Задача A. Нестандартный подход

Автор задачи и разработчик: Ильнур Валеев

Формализуем задачу. Требуется найти клетку (x, y) на границе поля, такую, что максимальное расстояние от неё до любой другой клетки поля минимально. Расстояние между клетками определяется как минимальное количество шагов, необходимых для перемещения из одной клетки в другую, где за один шаг можно перейти в соседнюю клетку по стороне.

Подзадача 1

Если $n = 1$ (поле представляет собой одну строку), то оптимальная клетка старта — это середина строки. Координаты старта будут равны $\left(1, \left\lfloor \frac{m+1}{2} \right\rfloor\right)$.

Подзадача 2

В данной подзадаче без разницы, какой край выбирать, так как $n = m$. По той же логике, что и в первой подзадаче, следует выбирать середину стороны, так как самая удаленная клетка — это один из углов на противоположной стороне. Координаты старта тогда будут равны $\left(1, \left\lfloor \frac{m+1}{2} \right\rfloor\right)$.

Подзадачи 3 – 4

Для полей небольшого размера ($n, m \leqslant 100$) можно перебрать все возможные стартовые клетки на границе и для каждой из них промоделировать распространение тумана, вычислив максимальное время заполнения поля. Затем выбрать клетку с минимальным временем. Когда $n, m \leqslant 40$, можно было также перебрать вообще все клетки на поле, но при обновлении ответа отдельным условием проверить, что текущая клетка находится на границе.

Подзадача 5

Для больших полей ($n, m \leqslant 1000$) моделирование становится невозможным из-за ограничений по времени. Вместо этого можно вывести формулу для вычисления времени заполнения поля. Как мы раньше заметили, туман дольше всего добирается до одного из углов поля. Поэтому время заполнения поля из клетки (x, y) можно вычислить как: $\max(x - 1, n - x) + \max(y - 1, m - y)$.

Здесь:

- $\max(x - 1, n - x)$ — время, за которое туман доберётся до самой дальней строки.
- $\max(y - 1, m - y)$ — время, за которое туман доберётся до самого дальнего столбца.

Затем, пользуясь этой формулой, достаточно было перебрать все клетки на границе и обновить ответ.

Полное решение

Для минимизации времени заполнения поля достаточно рассмотреть две кандидатные точки:

1. Точка на середине верхней границы: $\left(1, \left\lfloor \frac{m+1}{2} \right\rfloor\right)$.
2. Точка на середине левой границы: $\left(\left\lfloor \frac{n+1}{2} \right\rfloor, 1\right)$.

Вычислим время заполнения для каждой из этих точек по формуле выше и выберем точку с минимальным временем. Либо, еще проще, заметим, что оптимум достигается в середине наиболее длинной из двух сторон. В любом случае ответ находится за время $\mathcal{O}(1)$.

Задача B. Спасти Лютика!

Автор задачи и разработчик: Даниил Голов

В этой задаче было необходимо аккуратно проэмулировать процесс, описанный в условии. Сложность заключалась в том, что прямолинейная симуляция процесса с обработкой каждого бандита или каждого холла набирала не полный балл.

Подзадача 1

В первой подзадаче как раз работала самая простая симуляция процесса. Будем хранить массив x в явном виде, и каждую секунду для каждого из k новых бандитов находить его место в подземелье, перебирая все холлы от выхода ко входу. Работает это за $\mathcal{O}(nks)$, чего с запасом хватает при данных ограничениях. Любая менее эффективная симуляция тоже должна была получить баллы за эту подзадачу.

Подзадача 2

Здесь $e = 0$, а значит бандиты никогда не исчезают, и только по одному (так как $k = 1$) приходят в подземелье. В таком случае ответ можно дать за $\mathcal{O}(1)$ — первые $\min\left(n, \left\lfloor \frac{s}{m} \right\rfloor\right)$ холлов будут заполнены полностью, в следующем (если он есть) будет $s \bmod m$ бандитов, а остальные будут пустыми. Достаточно только понять, что больше — $\left\lfloor \frac{s}{m} \right\rfloor$ или n, и насколько.

Подзадача 3

Немного обратная ситуация создается при $m = 1$. Если $e = 0$, этот случай мы рассмотрели выше. А иначе в начале каждой минуты подземелье будет целиком очищаться, так как $e \geqslant m$. Поэтому в конце каждой минуты состояние подземелья будет одинаковым, и остается только вычислить ответ для него.

Это тоже несложно сделать — приходят k новых бандитов, каждый занимает свой холл от 1-го до k-го. Ответ тогда зависит только от отношения между k и n — либо $(k+1)$-й холл существует, и ответ будет $(k+1, 0)$, так как этот холл пуст, либо холла нет, и тогда ответ $(1, 1)$, так как бандитов везде поровну.

Подзадача 4

Случай $m \leqslant e \leqslant k$ похож на третью подзадачу тем фактом, что подземелье очищается в начале каждой минуты. Остальное решение основывается на той же логике: найдем, в каком состоянии будет подземелье после первой минуты, и оно будет повторяться каждую следующую минуту.

Условие $k \geqslant m$ гарантирует, что хотя бы первый холл будет заполнен. А по факту первые $t = \left\lfloor \frac{k}{m} \right\rfloor$ холлов будут заполнены, в следующем будет $k \bmod m$ бандитов, а дальше все холлы будут пустые. В итоге мы снова свели задачу к двум проверкам числа холлов: существует ли $(t+2)$-й холл и существует ли $(t+1)$-й. Ответ же будет либо $(t+2, 0)$, либо $(t+1, k \bmod m)$, либо $(1, m)$.

Подзадача 5

Решение для этой подзадачи уже описано в первой подзадаче: простая симуляция процесса за время $\mathcal{O}(nks)$. Единственный тонкий момент — k может быть большим, а m ограничено сверху значением 100. Но при $n, m \leqslant 100$ можно сразу ограничить число бандитов, которое сможет зайти, числом 10 000. И если этого было недостаточно для прохождения лимита по времени, можно было также соптимизировать симуляцию, опираясь на то, что при достаточно больших k состояния подземелья быстро зациклятся.

Подзадача 6

Здесь требовалась уже более аккуратная симуляция: будем все также поддерживать последовательность x, но новых бандитов будем обрабатывать не по одному, а сразу группами. Чтобы обработать первую фазу каждой минуты, по-честному выполним присвоение $x_i \leftarrow \max(0, x_i - e)$ для каждого i. А затем, чтобы добавить бандитов в подземелье, пройдемся по холлам от выхода ко входу и выполним $x_i \leftarrow x_i + \min(m - x_i, t)$, где t — число еще не распределенных бандитов у входа.

В итоге получаем симуляцию, которая работает за время $\mathcal{O}(ns)$.

Дальше ее несложно соптимизировать до решения, которое проходит оставшиеся две группы. Для этого необходимо заметить два полезных факта:

1. Число бандитов в каждом следующем по номеру холле не больше, чем в предыдущем. Это несложно показать на индукции: операция $-e$ к каждому числу не нарушает свойство монотонности, а операция добавления новых k бандитов увеличивает x в порядке увеличения номера холла.

2. Поведение x нетривиально в двух случаях: $e \leqslant k \leqslant m$ и $e \leqslant m \leqslant k$. Остальные случаи либо тривиальны, либо уже разобраны выше.

Полное решение

В случае $e \leqslant k \leqslant m$ в первый холл приходит больше бандитов, чем исчезают в начале каждой минуты, поэтому он будет постепенно заполняться. К моменту, как он заполнится, начнется периодичное изменение x_1 между m и $m - e$ (все сраженные в начале минуты будут к концу минуты заменяться e из k новых).

Посмотрим на второй холл и заметим, что с ним будет происходить то же самое — сначала он будет постепенно заполняться, а потом чередоваться между m и $m - e$ бандитами в нем. И так с каждым следующим холлом до тех пор, пока вычитаемая величина не превысит прибавляемую. Это случится примерно, когда в $t \geqslant \left\lceil \frac{k}{e} \right\rceil$ холлах станет хотя бы по e бандитов.

И такие наблюдения позволяют нам написать решение за $\mathcal{O}(s)$ времени для этого случая: просто будем поддерживать число холлов, которые уже перешли в циклические переходы между m и $m-e$ и обновлять по-честному только следующий холл. Если «заполненных холлов» уже t, то в конце очередной минуты надо к x_{t+1} прибавить $k - et$.

В случае же $e \leqslant m \leqslant k$ все похоже на описанный выше случай с тем отличием, что за одну минуту заполняется больше одного холла. Это тоже несложно обработать в симуляции: если $k = r \cdot m + q$, где $q < m$, то достаточно повторить всю описанную выше логику, но дополнительно увеличить t на r и прибавить оставшиеся q уже к новому x_t.

Отличие между последними двумя подзадачами в том, что в восьмой достаточно в явном виде хранить массив x, который имеет длину n, а для полного решения можно было вообще массив не хранить и обойтись только счетчиком t и информацией о том, насколько заполнен последний холл, который еще не перешел в фазу чередования m и $m - e$ бандитов.

Задача C. Полное прохождение

Автор задачи и разработчик: Владислав Власов

В задаче по сути нам дан маршрут из n точек, который мы начинаем из первой в момент времени 0. Дойдя до i-й точки, мы можем оставаться в ней на любое время или переходить к следующей (если она есть) за d_{i+1} времени. Если в момент a_i мы находимся в точке i, то можем моментально посмотреть стрим. Нужно пройти весь маршрут, посмотрев максимальное количество стримов. Заранее заметим, что рассматривать $a_i < \sum_{j=0}^{i} d_i$ не нужно, так как к этим стримам невозможно успеть.

Подзадача 1

Если $n \leqslant 10$, то существует всего 2^n различных подмножеств стримов, которые мы можем выбрать. С помощью перебора двоичных масок найдем все возможные множества. Чтобы проверить, можно ли посмотреть все стримы в множестве, просимулируем процесс: будем дожидаться стрима в i-й точке, если i-й бит маски единичный (если нет, сразу перейдем к следующей). Если мы не успеваем в i-ю точку к моменту a_i, то посмотреть все стримы из данного множества невозможно. Среди всех множеств, которые подошли, выберем максимальное по количеству элементов.

Подзадача 2

В этой подзадаче массив a отсортирован, $a_i - a_{i-1} \leqslant 1$ и все $d_i = 1$. То есть, чтобы добраться до следующей точки, всегда требуется одна единица времени. Тогда расстояние между двумя точками i и $j > i$ всегда равно $j - i$.

Если мы посмотрели стрим в точке i, а

- $a_i \leqslant a_{i+1}$, то мы не сможем посмотреть $(i+1)$-й стрим и все последующие (так как время на дорогу займет не меньше, чем разница во времени между стримами);
- $a_i = a_{i+1}$, то мы сможем посмотреть $(i+1)$-й стрим без дополнительных затрат по времени.

Таким образом остается только брать непрерывные отрезки, в которых $a_{i+1} - a_i = 1$, максимальный из которых можно найти за $\mathcal{O}(n)$.

Подзадача 3

В этой подзадаче $n \leqslant 1000$, что позволяет нам решать задачу за $\mathcal{O}(n^2)$. Применим динамическое программирование: пусть $\texttt{dp}[i]$ — максимальное количество посмотренных стримов, если мы посмотрели i-й последним.

Переберем предыдущий стрим j и попробуем посмотреть i после него. Мы сможем это сделать, если расстояние между i и j не больше чем $a_i - a_j$: из всех подходящих j выберем самое большое $\texttt{dp}[j]$ и присвоим $\texttt{dp}[i] \leftarrow \texttt{dp}[j] + 1$. Если перебирать j в убывающем порядке, то расстояние до j можно поддержать за $\mathcal{O}(1)$ (либо просто воспользоваться префиксными суммами по d). Итоговым ответом будет максимальное значение в массиве $\texttt{dp}$.

Подзадача 4

Для полного решения воспользуемся идеей из прошлой подзадачи. Для этого рассмотрим переходы: они были возможны, если $a_i - a_j \geqslant \texttt{dist}(i,j)$. Представим $\texttt{dist}(i,j)$ как $\texttt{dist}(1,i) - \texttt{dist}(1,j)$, тогда $a_i - a_j \geqslant \texttt{dist}(1,i) - \texttt{dist}(1,j)$, что эквивалентно

$$a_i - \texttt{dist}(1,i) \geqslant a_j - \texttt{dist}(1,j).$$

Выходит, мы хотим выбрать как можно больше элементов, чтобы величина $a_i - \texttt{dist}(1,i)$ не убывала для всей подпоследовательности. Это буквально наибольшая неубывающая подпоследовательность массива $a_i - dist(1,i)$, поиск которой за $\mathcal{O}(n \log n)$ является стандартной задачей.

Задача D. Не доверяйте свиткам

Автор задачи и разработчик: Даниил Орешников

Для начала разберем отдельные особые решения, которые нельзя было соптимизировать до полного решения. Первую подзадачу в разборе пропустим, так как в ней достаточно было разобрать очень небольшое число возможных случаев того, какие элементы a', b' и c' соответствуют каким элементам a, b и c.

Также пропустим подробный разбор девятой подгруппу: при $a_i = 0$ массивы b и c восстанавливаются однозначно без дополнительной информации (b уменьшается $\iff$ c увеличивается, и наоборот). Поэтому достаточно взять $b = b'$ и $c = c'$, после чего при необходимости дополнить их последними элементами до длины n.

Решение второй подзадачи опишем позже.

Подзадача 3

В этой подзадаче $n \leqslant 100$, что означало возможность написать решение, работающее за время $\mathcal{O}(n^3)$ и вообще не пользоваться информацией S. Одним из вариантов такого решения было динамическое программирование: за $\texttt{dp}[i][j][k]$ можно было обозначить булево значение «могли ли элементы a'_i, b'_j и c'_k соответствовать одному и тому же индексу в a, b и c».

Переход следующий: с увеличением индекса в a, b и c каждый из них мог либо поменять значение, либо сохранить предыдущее. Переберем 8 вариантов того, какое подмножество из трех исходных массивов имело различие между текущим и следующим элементом, посмотрим на соответствующее предыдущее состояние $\texttt{dp}$, и, если сумма элементов всех трех массивов не поменялась, выставим значение текущего $\texttt{dp}$ равным $\texttt{true}$.

Поскольку нам гарантируется, что массивы были получены из a, b и c, удовлетворяющих условию, нас не столько интересует сам ответ $\texttt{dp}[m_a][m_b][m_c]$, сколько путь для восстановления ответа, по которому можно восстановить, на каких позициях исходных трех массивов происходило изменение их элементов на следующий элемент из a', b' или c' соответственно.

Если итоговые восстановленные массивы окажутся длины меньше n, можно дополнить их до нужной длины, просто повторяя последние элементы.

Подзадача 4

Эта подзадача напрямую сводится к предыдущей. Заметим, что если один из массивов нам известен, то `dp` достаточно сделать с двумя измерениями, отвечающими за изменения в оставшихся двух массивах — изменился ли элемент a при очередном переходе, можно восстановить из условия на сохранение суммы. А $k = 1000$ достаточно, чтобы вывести в качестве S один из исходных массивов, даже не делая никакого сжатия информации (можно было даже добавить примерно 333 числа с какой-то еще дополнительной информацией).

Подзадача 5

Пятая подзадача уходит с той же идеей еще дальше, но уже имеет два выводящихся из нее решения.

Во-первых, можно выписать в качестве S два массива напрямую, после чего просто восстановить третий, зная фиксированную сумму (она всегда равна $a'_1 + b'_1 + c'_1$).

Либо, можно было заметить, что числа во вводе не превосходят $2^{20} - 1$, а выводить в качестве элементов S можно числа из 30 бит, то есть содержащие в полтора раза больше информации. Если компактно уложить по три исходных числа в два числа от 0 до $2^{30} - 1$, получится просто вывести все три массива в качестве S и даже не писать никакую логику для восстановления, кроме обратного декодирования двух «больших» чисел в три «маленьких».

Подзадачи 6 – 8

В этих подзадачах могли заходить по ограничениям различные оптимизации описанных идей. Так, например, можно было вместо самих исходных массивов сохранять позиции, в которых их элементы меняются на другие. Позиции не превосходят 2^{15}, поэтому требуют меньше информации для записи. Для записи всех позиций изменений в двух из трех массивов понадобится не более 60 000 чисел до 2^{15}, что можно уместить в 30 000 тридцатибитных чисел. Также информацию можно было попробовать сжать какими-то алгоритмами кодирования.

Наконец, можно было просто выписать по биту на каждую позицию в каждом массиве: 1, если в этой позиции очередной элемент массива отличается от предыдущего, и 0 иначе. Всего это 90 000 бит, что означает $|S| = 3\,000$. Чтобы точно уместиться в лимит, стоило аккуратно комбинировать биты в итоговые элементы S.

Подзадача 2

Наконец, рассмотрим вторую подзадачу, из решения которой на самом деле будет следовать ключевая идея задачи. Эта идея — полный перебор возможных вариантов. Но перебор сделаем такой, чтобы перебирать не очень много опций для каждого индекса i.

Для этого посмотрим на исходные a, b и c, и заметим, что если $a_{i+1} \neq a_i$, то чтобы сумма сохранилась, должно выполняться $b_{i+1} \neq b_i$ и/или $c_{i+1} \neq c_i$. То есть с каждым событием вида «траты по какой-то категории изменились» либо меняются траты по двум категориям, либо сразу по трем.

Тогда напишем рекурсивный перебор, который в качестве состояния хранит текущие индексы i_a, i_b и i_c элементов a', b' и c', соответствующих одной и той же позиции в a, b и c, и переходит в состояния $(i_a + 1, i_b + 1, i_c + 1)$, $(i_a + 1, i_b + 1, i_c)$, $(i_a + 1, i_b, i_c + 1)$ и $(i_a, i_b + 1, i_c + 1)$. При каждом переходе необходимо проверить, сохраняется ли сумма, и если нет, завершить эту ветвь перебора.

Из этого решения развивается альтернативное решение десятой подгруппы: закодируем все четыре возможные опции того, какие из массивов в данном индексе изменили значение, и выпишем для каждого i от 1 до n два бита, кодирующие соответствующее изменение. Это 60 000 бит или $|S| = 2\,000$.

Полное решение

Представим второй запуск, когда мы восстанавливаем исходные массивы. Запустим перебор и будем «помогать ему» информацией из S. Если очередной переход может быть только один (при остальных не сходится сумма), то нам не нужна дополнительная информация. А иначе заметим, что только две из четырех опций могут быть подходящими в данный момент.

Действительно, заметим, что если при переходе от i к $i+1$ во всех трех массивах a, b и c изменились значения элементов, то при изменении только двух из них изменилась бы сумма. Аналогично, если изменились только два из них, то при изменении трех изменилась бы сумма, а также одно из изменений — положительное, а другое — отрицательное. По принципу Дирихле среди $a'_{i_a+1} - a'_{i_a}$, $b'_{i_b+1} - b'_{i_b}$ и $c'_{i_c+1} - c'_{i_c}$ есть либо хотя бы два положительных, либо хотя бы два отрицательных числа. А если взять два изменения одного знака, сумма не сохранится.

Таким образом, на самом деле достаточно одного бита на каждое принятое решение вместо двух, как мы написали выше. А тогда можно обойтись $|S| = 1\,000$. А чтобы получить за задачу 100 баллов, надо не выписывать биты для тех решений (ветвлений в переборе), которые принимаются однозначно.

Задача E. Подъем на Высокий Хротгар

Автор задачи: Ильнур Валеев, разработчики: Константин Бац и Владимир Рябчун

Сразу начнем с ключевых фактов: если есть квест, требующий оказаться на высоте d, то в любом случае придется подняться на высоту d. При этом по пути можно выполнить все квесты, которые также ведут по направлению вверх. Поэтому все квесты вверх выполняются за $t_u \cdot \texttt{mx}$, где `mx` — максимальный из всех a_i и b_i.

Несколько сложнее обработать квесты, ведущие вниз. Если бы мы обязаны были спуститься в конце обратно на высоту 0, мы бы выполнили все эти квесты по пути. Но может оказаться так, что в оптимальном способе мы спускаемся из `mx` до некоторого $x > 0$, и оттуда сразу улетаем. Тогда верно следующее:

- нет квестов, ведущих из $a_i \geqslant x$ в $b_i < x$ — далее мы покажем, что в таком случае наш маршрут неоптимален;
- все квесты с $a_i, b_i < x$ уже выполнены по пути наверх.

То есть весь путь выглядит как подъем с 0 до `mx`, в процессе которого мы иногда делаем спуск из очередного a_i в b_i, после чего возвращаемся в a_i и продолжаем идти вверх. В итоге на каждый квест «ниже» x мы потратили доп. время на спуск и подъем. Это доказывает первое утверждение: если какой-то квест из $a_i \geqslant x$ в $b_i < x$ остался невыполненным — нам придется спуститься как минимум до b_i, а иначе мы потратили лишнее время на его выполнение по пути наверх ($a_i \to b_i \to a_i$) вместо того, чтобы просто добавить к пути спуск $x \to b_i$, что требует меньше времени.

Подзадачи 1, 2, 5 и 6

Заметим, что если отрезки, соответствующие двум квестам вниз, пересекаются, то их нет смысла выполнять по отдельности — если $a_1 > a_2 > b_1 > b_2$, то всегда выгодно заменить два независимых прохода $a_2 \to b_2$ и $a_1 \to b_1$ на один проход $a_1 \to b_2$.

Тогда первым действием объединим пересекающиеся отрезки квестов вниз. Это можно сделать простым массивом покрытия (сколько отрезков покрывает такую-то высоту) в первых двух подзадачах и с помощью сортировки событий в пятой подзадаче. После этого выделим получившиеся объединенные отрезки.

Переберем x (куда мы в конечном итоге спускаемся) и для каждого x вычислим ответ. Нам подходят только x, являющиеся нижними точками полученных на предыдущем шаге отрезков (спускаться в «пустоту» между двумя квестами нет смысла), и для каждого из них мы хотим посчитать сумму:

- подъема до `mx`;
- спуска из `mx` до x;
- во время подъема — дополнительных спусков и подъемов по объединенным отрезкам квестов вниз.

Иначе говоря, если объединенные отрезки квестов вниз имеют начала и концы $u_i \to d_i$, то мы хотим найти

$$\texttt{answer}(x) = \left[\sum_{u_i<x} (u_i - d_i) \cdot (t_u + t_d)\right] + t_u \cdot \texttt{mx} + t_d \cdot (\texttt{mx} - x).$$

Для оптимального решения — будем перебирать x по возрастанию и поддерживать сумму по отрезкам ниже x, тогда эту величину можно будет пересчитывать за $\mathcal{O}(1)$.

В первых двух подзадачах достаточно было делать этот пересчет более прямолинейными и менее эффективными способами. Между пятой и шестой подзадачами отличие в невозможности искать объединения отрезков квестов с помощью стандартного дерева отрезков и в невозможности пересчитывать этот ответ в стандартном дереве отрезков. Можно было либо воспользоваться динамическим (неявным) ДО, либо просто не пользоваться им вообще.

Подзадача 4

Из всех рассуждений выше довольно легко вывести ответ для случая, когда квестов, ведущих вниз, нет — просто поднимемся до `mx` и сразу улетим оттуда. На это уйдет $\texttt{mx} \cdot t_u$ времени.

Подзадачи 3, 7 и 8

Случай, когда $q > 0$, требует динамического пересчета ответа при добавлении квестов. То есть по сути надо q раз добавить квест и обновить ответ.

Это можно делать с помощью дерева отрезков (неявного в последней подгруппе) или дерева поиска на объединенных отрезках квестов. Вспомним, что нас интересует вычисление указанной выше функции в точках, являющихся нижними концами этих отрезков. В самой функции могут меняться следующие величины:

- `mx` — его можно поддерживать за $\mathcal{O}(1)$ и пересчитывать связанную с ним составляющую функции сразу для всех x;
- сумма длин отрезков ниже x.

При этом третья компонента функции линейно зависит от x, что делает неудобным ее пересчет при массовых операциях изменения другой компоненты (отрезки под x). Перепишем функцию, добавив и вычтя сумму длин всех отрезков над x включительно, умноженную на t_d, и получим:

$$\texttt{answer}(x) = f(\texttt{mx}) + t_u \cdot \left[\sum_{u_i<x} (u_i - d_i)\right] + t_d \cdot \left[\sum_i (u_i - d_i)\right] + t_d \cdot (\text{empty above } x),$$

где «empty above x» — сумма длин промежутков между отрезками квестов над x, а $f(\texttt{mx})$ зависит только от `mx`.

Теперь изменяющиеся части разбились на «сумму длин отрезков под x», «сумму длин промежутков между отрезками над x» и «сумму длин всех отрезков». Построим ДО, хранящее в позиции x значение $\texttt{answer}(x)$, тогда добавление нового квеста $(p_i \to q_i)$ вниз — это просто пересчет суммы длин всех отрезков и операции «`+=`» на префиксе и суффиксе данного ДО (увеличилась сумма длин отрезков под x для всех $x \geqslant p_i$ и уменьшилась сумма длин пустых интервалов для всех $x \leqslant q_i$).

Приращение этих величин также можно искать с помощью ДО с операцией «количество нулей на отрезке».

Опять же, различия между группами в эффективности реализации этой логики (линейные проходы вместо ДО, стандартное ДО, неявное ДО). Полное решение работает за $\mathcal{O}((m+q)\log n)$. Было также альтернативное решение с той же асимптотикой и декартовым деревом самих отрезков с поддержкой тех же операций.

Задача F. Простая загадка

Автор задачи и разработчик: Павел Скобелин

Во всех подзадачах для нахождения простых чисел и ответа на запрос количества чисел на отрезке можно было построить Решето Эратосфена за $\mathcal{O}(n)$ и префиксные суммы на нем. В подзадачах с небольшими l и r можно было найти множество простых чисел от 0 до r более простыми методами.

Подзадачи 1 и 2

В этих подзадачах были небольшие ограничения на l, r. Получается, что изначально вариантов ввода не более 10^4, поэтому можно было сделать аккуратный предподсчет.

А именно, предподсчитаем для каждого возможного отрезка во вводе число простых на нем и число простых при нескольких случайных сдвигах. Поскольку простые числа распределены достаточно случайно и примерно с частотой «$\frac{n}{\log n}$ простых от 1 до n», то достаточно было очень небольшого числа запросов, чтобы однозначно идентифицировать исходные, а значит и текущие значения l и r.

Подзадачи 3 и 4

Здесь вариантов ввода уже больше — порядка 10^6, из-за чего предподсчет нескольких случайных запросов для каждого отрезка мог занять слишком долго времени. В целом, как и в первой и второй подгруппе, в третьей можно было просто сделать $m/2$ запросов $(0, +1)$ и $m/2$ запросов $(+1, 0)$ после этого и определить оба значения по профилю того, на каких позициях относительно l и r встретились простые числа.

Либо можно было сразу решать четвертую подгруппу и сгруппировать все возможные отрезки ввода по числу простых на них, после чего совместить разные идеи из числа:

- Приблизительно оценить длину отрезка и сдвинуть l и r вместе тернарным поиском;
- Делать случайные запросы, ответы на которые работали бы как дерево решений;
- Сделать достаточное число запросов $(0, +1)$, чтобы заметно сузить множество возможных исходов, зная расстояния между соседними ближайшими простыми.

Подгруппы 5 и 6

Здесь лимит на число запросов был равен 170. Давайте рассмотрим числа от 1 до 10^6, и выпишем 1, если число простое, и 0 иначе. Предпосчитав локально, можно заметить, что все подотрезки этой маски длины $k = 160$ и более — различные. Таким образом, зная информацию про простоту каждого из 160 подряд идущих чисел, можно узнать, где находится указатель.

Тогда давайте k раз сделаем запрос $(0, +1)$. Делая такой запрос, мы увеличиваем r на 1, и, сравнив с предыдущим ответом, понимаем, является ли $r + 1$ простым. Значит, за k запросов мы можем понять, где находится правый указатель. После нахождения правого указателя мы знаем его и количество простых чисел между l и r (пусть их x). Таким образом, мы можем найти возможные границы, где находится l (отсчитав x простых чисел влево от r, с помощью бинарного поиска или линейного прохода).

Расстояние между простыми числами в диапазоне до 10^6 не превосходит 115, значит можно найти l с помощью бинарного поиска за 7 запросов. Таким образом, можно найти обе границы не более чем за 170 запросов.

Если реализовывать это менее эффективно, каждый раз проверяя все возможные маски и пользуясь линейными проходами вместо бинарного поиска, решение будет иметь асимптотику $\mathcal{O}(10^6 \cdot k)$ и пройдет только пятую подзадачу. Если же реализовывать поиск подмаски в маске с помощью хэшей или словаря, решение пройдет и пятую, и шестую подзадачи.

Подзадачи 7 и 8

Здесь требовалось немного улучшить предыдущее решение: заметим, что в маске простых чисел каждое второе число будет нулем (кроме 2 и 3, стоящих подряд), значит можно двигать r с шагом 2 и получить ту же маску за 80 запросов. С теми же рассуждениями это дает нам решение за 90 запросов.

Подзадачи 9 – 12

Эти подзадачи можно было пройти, придумав некоторые или все идеи полного решения.

Заметим, что в предыдущем решении с одного запроса мы получаем всего лишь один бит информации — является число простым или нет. Попробуем пользоваться количеством чисел, которые получаем из запроса, более активно.

Введем операцию сдвига в ближайшее простое. Чтобы подвинуть r в ближайшее простое число, сделаем уже упомянутый выше бинпоиск за $1 + 7 + 1 = 9$ запросов: один запрос на инициализацию, семь запросов на бинарный поиск (для нахождения положения r, ведь расстояние до ближайшего простого не больше 115), и последний запрос, чтобы поставить r в конце в найденное простое число.

Далее возьмем последовательность $[\Delta_1, \Delta_2, \dots \Delta_q]$ и сделаем q запросов $(0, +\Delta_1), \dots (0, +\Delta_q)$. Таким образом мы узнаем количество простых чисел на отрезках $[r{+}1, r{+}\Delta_1], [r{+}\Delta_1{+}1, r{+}\Delta_1{+}\Delta_2], \dots$. Найдем такую последовательность Δ_i, чтобы по ответам на эти q запросов можно было однозначно восстановить r. Такую последовательность можно было найти локально, генерируя случайные последовательности и проверяя нужное свойство. Например, последовательность длины 10 сгенерировалась у жюри не дольше, чем за минуту — это

$$320, 184, 175, 68, 235, 334, 149, 209, 103, 266.$$

.

Таким образом за $1 + 7 + 1 + 10$ запросов можно найти r. После этого, аналогично предыдущим подзадачам, за 7 запросов можно найти левую границу. Итоговое решение работает за $1 + 7 + 1 + 10 + 7 = 26$ запросов (хотя мы верим, что существует более короткая последовательность Δ_i, справляющаяся с той же задачей).