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.