Задача 1. Урок физкультуры

Авторы: Тихон Евтеев, Николай Будин
Разработчик: Мария Жогова

Подзадача 1

Сначала переберем пару элементов, расстояние между которыми мы хотим максимизировать. Потом переберем все возможные подотрезки массива, отсортируем, и обновим ответ получившимся расстоянием. Получили решение, работающее за $\mathcal{O}(n^5 \cdot \log n)$ или $\mathcal{O}(n^6)$, в зависимости от сортировки.

Здесь и далее заметим, что в некоторых случаях может оказаться выгоднее не передвигать элементы, поэтому при поиске максимума во всех подзадачах будем также учитывать начальное расстояние между элементами.

Подзадача 2

Сначала переберем границы отрезка, отсортируем. Затем в отсортированном массиве переберем все пары элементов и обновим ответы для них. Это решение работает за $\mathcal{O}(n^4)$.

Подзадача 3

Далее необходимо заметить, что для пары элементов на позициях l и r интересными являются только следующие действия:

- отсортировать $[1,\ n]$
- отсортировать $[1,\ l]$
- отсортировать $[r,\ n]$

Действительно, если отрезок сортировки содержит оба числа, то после сортировки расстояние между ними будет не больше $|a-b|$, где a и b — их значения. Если отрезок содержит только элемент на позиции l, то этот элемент нужно «подвинуть» как можно левее. Легко заметить, что наилучший результат достигается, если отсортировать отрезок $[1, l]$. И аналогично для случая, если отрезок содержит только элемент на позиции r.

Переберем пару элементов, проверим три варианта сортировки, получим решение за $\mathcal{O}(n^3 \cdot \log n)$.

Подзадача 4

Заметим, что нет необходимости сортировать отрезок. Пусть элемент на позиции l равен a, а элемент на позиции r — b. Ответ для этой пары элементов равен $\min(|a-b|, r-l+x[l], r-l+y[r])$, где $x[l]$ — количество элементов слева от l, которые больше a, а $y[r]$ — количество элементов правее r, которые меньше b.

Таким образом, ответ для пары элементов можно вычислить за $\mathcal{O}(n)$, а все решение будет работать за $\mathcal{O}(n^3)$.

Полное решение

Заметим, что массивы $x[l]$ и $y[r]$ можно предпосчитать за $\mathcal{O}(n^2)$, после чего вычисление ответа для пары элементов будет происходить за $\mathcal{O}(1)$. Таким образом, решение будет работать за $\mathcal{O}(n^2)$.

Задача 2. Оптимизация закупок

Автор: Азат Исмагилов
Разработчик: Рамазан Рахматуллин

Для решения задачи, воспользуемся методом динамического программирования по поддеревьям.

Пусть мы решили задачу для сыновей вершины v, то есть для каждого её сына u нам уже известны:

- минимальное количество комплектов оборудования a_u,
- их оптимальная суммарная стоимость w_u,
- список троек (c_o, e_o, o), обозначающих, что дополнительно можно купить не более e_o комплектов в вершине o стоимостью c_o, не нарушая условия ни в однов из узлов поддерева.

Положим $a_v = \sum_u a_u$. Если $a_v < l_v$, то нам необходимо добрать $l_v - a_v$ минимальных по стоимости элементов в поддереве вершины v. Кандидатами являются полученные тройки (c_o, e_o, o), а также тройка $(c_v, 10^9, v)$. Упорядочим тройки по возрастанию c_o и жадно наберем до необходимой суммы l_v, уменьшая e_o и удаляя тройки с занулившимся e_o. Теперь необходимо убедиться, что при передаче данных к предку вершины v, условие на r_v будет выполнено.

Для этого необходимо объединить списки троек по всем u, оставив только те тройки, которые не приведут к нарушению условия в вершине v. Так как из поддерева вершины v можно дополнительно купить не более $r_v - a_v$ комплектов, нужно удалить некоторое количество максимальных по стоимости тройек и уменьшить e_o в последней тройке так, чтобы оставшаяся сумма e_o не превышала $r_v - a_v$.

Для эффективной реализации этого алгоритма будем хранить тройки в упорядоченном множестве. Необходимо поддерживать операции объединения множеств, удаления минимума, удаления максимума. Используя метод приливания меньшего к большему, получаем решение, работающее за $O(n \log^2 n)$.

Задача 3. Интересные выходные

Автор: Тихон Евтеев
Разработчик: Дмитрий Умнов

Если рассмотреть граф, вершинами которого будут узлы, а ребрами — те трубы, по которым проехал Коля, то он будет деревом (например, потому что у каждого узла, кроме начального, есть ровно один предок).

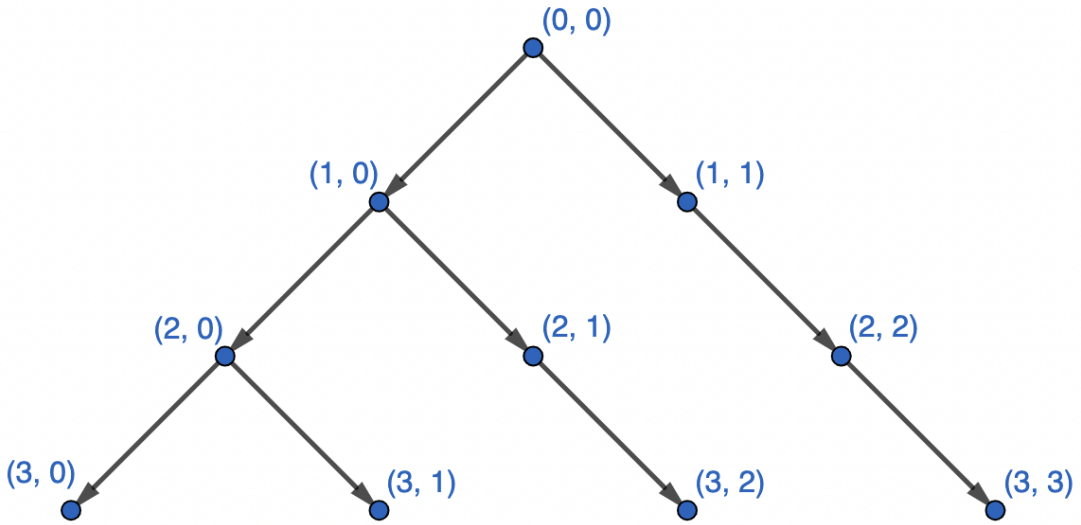
Также можно заметить, что запрос брата сводится к запросу поиска наименьшего общего предка (LCA) двух вершин.

Подзадачи 1 и 2

Чтобы решить задачу за $\mathcal{O}(n^2 + n \cdot q)$, достаточно явно построить описанное дерево за $\mathcal{O}(n^2)$, и на каждый запрос поиска LCA отвечать за $\mathcal{O}(n)$. Это можно сделать, например, поднимая первую вершину запроса по дереву до того момента, когда она станет предком второй вершины запроса. Это решение проходит первые две подзадачи и набирает 37 баллов.

Подзадача 3

Чтобы решить подзадачу 3, рассмотрим, как выглядит построенное дерево в этой подзадаче, (для массива 1, 2, 3, 4):

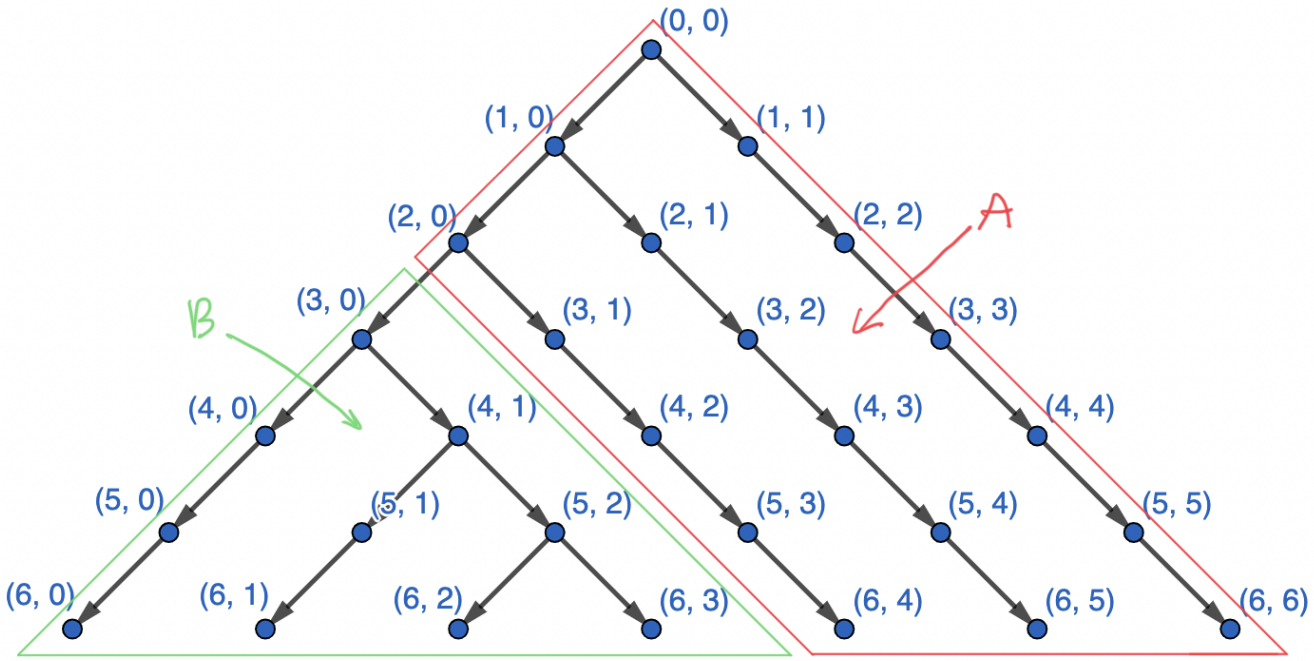


Найдем LCA вершин (r_1, c_1) и (r_2, c_2).

Если $r_1 - c_1 = r_2 - c_2$, то эти вершины лежат на одной ветке дерева, а значит их LCA это та вершина из запроса, которая находится выше. Иначе, если эти две вершины дерева находятся в разных ветках, то их LCA лежит в основании одной из веток, а значит его координаты равны $(\min(r_1 - c_1, r_2 - c_2), 0)$.

Подзадача 4

Аналогично рассмотрим, как выглядит дерево в подзадаче 4 (для массива 1, 2, 3, 6, 5, 4):



Видно, что дерево можно разделить на две части — A и B, где в части A будут находится k самых крайних веток. Для решения этой подзадачи нужно аккуратно рассмотреть все 3 возможных случая: обе вершины находятся в части A, обе вершины запроса находятся в части B, вершины находятся в разных частях. Все эти случаи можно разобрать за $\mathcal{O}(1)$.

Полное решение

Для дальнейшего решения разобьем задачу на две части:

1. Для каждого узла каждого запроса найдем ту инструкцию (одну из $n+1$), в момент которой брат проезжал этот узел.
2. Ответим на все запросы, используя информацию из предыдущей части.

Часть 1 можно решить за $\mathcal{O}((n+q)\log^2(n))$ с помощью параллельного бинпоиска и прохода с деревом Фенвика.

Затем, если нам нужно найти LCA двух вершин и мы знаем номера инструкций, в ходе которых эти вершины были посещены, то высотой (r-координатой) LCA будет длина наибольшего общего префикса этих двух инструкций, а c-координатой — количество команд «вниз-вправо» на этом общем префиксе.

Чтобы находить наибольший общий префикс двух команд i и j достаточно найти минимум в в массиве a на отрезке $[i, j)$. Этот минимум и будет равен длине наибольшего общего префикса. Так мы найдем r-координаты ответов.

Чтобы найти c-координаты ответов, зная r-координаты ответов, достаточно сделать один проход с деревом Фенвика.

Таким образом, мы получили решение за $\mathcal{O}((n+q)\log^2(n))$, которое набирает от 75 до 89 баллов в зависимости от реализации.

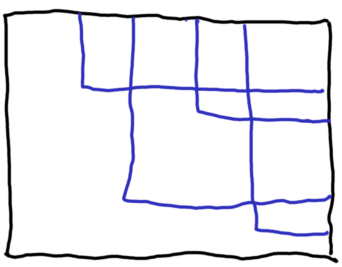
Чтобы получить решение за $\mathcal{O}((n+q)\log(n))$ нужно оптимизировать первую часть решения. Для этого можно сделать проход с деревом отрезков. Для каждого из $n+1$ уровней будем хранить смещение самого правого запроса на этом уровне относительно текущего маршрута. Тогда изменение маршрута сводится к операции вычитания на суффиксе, а в тот момент, когда смещение на каком-то уровне стало равно 0, мы можем определить номер инструкции для соответствующего запроса. Это решение набирает от 89 до 100 баллов в зависимости от реализации.

Задача 4. Прожекторы

Автор:	Николай Будин
Разработчики:	Николай Будин, Павел Маврин, Геннадий Короткевич, Иван Сафонов

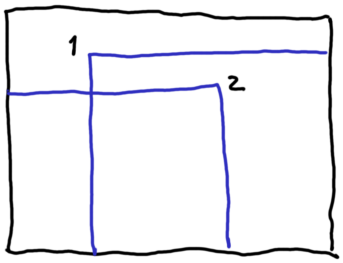
Разрешено только направление 1

Если разрешено только направление 1, у нас нет свободы выбора, в какую сторону ориентировать прожекторы. Поэтому, вся сложность подзадачи состоит в том, чтобы посчитать площадь объединения прямоугольников, освещённых этими прожекторами. Причем, у всех прямоугольников верхний-правый угол совпадает. Для этого достаточно отсортировать точки по x, идти в порядке сортировки и поддерживать минимум по y.

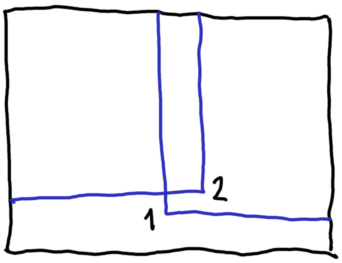


Разрешены все направления

Если $n \geqslant 4$, то оказывается, что проекторы могут осветить весь прямоугольник. Например, это можно сделать следующим образом. Рассмотрим два верхних прожектора. Направим левый из них в направлении 4, а правый — в направлении 3. Тогда они в объединении осветят полуплоскость ниже горизонтальной прямой, проходящей через верхний из них. Аналогично, две нижние точки могут осветить полуплоскость выше них.



(a) Две верхние точки

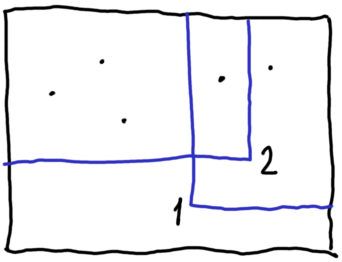


(b) Две нижние точки

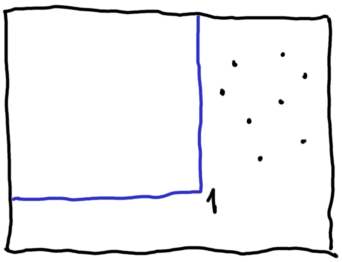
Иначе ($n < 4$), можно перебрать все варианты направлений за $4^n \leqslant 4^3$. И для каждого варианта вычислить площадь объединения. Чтобы вычислить площадь объединения, можно, например, воспользоваться сжатием координат, либо формулой включения-исключения.

Разрешены направления 1 и 2

Рассмотрим две самые нижние точки. Назовем их 1 и 2. Есть два варианта, в какую сторону ориентировать самую нижнюю точку. Если она покрывает вторую, то вторую нужно ориентировать в другую сторону. Тогда они в объединении покрывают все оставшиеся точки, и на этом процесс останавливается. Обновим ответ такой площадью.

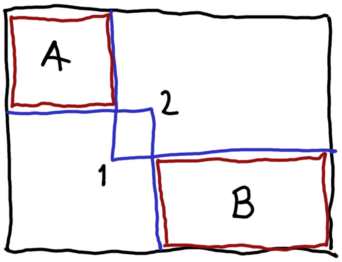


Иначе, мы фиксируем направление для самой нижней точки. Это сужает область по x, после чего мы переходим к задаче с $n - 1$ прожектором.

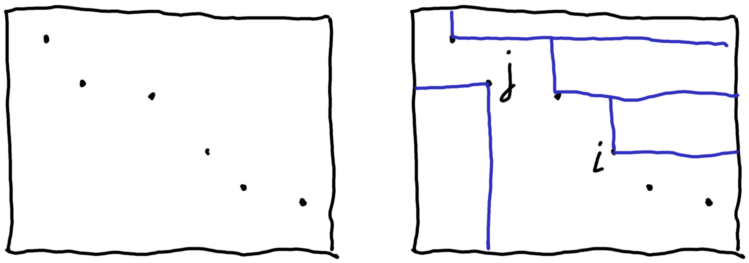


Разрешены направления 1 и 3

Если есть две точки, которые могут покрыть друг друга, их выгодно расположить в таких направлениях. После чего, задача разобьется на две независимых (области A и B на рисунке).



Если таких пар нет, то все точки расположены монотонно по x и y. В таком случае, для решения можно воспользоваться методом динамического программирования. Упорядочим точки по возрастанию x. Обозначим за $dp[i][j]$ максимальную площадь, которую могут освещать прожекторы до номера $\max(i, j)$ включительно, если последний прожектор в направлении 1 имеет номер i, а последний в направлении 3 — номер j. В таком случае, у нас будет $\mathcal{O}(n^2)$ состояний и $\mathcal{O}(1)$ переходов. Получится решение, работающее за $\mathcal{O}(n^2)$.



(a) Монотонная последовательность

(b) $dp[i][j]$

Есть два способа ускорения этого решения.

Первый способ

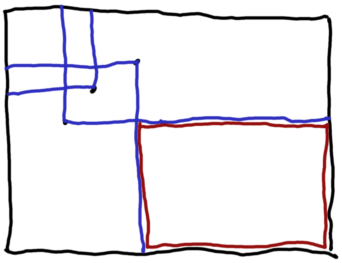
Можно доказать, что не более $\log_2(10^9) \approx 30$ подрядыдущих прожекторов будет светить в одном направлении. Поэтому, нам интересны только такие состояния $dp[i][j]$, в которых $|i - j| \leqslant 30$. Таким образом, получается решение, работающее за $\mathcal{O}(n \cdot \log_2(10^9))$.

Второй способ

Для оптимизации решения можно воспользоваться оптимизацией выпуклой оболочки (convex hull trick). Обозначим $dp_1[i] = dp[i][i-1]$, $dp_2[i] = dp[i-1][i]$. Тогда для вычисления $dp_1[i]$ (и $dp_2[i]$), нужно перебрать отрезок подрыдущих прожекторов, которые светят в одном направлении, до предыдушего момента, когда происходила смена направления. Если расписать и преобразовать формулу вычисления $dp_1[i]$ (и $dp_2[i]$), получится формула, в которой нужно выбирать минимум из нескольких линейных функций в какой-то точке. Это и позволяет сделать CHT оптимизация.

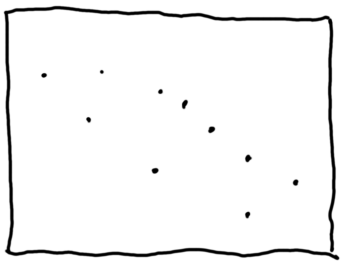
Разрешены направления 1, 2 и 3

Как и в предыдущем случае, сначала попробуем избавиться от разных конструкций и свести задачу к более простой. Если есть три точки, расположенные как на рисунке, их нужно ориентировать таким образом. Тогда останется область, выделенная красным.

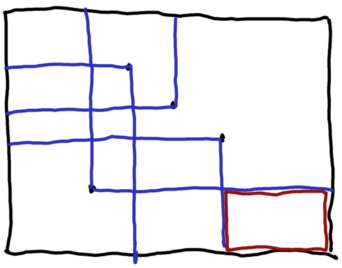


(a) Правило 1

Если такие тройки отсутствуют, то все тройки можно разбить на две монотонные последовательности. Заметим, что не имеет смысла два проектора из одной такой последовательности ориентировать в направлении 2. Поэтому, в направлении 2 будет ориентированно не более двух прожекторов из оставшихся. Рассмотрим прожектор, и все прожекторы справа сверху от него. Пусть таких прожекторов хотя бы 3. Они обязательно расположены монотонно (иначе была бы тройка, к которой мы можем применить правило 1). Тогда применим следующее правило, останется область, выделенная красным.



(a) Две монотонные последовательности



(b) Правило 2

Если мы удалили все такие четверки, у каждой точки есть не более двух, находящихся справа-сверху от нее. Следовательно, есть только $\mathcal{O}(n)$ вариантов пар прожекторов для которых нужно одновременно выбрать направление 2.

Теперь переберем ноль, один или два прожектора, которые будут светить в направлении 2. Остальные будут светить в направлениях 1 или 3. Воспользуемся решением с методом динамического программирования из решения для направлений 1 и 3. Может получится либо решение за $\mathcal{O}(n^3)$, либо за $\mathcal{O}(n^2 \cdot log_2(10^9))$.

Также, у жюри есть решение за $\mathcal{O}(n \log(n))$.

Задача 5. Максимизация выигрыша

Автор:	Николай Будин
Разработчик:	Даниил Орешников

Подзадача 1

Для решения первой подзадачи можно было просто перебрать все возможные перестановки цифр. По исходному числу и перестановке его цифр можно за время $\mathcal{O}(n^2)$ восстановить количество обменов соседних цифр, необходимое для получения такой перестановки. Например, можно найти в исходном числе первое вхождение цифры, стоящей на первом месте в перестановке, за $\mathcal{O}(n)$ обменов переставить ее на первое место, а затем повторить то же действие для оставшихся цифр. В итоге для каждой перестановки можно посчитать штраф, вычислить выигрыш и найти среди всех перестановок оптимальную.

Подзадача 2

Во второй подзадаче вместо перебора всех перестановок можно было воспользоваться методом динамического программирования. Состоянием динамики будет маска, кодирующая подмножество цифр исходного числа. Для каждой такой маски m будем в $\texttt{dp}[m]$ считать минимальный возможный штраф, необходимый для получения префикса, состоящего из соответствующего маске набора цифр. Переберем, какая цифра из набора будет последней на префиксе, тогда нам известны $\texttt{dp}[m_{prev}]$ — штраф за получения префикса без этой цифры, и количество обменов, необходимое для перемещения последней цифры набора на свое место. Его можно посчитать за $\mathcal{O}(n)$, для этого надо найти позицию цифры в исходном числе и количество цифр из набора m, которые были перемещены с позиций справа от нее налево.

Если в динамике так же запоминать саму цифру, которую выгодно ставить последней, то в конце можно восстановить и само оптимальное число — достаточно «собирать» его с конца, каждый раз переходя к маске без последней цифры. Такое решение работает за $\mathcal{O}(2^n \cdot n)$, если для маски предподсчитывать количество цифр, перемещенных на префикс с каждого суффикса (суффиксными суммами).

Подзадача 3

При $y = 1$ в третьей подзадаче можно заметить, что любой обмен соседних цифр, приводящий к увеличению x_{new}, выгоден, так как увеличивает результат больше, чем начисляет штраф. Таким образом, на самом деле достаточно отсортировать цифры числа по невозрастанию, и полученный x_{new} будет оптимальным.

Полное решение

Для решения следующих подзадач достаточно заметить следующий факт: штраф за перемещение большей цифры в разряд старше 9 всегда меньше, чем приращение результата. Действительно, чтобы поменять местами цифры в разрядах a и b, требуется совершить обмены на $(2a - 2b - 1) \cdot y$ штрафа, тогда как выгода будет равна $d \cdot (10^a - 10^b)$, где d — разность между цифрами в этих разрядах. При $y \leqslant 10^8$ суммарный штраф не превысит $2 \cdot 10^5 \cdot 10^8 = 2 \cdot 10^{13}$, тогда как второе выражение не меньше $9d \cdot 10^{a-1}$ и при $d > 0$ и $a > 13$ строго больше, чем $2 \cdot 10^{13}$.

Однако, для разрядов с 10 по 13 верно, что $2a - 2b - 1 < 30$, а значит оценку на штраф за максимизацию этих разрядов на самом деле можно уточнить: $(2a - 2b - 1) \cdot y < 3 \cdot 10^9 < 9d \cdot 10^{a-1}$.

Соответственно, во всех случаях выгодно максимизировать все цифры, кроме последних десяти. Отсортируем все цифры числа по невозрастанию, оставим префикс длины $n - 10$, а оставшиеся 10 цифр выпишем в порядке, в котором они следуют в исходном числе. Дальше, в четвертой подзадаче для максимизации этого суффикса было удобно применить полный перебор или динамику, описанные для первой и второй подзадачи. Пятую подзадачу, аналогично, можно было решить полным перебором перестановок суффикса длины 10, однако в ней было больше крайних случаев, так как были разрешены произвольные цифры.

В последней подзадаче не верна данная оценка на максимизируемый префикс, однако при $y \leqslant 10^{16}$ можно аналогичным образом доказать, что все разряды, начиная с разряда номер 16+5 = 21 будут заняты максимальными возможными цифрами, а для оставшихся разрядов достаточно применить решение с динамическим программированием из второй подгруппы.

Задача 6. Экспедиция на Сириус

Автор:	Тихон Евтеев
Разработчик:	Тихон Евтеев

Будем называть множество игроков с одинаковым опытом группой.

Отсортируем всех игроков по убыванию опыта и далее будем работать с ними только в таком порядке, так как они никогда не будут меняться в порядке сортировки.

Подзадача 1

Чтобы решить первую подзадачу, достаточно отсортировать все вопросы и за $O(n \cdot \max(k_i))$ просимулировать процесс.

Вопросы первого типа

Чтобы лучше понять механику, описаную в задаче, разберемся с вопросами первого типа. Посмотрим на две соседних группы: вплоть до момента, когда опыт игроков в этих группах сравняется, первые будут получать ровно на одну единицу меньше, чем вторые (конкретные величины могут меняться, но это соотношение останется).

Из этого наблюдения следует вывод, что если опыт в двух соседних группах соответственно c_1 и c_2, то граница между ними исчезнет через $|c_1 - c_2|$ раундов. На самом деле это утверждение верно и для случая, когда у 2-х игроков изначально равный опыт. «Граница» между ними исчезает через 0 раундов. Выпишем все изначальные разности соседних игроков в списке (в том числе те, что равны 0). Ответ на вопрос 1-го типа для k раундов $n - cnt(k)$, где $cnt(k)$ — это количество чисел в этом списке, меньше либо равных k. Такое число можно найти, например, бинарным поиском.

Решение для $n \leqslant 5000$

Назовем «интересными» моменты времени, когда какие-то 2 группы объединяются. В промежутках между соседними интересными моментами количество групп не меняется, игроки получают фиксированный суммарный опыт и каждый раунд каждый человек получает фиксированный опыт. Отсортируем все «интересные моменты», а так же все моменты из вопросов в один список по увеличению времени.

Теперь мы легко можем решить подзадачи, в которых $n \leqslant 5000$. На каждом интересном моменте мы будем заново пересчитывать опыт каждого игрока, сколько опыта каждый человек будет получать каждый раунд до следующего интересного момента, и сколько суммарно все игроки будут получать опыта каждый раунд. Если не пересчитывать это на моментах вопросов, то получим решение за $O(n^2 + q \cdot \log(q))$, в противном случае за $O((n + q) \cdot n)$.

Если пересчитывая, сколько каждый игрок будет получать в промежутке до следующего интересного момента, мы будем перебирать только группы (так как внутри группы все будут получать одинаково), мы получим решение за $O((n + q) \cdot \log(n + q) + \max(c_i))$, так как каждое перемещение к другой группе внутри одного пересчета сокращает разрыв между игроком с самым большим опытом и игроком с самым маленьким опытом на 1, а следовательно таких перемещений будет не больше, чем изначальный разрыв. Для этого решения понадобится структура данных «система непересекающихся множеств» (СНМ, англ. DSU) или другая, выполняющая её функции.

Вопросы второго типа

Научимся теперь отвечать на вопросы 2-го типа. Для этого, нужно понять, насколько уменьшается суммарный пророст опыта за раунд в данном «интересном» моменте. Если в этом интересном моменте стирается граниица между группами, последний представитель одной из которых находится на позиции i, а первый другой — на $i + 1$, то суммарный прирост уменьшится на $n - i - 1$ за раунд (в 0-индексации).

Вопросы третьего типа

Чтобы наконец быстро отвечать и на вопросы 3-го типа, нужно научиться быстро пересчитывать опыт и «прирост» каждого человека в интересных моментах. Как уже было замечено, если стирается граница между позициями i и $i + 1$, то все игроки, начиная с $i + 1$ начинают получать на 1 опыт за раунд меньше. Такую операцию (вычитание на суффиксе) можно легко поддерживать, используя дерево Фенвика или дерево отрезков. Если мы просто уменьшим «прирост» каждого человека на суффиксе, то считая его опыт в будущем мы не учтем всё, что он набрал, пока имел более высокий прирост. Чтобы это исправить, добавим к опыту каждого человека на суффиксе t_i опыта (где t_i -

текущий интересный момент). Тогда чтобы посчитать опыт данного человека, достаточно взять то, что у него лежит в массиве опыта, и добавить $t \cdot add$, где add — его текущий прирост, а t — номер раунда.

Задача 7. Тяжелый груз

Автор: Тихон Евтеев
Разработчик: Иван Сафонов

Будем рассматривать склад как граф, где комнаты это вершины, а коридоры это ребра.
Заметим, что весь процесс перемещения коробки состоит из следующих циклов:

- Погрузчик стоит вместе с коробкой в вершине x. Он кладет коробку в вершину i.
- Погрузчик перемещается из вершины x в вершину y по некоторому пути, не проходящему через вершину i.
- Погрузчик поднимает коробку из вершины i и оказывается в вершине y с поднятой коробкой.

Этот цикл действий занимает две единицы времени. Необходимые условия:

- Вершины x, i соединены ребром. Обозначим это ребро за e_{xi}.
- Вершины y, i соединены ребром. Обозначим это ребро за e_{yi}.
- Существует путь между x, y, не проходящий через вершину i.

Построим граф, в котором соединим ребром все возможные пары (x, y), для которых существует подходящая вершина i. Тогда кратчайшие расстояния из вершины 1 этого графа до остальных вершин, умноженные на 2, это ответы на задачу.

Подзадача 1

В первой подзадаче можно перебрать ребро e_{xi}, затем с помощью dfs найти все достижимые вершины y из вершины x, если нельзя заходить в вершину i. Так можно найти все возможные пары (x, y), после чего в полученном графе найти кратчайшие расстояния. Время работы решения $O(m(n+m)+n^2) = O(m^2)$.

Подзадача 2

Заметим, что последнее условие на вершины (x, y) можно переформулировать так: после удаления вершины i из графа, вершины x, y находятся в одной компоненте связности. Во второй подзадаче можно перебрать вершину i, удалить ее из графа и с помощью dfs найти в оставшемся графе все компоненты связности. После этого пары вершин x, y, соединенных с i ребром и оказавшихся в одной компоненте связности нужно соединить в графе для поиска кратчайших расстояний. Чтобы количество ребер этого графа было $O(n^2)$ для удаления кратных ребер удобнее использовать матрицу смежности. Время работы решения $O(n(n+m)+n^2) = O(nm)$.

Для первых двух подзадач можно использовать альтернативное решение: рассмотрим граф, в котором вершинами будут пары вершин (a, b) — где стоит погрузчик, где находится коробка. В этом графе можно с помощью алгоритма 01-bfs найти кратчайшие расстояния от вершины $(1, 1)$ до вершин (p, p). Несложно понять, что количество вершин в таком графе на парах вершин будет $n(d_1 + d_2 + \ldots + d_n) = O(nm)$ (d_i это степени вершин). Время работы решения $O(nm)$.

Подзадача 3

В третьей подзадаче можно ускорить решение второй подзадачи с помощью bitset. Для поиска компонент связности графа без вершины i можно оставить только некоторые ребра: ребра дерева dfs всего графа, для каждой вершины внешнее ребро, которое выходит из нее и ведет в наименее глубокую вершину. Таких ребер $O(n)$. Поэтому поиск компонент связности будет работать за $O(n)$. Нахождение всех пар верших (x, y) после этого можно сделать с помощью bitset, поэтому эта часть решения будет работать за $O(\frac{nm}{64})$. Наконец, после этого, bfs на матрице смежности будет работать за $O(n^2)$. Время работы решения $O(n^2 + \frac{nm}{64})$.

Подзадача 4

В четвертой подзадаче всегда можно добраться от x до y, не проходя через вершину i (например, по большому циклу). Значит нам нужно найти кратчайшие расстояния в графе, в котором соединены пары вершин (x, y), такие что существует путь длины 2 от x до y. Рассмотрим граф (t, c), где t вершина нашего графа, а $c \in \{0, 1\}$ (она будет обозначать «четность»). Каждое ребро (a, b) обычного графа добавляет два ребра между $(a, 0)$ и $(b, 1)$, а также между $(a, 1)$ и $(b, 0)$. Теперь нам нужно найти кратчайшие расстояния в этом графе от вершины $(1, 0)$ до всех вершин $(p, 0)$, что можно сделать, потому что количество ребер нового графа $2m$. Время работы решения $O(m)$.

Можно заметить, что нас интересуют компоненты связности после удаления вершин i, которые являются точками сочленения (иначе компонента точно будет одна). Также заметим, что если в графе есть мост, то мы через него точно не сможем пройти и дойти до всех вершин, которые будут в другой компоненте связности с вершиной 1 после удаления этого моста.

Подзадача 5

В пятой подзадаче найдем все мосты и выкинем части графа, до которых нужно пройти через мост. Можно показать, что в оставшейся части графа нет точек сочленения (так как из каждой точки сочленения в графе со степенями $\leqslant 3$ выходит хотя бы один мост). Значит на оставшейся части графа можно использовать решение для предыдущей подзадачи. Время работы решения $O(m)$.

Подзадача 6

Заметим, что условие того, что от вершины x можно дойти до вершины y, не проходя через вершину i, равносильно тому, что ребра e_{xi} и e_{yi} лежат в одной компоненте вершинной двусвязности. Найдем все компоненты вершинной двусвязности. Для каждой вершины i рассмотрим концы ребер $a_1, a_2, \ldots, a_k$, такие что ребра $e_{a_1 i}, e_{a_2 i}, \ldots, e_{a_k i}$ лежат в одной компонете вершинной двусвязности. Тогда все пары концов (a_s, a_f) нужно соединить в графе для поиска кратчайших расстояний. Сделаем двудольный граф, где в левой доле будет n вершин изначального графа, а в правой доле для каждой вершины i и каждого ее набора $\{a_1, a_2, \ldots, a_k\}$ создадим вершину, которую соединим с вершинами $a_1, a_2, \ldots, a_k$ левой доли. Тогда кратчайшие расстояния от вершины 1 левой доли до всех вершин p левой доли будут ответами. Время работы решения $O(m)$.

Задача 8. Большие вызовы

Авторы:	Рамазан Рахматуллин, Николай Калинин
Разработчики:	Рамазан Рахматуллин, Николай Калинин

Переформулируем задачу. Дан массив a длины n, m отрезков типов 0 и 1, у каждого отрезка есть число c_i. Фиксируя x, отрезки типа 1 расширяются до x. После этого, строится двудольная сеть, из отрезка проводятся бесконечные ребра в элементы массива на этом отрезке. У отрезков пропускная способность равна c_i, у элементов массива пропускная способность равна a_i. Необходимо найти размер максимального потока для всех x.

Для начала научимся искать поток в таком графе эффективно, если отрезки не двигаются. Будем набирать поток жадно, идя слева направо по массиву. В каждый момент времени у нас есть множество открытых отрезков, покрывающих данную позицию i. Нужно выбрать, какой из

отрезков сопоставлять с i. Легко понять, что выгоднее всего выбирать отрезок, заканчивающийся раньше всего. После выбора отрезка, нужно уменьшить пропускную способность у элемента массива и у отрезка, удалив таким образом одного из них. Также необходимо открывать и закрывать отрезки в соответствующие моменты времени, если пропускная способность не уменьшилась до 0. Используя структуру данных упорядоченное множество, это решение работает за $O((n+m)\log m)$. Используя это решение для каждого x независимо, получим время работы $O(n\cdot(n+m)\log m)$.

Заметим, что если есть отрезок массива, внутри которого не начинаются и не заканчиваются отрезки, то этот отрезок можно сжать до одного элемента. Так как концов отрезков суммарно $O(m)$, можно предварительно сжать координаты, получив решение $O(nm\log m)$.

Опишем решение в случае, когда нет отрезков типа 0. Зафиксируем x, применим лемму Холла о существовании совершенного паросочетания, накрывающего массив a. Напомним, что лемма Холла в данной формулировке звучит так:

- Для любого множества элементов M_a массива a с суммой X, определим Y как сумму c_i по всем отрезкам M_b, накрывающим хотя бы один элемент из a. В графе существует совершенное паросочетание, накрывающее массив a тогда и только тогда, когда $X \leqslant Y$ для любого M_a.

Здесь в качестве совершенного паросочетания имеется в виду поток размера $\sum a_i$ в описанной сети. Так как лемма Холла дает критерий для совершенного паросочетания, нужно его изменить для получения значения максимального паросочетания. Добавим отрезок $[1;n]$ со значением b. Лемма Холла утверждает, что паросочетание существует, если $X \leqslant Y+b$ для любого M_a. Переформулируя, необходимо найти минимальное $b \geqslant 0$ такое, что для всех подмножеств M_a массива a выполнено $X \leqslant Y+b$, тогда размером максимального паросочетания будет $\sum a_i - b$. Преобразовав, получаем $b \geqslant X-Y$, то есть необходимо найти максимум $X-Y$ по всем M_a.

Зафиксируем M_a, пусть $l \leqslant x$ — самый правый взятый элемент левее x включительно, $r \geqslant x$ — самый левый взятый элемент правее x включительно. Заметим, что M_b состоит из отрезков, у которых левая граница лежит левее l включительно или правая граница лежит правее r включительно. Добавив в множество M_a все элементы левее l или правее r получим, что X увеличилось, а Y не изменилось. Раз мы максимизируем $X-Y$, достаточно рассматривать только множества a, являющиеся объединением префикса $1\dots l$ и суффикса $r\dots n$, причем $l \leqslant x$ и $r \geqslant x$.

Для таких множеств мы можем эффективно записать значение $X-Y$. Пусть $A=\sum a_i$, $B=\sum c_i$. Так как $X=\sum_{i=1}^{l} a_i + \sum_{i=r}^{n} a_i = A - \sum_{i=l+1}^{r-1} a_i$, а $Y=\sum_{i=1}^{m} c_i - b_{[l+1;r-1]} = B - b_{[l+1;r-1]}$, где $b_{[l;r]}$ — сумма c_i по отрезкам, лежащим внутри отрезка $[l;r]$, то $X-Y = A - \sum_{i=l+1}^{r-1} a_i - B + b_{[l+1;r-1]}$. Так как A и B константы, необходимо максимизировать $b_{[l;r]} - \sum_{i=l}^{r} a_i$ ($l+1$ заменено на l, $r-1$ заменено на r).

Обозначим g_x за максимум $X-Y$ по всем отрезкам $[l;r]$. Изначально скажем $g_x = -\infty$. Заметим, что для обновления g_x рассматриваются только отрезки, включающие x. Обозначим $f(l,r) = b_{[l;r]} - \sum_{i=l}^{r} a_i$ для исходных отрезков, которые не расширялись. Аналогично обозначим $f_x(l,r)$ как та же функция, но для расширенных до x отрезков. Заметим, что $f(l,r) = f_x(l,r)$ для $l \leqslant x \leqslant r$. Так как g_x обновляется только для $l \leqslant x \leqslant r$, получаем следующую формулировку: g_x равно максимуму $f(l,r)$ по всем $l \leqslant x \leqslant r$. Эквивалентно, необходимо для всех отрезков $[l;r]$ обновить g_x через $f(l,r)$ для $l \leqslant x \leqslant r$.

Данная задача решается методом разделяй и властвуй. Запустим функцию от отрезка $[1;n]$. Каждый раз мы получаем отрезок $[L;R]$. Предположим, что он не единичной длины. Пусть $M=\frac{L+R}{2}$. Обновим ответ от всех отрезков, которые содержатся в $[L;R]$ и содержат M и $M+1$, после чего вызовем функцию от $[L;M]$ и $[M+1;R]$. Задача свелась к тому, чтобы обновить g_x для всех отрезков $[l;r]$ при условии $L \leqslant l \leqslant M$ и $M+1 \leqslant r \leqslant R$. Сделаем это независимо для $x \leqslant M$ и $x \geqslant M+1$, рассматриваем $x \leqslant M$.

Будем двигать l от M до L. Найти максимум $f(l, M+1 \leqslant r \leqslant R)$ можно с помощью структуры данных дерево отрезков. При движении l на один налево необходимо поддерживать операции прибавления на отрезке и максимум на отрезке. Таким образом, мы получим массив $w_L,\dots,w_M$. Далее необходимо обновить g_x через $w_{i\leqslant x}$. Это делается легко, если перебирать i слева направо и хранить максимум на префиксе. Данное решение работает за $O((n+m)\log^2 n)$. Это решение можно ускорить до $O(n\log^2 n + m\log n)$ если выполнять функцию в следующем порядке: вызов $[M+1,R]$,

пересчет отрезков $[L; R]$, вызов $[L; M]$. В таком случае не нужно отменять операции прибавления. Данное решение также можно ускорить до $O((n+m)\log n)$ если сделать так, чтобы отрезки вершин дерева отрезков совпадали с отрезками рекурсивной функции.

Это решение легко обобщить на случай отрезков обоих типов. Пусть $f_0(l, r)$ — упомянутая функция если рассматривать только отрезки типа 0. Посчитаем p_i — максимум суммы $f_0(l, r)$ по непересекающимся отрезкам на префиксе длины i массива a. Аналогично s_i — максимум суммы $f_0(l, r)$ на суффиксе до элемента i. Утверждается, что необходимо обновить g_x через $p_{l-1} + s_{r+1} + f(l, r)$ по всем $l \leqslant x \leqslant r$. Этот факт следует, если снова применить лемму Холла и посмотреть на интересные множества M_a. Данную модификацию легко добавить в предыдущее решение, так как мы перебираем один из концов отрезка, а второй храним в дереве отрезков.

Отметим, что эквивалентом леммы Холла в данном случае является теорема о минимальном разрезе, то есть данное решение можно было придумать другим способом.

Альтернативным решением задачи было следующее: будем пересчитывать ответ для $x+1$ через ответ для x. Для начала также решим случай $t_i = 1$. Утверждается, что слева от x можно применить жадное решение за $O(n \log n)$, которое хранит множество открытых отрезков. При движении x это жадное решение легко поддерживать, ведь обновляется только паросочетание для позиции x.

Теперь заметим, что справа от x все отрезки начинаются левее $x+1$, так как все отрезки расширяются до x. Так как часть массива до x нас не интересует, можно сказать, что все отрезки начинаются в x. Такой специальный случай, когда все отрезки начинаются в начале массива, можно решить с помощью дерева отрезков. Для решения можно также применить лемму Холла, либо же хранить функцию ответа $f(x)$ в вершине дерева отрезков, которая принимает число x — сумма непокрытых элементов массива a слева от отрезка и справа от x.

Это решение обобщается на случай отрезков типа 0, однако его мы описывать не будем. Идея заключается в том, что нужно научиться поддерживать жадное решение слева и разобраться с отрезками типа 0 справа. Для отрезков типа 0 справа можно запустить жадное решение справа налево и сохранить полученное паросочетание. Пересчитывать такое решение можно при движении x влево, движение x вправо моделируется откатом таких действий. Для отрезков типа 0 слева необходимо поддерживать отмену паросочетания из-за смены приоритета в жадном решении между отрезками типа 0 и 1, амортизированная оценка получится $O(n)$.