. Полное решение** $O(n \log^2 n)$ **(18 баллов)**
  Аналогично подзадаче 6 будем идти по порядку удаления и поддерживать суммы в поддеревьях. Для этого воспользуемся структурой данных Heavy-Light Decomposition (HLD). При рассмотрении очередной вершины прибавим 1 на пути до корня. Для ответа на запрос вычтем ∞ на пути до корня (чтобы не рассматривать поддеревья, содержащие вершину запроса), возьмем максимум в дереве и затем отменим вычитание.

Задача D. Милые подпоследовательности

Автор и разработчик задачи: Алексей Васильев

Пусть $i_1, \ldots, i_k$ — индексы, которые максимизируют ценность для каждой выбранной подпоследовательности в оптимальном ответе. Заметим, что сам ответ тогда равен $a_{i_1} + \ldots + a_{i_k} + max(i_1, \ldots, i_k)$.

Значит, что если мы зафиксировали $max(i_1, \ldots, i_k) = x$, то мы хотим максимизировать сумму $k-1$ элемента массива слева от x-го. Для этого можно идти слева, поддерживая мультисет, в котором

хранятся текущие $k-1$ максимальных элементов и поддерживать их сумму. Тогда ответом будет максимум по всем x, $x + a_x + sum_x$, где sum_x — сумма $k-1$ максимальных элементов слева от x.

Задача E. Сильная связность наносит ответный удар

Авторы задачи: Роман Первутинский, Денис Мустафин
Разработчик задачи: Роман Первутинский

Подгруппа 1

Для каждого подмножества рёбер проверим, поменяются ли КСС, если развернуть лежащие в нём рёбра. Хорошее подмножество не содержит подмножеств, для которых КСС не поменяются. Для каждого подмножества проверим, является ли оно хорошим, перебрав все его подмножества. С поиском КСС алгоритмом Тарьяна или Косарайю получаем решение за $O(3^m + 2^m \cdot m)$.

Подгруппа 2

Для каждого множества нам нужно проверить, содержит ли оно одно из «плохих» подмножеств. Воспользуемся ДП по подмножествам:

$$\text{is\_good}[S] = \neg\text{can\_reverse}[S] \wedge \bigwedge_{e \in S} \text{is\_good}[S \setminus \{e\}]$$

Итого получаем решение за $O(2^m \cdot m)$.

Подгруппы 3, 4 (ациклический граф)

Рассмотрим некоторое ребро $e = (u, v)$. Если не существует другого пути из u в v, e можно развернуть, и граф останется ациклическим. Таким образом, e не может лежать в хорошем множестве.

Пусть теперь существует другой путь из u в v. Тогда мы можем восстановить направление e из направлений рёбер на этом пути, то есть направление на самом e можно стереть. Множество из всех таких рёбер e — хорошее. Это множество является единственным максимальным.

Чтобы проверить существование пути, не проходящего через ребро, удалим ребро из графа и запустим поиск в ширину. Итого получаем решение за $O(m^2)$.

Подгруппа 5 (гамильтонов цикл)

Назовём все рёбра, не лежащие на гамильтоновом цикле, *прямыми*. Поскольку при их развороте сильная связность графа не нарушается, они не могут лежать в хорошем множестве.

Прямые рёбра покрывают некоторые отрезки цикла. Рассмотрим пересечение нескольких таких отрезков (возможно, не связное, поскольку отрезки циклические). Утверждение: все рёбра цикла, принадлежащие этому пересечению, можно развернуть, и граф останется сильно связным.

Таким образом, хотя бы одно ребро цикла на пересечении отрезков должно не лежать в хорошем множестве. Заметим, что если мы можем взять ещё один отрезок в наше пересечение, и оно уменьшится, мы получим более сильное условие. При этом «минимальные» пересечения, которые больше нельзя уменьшить, не пересекаются.

Выберем на каждом минимальном пересечении некоторое ребро, и возьмём множество из всех рёбер, кроме прямых и выбранных. Почему оно будет хорошим? Поскольку пересечения минимальны, в графе их рёбра образуют цепочки между вершинами степени 2. Поскольку в итоговом графе не должно быть стоков и истоков, направления на всех рёбрах цепочки воостанавливаются однозначно.

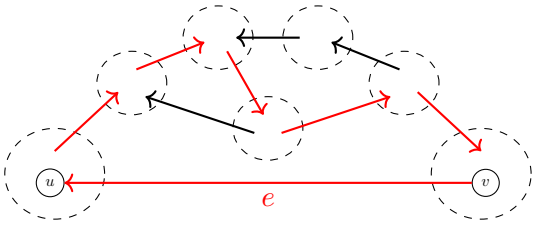
Полученное множество будет максимальным хорошим. Количество способов — произведение размеров минимальных пересечений.

Подгруппа 6 (сильно связный граф)

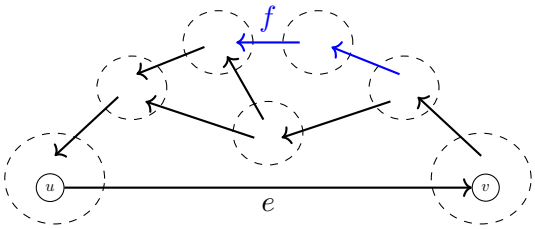
Обобщим идею «минимальных» множеств, которые не должны полностью лежать в хорошем, на общий случай.

Рассмотрим ребро $e = (u, v)$. Опять, если при его удалении граф останется сильно связным, e не может лежать в хорошем множестве. Иначе при развороте e граф распадётся на несколько КСС. Назовём множество рёбер, принадлежащих конденсации, $S(e)$ ($e \in S(e)$).

Пусть мы развернули e и хотим развернуть ещё некоторое подмножество рёбер, чтобы граф остался сильно связным. Мы обязаны развернуть некоторое подмножество $S(e)$, чтобы появился путь из u в v.



Заметим, что если $f \in S(e)$, то $S(f) \subseteq S(e)$. Если $S(f) = S(e)$ для всех $f \in S(e)$, то рёбра в $S(e)$ образуют цикл в конденсации. Назовём такое $S(e)$ *нужным* — это эквивалент «минимального» множества из прошлого решения.



Рассмотрим некоторое множество рёбер, которое можно развернуть, и граф останется сильно связным. Пусть e — ребро из этого множества с минимальным $S(e)$. Тогда $S(e)$ — нужное. Действительно, пусть это не так, тогда существует такое ребро $f \in S(e)$, что существует путь из v в u в графе конденсации, не проходящий через f. Тогда $S(f) \subset S(e)$; противоречие.

Поскольку $S(e)$ образует цикл, $S(e)$ целиком лежит в рассматриваемом множестве. Поэтому любое множество рёбер, которое можно развернуть без нарушения сильной связности, содержит какое-то нужное.

При этом можно развернуть само $S(e)$, чтобы граф остался сильно связным. Таким образом, A хорошее $\iff$ A не содержит ни одно нужное $S(e)$.

Для этого нам нужно не взять в A по одному ребру из каждого нужного множества. Поскольку они не пересекаются, сделать это можно как угодно. Максимальное хорошее множество состоит из всех рёбер, кроме не влияющих на сильную связность и выбранных из нужных множеств. Количество способов — произведение размеров нужных множеств.

Полное решение

Решим задачу отдельно для каждой КСС и для графа конденсации. В графе конденсации может быть несколько рёбер, соединяющих две КСС; из них мы должны не взять в хорошее множество одно любое. При этом домножим ответ на их количество.

Итоговое решение работает за $O(m^2)$.

Решение на частичные баллы

Из полного решения следует интересный факт: любое хорошее множество лежит в некотором максимальном хорошем. Это позволяет написать следующее жадное решение:

Пусть мы уже выбрали некоторое хорошее множество. Теперь мы хотим понять, можно ли добавить в него ещё какое-то ребро. В каком случае мы не можем этого сделать? В том, когда можно развернуть наше ребро и ещё какое-то подмножество уже стёртых, сохранив сильную связность.

Как уже сказано выше, если после разворота нашего ребра (u, v) перестанет существовать путь из u в v, то нам нужно развернуть какие-то ребра из уже взятого множества, чтобы этот путь появился. Проверить, можно ли это сделать, мы можем, ища путь из u в v в графе, где для каждого стёртого ребра мы добавили развёрнутое.

Если такого пути нет, то ребро (u, v) можно стереть. Почему, если такой путь существует, мы не можем это сделать? Утверждается, что после разворота ребра (u, v) и рёбер на найдённом пути граф останется сильно связным. Действительно, мы получили цикл из этого пути и развернутого ребра (v, u). Для всех рёбер, которые мы развернули на этом пути, достижимость между концами ребра осталась, поскольку все вершины на цикле достижимы друг из друга.

Рассмотрев все рёбра графа в любом порядке, мы построим некоторое максимальное хорошее множество.

Задача F. Лучший бегун

Автор задачи: Сергей Князевский
Разработчик задачи: Алексей Дацковский

Сначала разберём подгруппы, в которых не используется основная идея из общего решения:

Подгруппа 2: Всегда выиграет первый бегун: он должен после каждой дорожки перемещаться на одну влево (пока это возможно). Так он пробежит больше всех дорожек, так как он бегал по максимально коротким дорожкам.

Подгруппа 3: Можно решить с помощью динамического программирования. $dp[i][j]$ (где i — текущая дорожка бегуна, а j — оставшееся время) равно максимальному количеству дорожек, которое можно пробежать, начиная с такой ситуации.

Подгруппа 5: Всегда выигрывает бегун, который начинает на дорожке с минимальной длиной, так как он может пробежать максимальное число раз по самой короткой дорожке.

Для остальных подгрупп нужно понять, как будет выглядеть оптимальный маршрут бегуна. Пусть он начинает на дорожке i.

- Он должен закончить на самой короткой дорожке, на которой он был (обозначим её j). Если бы он остался на самой короткой дорожке и бегал по ней до конца времени, то пробежал бы не меньше дорожек, чем в любом другом сценарии.
- Также он должен максимально быстро перейти от i до j, и потом бегать только по j. Так он пробежит максимальное число дорожек среди всех сценариев.

Пусть у нас есть бегун с начальной дорожкой i и конечной дорожкой j. Определим количество дорожек, которое он пробежит. Не умаляя общности, пусть $i < j$. Тогда он потратит $a_i + a_{i+1} + \ldots + a_{j-1}$ времени, чтобы дойти до дорожки j, а потом оставшееся время будет бегать по j. Значит, он пробежит $j - i + \lfloor \frac{T-(a_i+a_{i+1}+\ldots+a_{j-1})}{a_j} \rfloor$ дорожек (если время от дорожки i до дорожки j не превышает T). Если поддерживать префиксные суммы массива a, то мы сможем вычислять количество дорожек за $O(1)$.

Этого достаточно, чтобы решить **подгруппу** 1: для каждой начальной позиции перебрать все конечные позиции и найти максимальное число дорожек.

Подгруппа 4: здесь нужно перебирать в качестве конечных позиций только те дорожки, которые строго короче всех дорожек между начальной и ней. Так как длины дорожек не превосходят 20, то для каждой начальной позиции мы рассмотрим не более 20 конечных дорожек слева и справа. А чтобы быстро находить ближайшую слева (или справа) дорожку короче текущей, можно предпосчитать их заранее.

Подгруппа 6: осталось применить небольшой трюк. Будем наоборот для каждой конечной позиции j находить бегуна, который пробежит максимальное число дорожек и закончит на дорожке j. Можно показать, что это будет либо ближайший бегун слева от дорожки, либо ближайший бегун

справа (или бегун, начинающий на этой дорожке, если такой есть) — другие бегуны потратят больше времени на то, чтобы добраться до j (если j — оптимальная конечная позиция). Значит, нам нужно перебрать не более 2 начальных дорожек для каждой конечной дорожки.

Задача G. Переворот карт

Автор задачи: Алексей Михненко
Разработчик задачи: Артём Агафонов

Пусть на столе осталось k карт, все из которых являются односторонними или перевёрнутыми двусторонними. Тогда игроки могут совершать только первое действие, а значит победитель определяется однозначно. Если k нечётное, то побеждает первый игрок, иначе — второй.

Пусть теперь на столе лежат односторонние и перевёрнутые карты, а также одна неперевёрнутая двусторонняя карта, действие с которой будет совершать текущий игрок. Заметим, что в такой позиции текущий игрок побеждает. Действительно, каждый из двух возможных ходов оставит после себя число карт разной чётности, а значит текущий игрок может совершить ход, который по прошлому замечанию однозначно будет гарантировать ему победу. Получаем, что в игре побеждает тот игрок, который сможет взять последнюю двустороннюю карту.

Прошлое замечание позволяет упростить задачу. Обозначим за x наибольшее число, записанное на лицевой стороне двусторонних карт. Тот игрок, который будет играть с картой x, побеждает. Тогда:

1. Мы можем удалить все односторонние карты с числом больше x. Их наличие никак не влияет на номер игрока, который будет совершать ход с картой x.

2. Мы можем заменить все двусторонние карты, имеющие число на обратной стороне большее x, на односторонние карты. Заметим, что вне зависимости от того, будет убрана или перевёрнута данная карта, после хода она не будет влиять на номер игрока, который возьмёт карту x. Значит мы можем считать, что это просто односторонняя карта с числом, записанным на лицевой стороне исходной двусторонней карты.

3. После двух прошлых изменений в момент, когда ход дойдёт до карты x, никаких других карт не останется, а значит текущий игрок обязательно сбросит её, чтобы обеспечить себе победу. Поэтому мы можем заменить данную карту на одностороннюю с числом x.

Данное упрощение задачи не привело к изменению победителя игры, но число двусторонних карт на столе уменьшилось хотя бы на 1.

Так как после каждого упрощения задачи число двусторонних карт уменьшается, то, применив данную операцию некоторое конечное число раз, мы получим позицию, содержащую только односторонние карты. Тогда, воспользовавшись первым замечанием, мы сможем определить победителя данной игры.

Заметим, что, чтобы упростить задачу, необходимо перебрать все карты, то есть сделать $O(n+m)$ действий. Так как после этого число двусторонних карт уменьшится хотя бы на 1, то суммарно будет не более $O(n)$ упрощений. Таким образом мы получили решение, имеющее сложность $O(n(n+m))$.

Попробуем ускорить предыдущее решение. Вместо того, чтобы честно совершать очередное упрощение, будем просто искать двустороннюю карту, которая будет последней в новой игре. Заметим, что данная карта обладает следующим свойством: на её обратной стороне записано число меньшее текущего x, и среди таких карт она обладает наибольшим числом на лицевой стороне. Тогда, отсортировав двусторонние карты в порядке уменьшения числа на лицевой стороне за $O(n\log(n))$, мы будем перебирать их в таком порядке и в тот момент, когда будет выполнено условие на число, записанное с обратной стороны, будем обновлять значение x, тем самым переходя к следующей карте. Совершив такое действие, мы найдём число x — число на лицевой стороне двусторонней карты, которая будет последней в последнем упрощении. Посмотрев на чётность количества двусторонних карт, имеющих на лицевой стороне число не большее x, и односторонних карт, имеющих число меньшее x, мы определим победителя игры. Сложность данного решения — $O(n\log(n)+m)$.

Используя сортировку подсчётом, можем получить асимптотику $O(n+m)$, но в данной задаче этого не требовалось.

Задача H. Порядковая статистика

Автор и разработчик задачи: Валерий Родионов

В подзадаче 1 достаточно промоделировать процесс из условия за $(mn \log n)$, m раз сортируя массив и уменьшая на 1 последние k элементов.

В подзадачах 2-4 выполняется $k = 1$, то есть на каждой из m операций максимальный элемент в массиве уменьшается на 1. Предположим, что наибольший элемент в массиве после m операций равен x. Тогда выполняется $m \leqslant \max(a_1 - x, 0) + \ldots + \max(a_n - x, 0) = T(x)$ Нетрудно заметить также, что наибольшее значение в массиве после m операций равно наименьшему x, удовлетворяющему этому условию. Найдем это x с помощью бинарного поиска. После $T(x)$ операций все элементы массива, которые были больше x, станут равны x, а остальные не изменятся. Мы знаем, что меньше максимум в массиве после этого уже не станет, а значит оставшиеся $m - T(x)$ операций будут уменьшать элементы, равные x. Мы знаем количество оставшихся операций, поэтому просто уменьшим на 1 любые $m - T(x)$ элементов, равных x, и получим финальный массив. Такое решение работает за $O(n \log \max a_i)$ и проходит подзадачу 2. Чтобы пройти подзадачи 3 и 4, нужно научиться быстро считать величину $T(x)$ для заданного x. В подзадаче 3 нет изменений, поэтому достаточно насчитанных префиксных сумм на отсортированном массиве a, а в подзадаче 4 для поддержания изменений использовать дерево отрезков, дерево Фенвика или декартово дерево, получая итоговую асимптотику $O(n \log m \log n)$.

В подзадаче 5 выполняется $k = 2$. Отсортируем массив a. Пока a_n больше максимального из $a_1, a_2, \ldots, a_{n-1}$, задача эквивалентна случаю $k = 1$, так как на каждом шаге уменьшается a_n и максимальный из $a_1, a_2, \ldots, a_{n-1}$. Найдем с помощью бинарного поиска и решения для $k = 1$ момент, когда a_n станет равен максимуму на префиксе $a_1, a_2, \ldots, a_{n-1}$. Эта часть решения работает за $O(n \log^2 C)$. Снова отсортируем массив, теперь выполняется $a_n = a_{n-1}$. С этого момента отсортированный массив после каждой операции будет выглядеть следующим образом: некоторый неизменный префикс массива, после идет суффикс, состоящий из чисел x или $x + 1$ для некоторого x, причем длина суффикса не меньше 2. Для простоты скажем, что длина суффикса четная, нечетный случай разбирается более-менее аналогично. Если L — длина суффикса, и минимальное значение на суффиксе больше максимального на префиксе, то за $L/2$ операций все элементы на суффиксе уменьшатся на 1. В тот момент, когда x станет равен максимуму на префиксе, длина суффикса увеличится на 1. Всего расширений суффикса будет не больше n,. Если аккуратно обработать все расширения, то эта часть решения будет работать за $O(n)$. Получаем решение за $O(n \log^2 C)$.

В подзадаче 6 выполняется $m \leqslant 10^6$. Научимся обрабатывать одну операцию за $O(\log n)$, при этом поддерживая массив отсортированным. Пусть массив перед операцией отсортирован, и на позиции $n - k + 1$ стоит число x. Пусть количество чисел в массиве, больших x, равно k_1. Тогда на этой операции уменьшатся все элементы массива, большие x, а также $k - k_1$ элементов, равных x. Чтобы поддержать отсортированность массива, среди элементов, равных x, нужно выбрать $k - k_1$ из них, стоящих в массиве левее всего. Находить количество элементов, больших x, и первый элемент не меньший x, а также уменьшать все элементы на отрезке на 1 можно с помощью дерева отрезков за $O(\log n)$. Итого, получаем решение за $O(m \log n)$.

В оставшихся подзадачах так или иначе используется основная идея полного решения.

Основная идея: представим, что на каждой операции мы уменьшаем не k наибольших элементов, а любые k элементов по нашему выбору. Также будем считать, что операции применяются так, чтобы поддерживать отсортированность массива (обратите внимание, что это требование никак не ограничивает операции, если рассматривать их как операции над неупорядоченным набором элементов массива). Пусть b — массив, полученный применением m раз операции из условия, а c — массив, полученный применением произвольной последовательности из m операций, где каждая операция уменьшает k любых элементов на 1. Поймем, чем массив b, который мы хотим найти, выделяется среди всевозможных массивов c. Пусть массивы b и c отсортированы. Тогда утверждается, что $b_1 + \ldots + b_i \geqslant c_1 + \ldots + c_i$ для всех i от 1 до n, то есть операции из условия в каком-то

смысле «максимизируют» полученный массив. Отсюда также следует, что массив $b_1, b_2 \ldots, b_n$ является лексикографически максимальным массивом, который можно получить такими операциями. Доказательство: рассмотрим последовательность операций, приводящую к максимальной возможной сумме $c_1 + \ldots + c_t$. Посмотрим на последнюю плохую операцию (ту, которая уменьшала на 1 не k максимальных элементов). Тогда если элементы i, j, такие, что до этой операции $a_i < a_j$ ($i < j$, так как операции сохраняют порядок элементов), при этом i-й элемент был уменьшен на 1, а j-й не был. Пусть после применения всех операций массив стал равен $b_1, b_2 \ldots, b_n$. Если $b_i < b_j$, поменяв в рассматриваемой плохой операции i-й и j-й элемент местами мы получим в конце не меньшее значение $b_1 + b_2 + \ldots + b_t$. Если $b_i = b_j$, то после плохой операции была хотя бы одна, которая уменьшила a_j и не уменьшила a_i. Поменяем местами a_i и a_j в обеих операциях, получим тот же массив b, но при этом увеличится сумма элементов, которые были уменьшены на 1 в плохой операции. Количество операций и сумма элементов конечны, поэтому такой процесс когда-нибудь закончится, и мы получим последовтельность, состоящую из хороших операций и максимизирующую сумму $c_1 + \ldots + c_t$.

Мы поняли, что b максимизирует префиксные суммы по всем массивам c, получаемым с помощью m операций вычитания 1 из k элементов. Следующая задача позволяет легко описать все подходящие массивы c, по которым мы будем искать этот максимум.

Вспомним такую классическую задачу: как проверить, что в массиве $d_1, d_2, \ldots, d_n \geqslant 0$ можно m раз уменьшить k разных (но не обязательно различных по значению), элементов, так, чтобы все элемнеты массива остались неорицательными. Ответ: это возможно тогда и только тогда, когда $\min(d_1, m) + \ldots + \min(d_n, m) \geqslant mk$. Доказательство как обычно остается в качестве упражнения.

С помощью описанного решения классической задачи можно проверять, можно ли получить из a после выполнения m операций массив c, такой, что $c_i \geqslant b_i$ для всех i. Тогда сделав последовательно бинпоиск по b_1, потом по b_2 с фиксированным b_1 и так далее мы найдем максимальный лексикографически массив, который можно получить из a за m операций, а по факту выше он и будет ответом. В зависимости от эффективности реализации получается решение за $O(n^2 \log m)$, $O(n^2)$, $O(n \log m)$ или $O(n)$, проходящее подзадачу 7, и, возможно, 8.

Научимся считать максимальную возможную префиксную сумму $a_1 + a_2 + \ldots + a_i$, которую можно получить после m операций. Предположим, что мы уменьшили элемент a_i до x. Тогда элементы $a_1, a_2, \ldots, a_{i-1}$ должны быть уменьшены хотя бы до x, а элементы $a_{i+1}, \ldots, a_n$ могут быть уменьшены не ниже, чем до x, потому что порядок элементов в массиве должен сохраняться. Рассмотрим две величины $f(x)$ — минимальное число, на которое мы должны суммарно уменьшить элементы на префиксе до i-го, и $g(x)$ — максимальное число, на которые мы можем уменьшить элементы на суффиксе с $i+1$-го, чтобы сохранить неубывающий порядок элементов в массиве. Так как суммарно за m операций все элементы уменьшатся на mk, то на префиксе суммарное уменьшение будет не меньше $\max(f(x), mk - g(x))$. Легко видеть, что эта оценка достигается, поэтому в случае, когда мы уменьшаем a_i до x, максимальная возможная префиксная сумма равна $a_1 + a_2 + \ldots + a_i - \max(f(x), mk - g(x))$. Найти максимум такой функции по x можно с помощью бинарного поиска, так как функция $f(x)$ невозрастает, а функция $mk - g(x)$ неубывает. Без запросов изменения функции $f(x)$ и $g(x)$ можно считать с помощью префиксных сумм на отсортированном массиве, что проходит подзадачи 9 и 10. Для обработки запросов изменения можно использовать дерево отрезков, дерево Фенвика или декартово дерево, что проходит 11 и, в зависимости от эффективности реализации, 12 подзадачу. Время работы решения $O(n \log m \log n)$.