Задача A. Fence Painting

Автор задачи и разработчик: Маргарита Саблина

Подзадача 1

Для решения первой подзадачи достаточно было перебрать место встречи, после чего проходом по всем столбикам от 1 до $a + b$ найти, сколько денег в итоге получат Том и Гек. Для каждого столбика i можно быстро понимать, сколько монет за его покраску выдается: x, если $i \leqslant a$, и y иначе. Из всех возможных мест встречи оставалось выбрать такое, при котором абсолютная разность между величинами выплаченных сумм минимальна.

Подзадачи 2 и 3

Во второй и третьей подзадаче проходили частные решения. В случае $x = y$ все столбики стоят одинаково. В этом случае требовалось, чтобы Том и Гек покрасили примерно одинаковое количество столбиков (одинаковое или отличающееся на 1). В качестве ответа достаточно было вывести $\left\lfloor \frac{a+b}{2} \right\rfloor$.

В случае $a = b$ можно было однозначно определить, кому из ребят придется покрасить больше столбиков: например, если $x < y$, то Тому придется покрасить все деревянные столбики, и возможно еще сколько-то металлических. В таком случае количество металлических столбиков, которые ему пришлось бы покрасить, можно было вычислить, исходя из того, насколько ax меньше by: за каждый металлический столбик, отданный Тому вместо Гека, первый получает на y денег больше, а второй — на y меньше. Поэтому количество металлических столбиков, которые следовало отдать Тому, должно быть равно либо $\left\lfloor \frac{by-ax}{2y} \right\rfloor$, либо $\left\lceil \frac{by-ax}{2y} \right\rceil$.

Подзадача 4

Как и в первой подзадаче, достаточно было перебрать место встречи, однако в здесь ограничения уже не позволяли вычислять полученное каждым из ребят число монет за линейное время. Можно было вместо этого воспользоваться формулой: если $i \leqslant a$, то Том получил бы за первые i столбиков $g_t = i \cdot x$, иначе — $a \cdot x + (i - a) \cdot y$. Количество монет, полученных Геком, можно вычислить как $g_h = ax + by - g_t$.

Подзадача 5

Для полного решения оставалось заметить, что функция $g_t(i) - g_h(i)$ монотонно возрастает, либо же что $|g_t(i) - g_h(i)|$ сначала монотонно убывает, затем возрастает. Таким образом, можно воспользоваться двоичным поиском, чтобы найти значения $g_t(i) - g_h(i)$, наиболее близкие к нулю, либо же тернарным поиском, чтобы найти абсолютный минимум $|g_t(i) - g_h(i)|$.

Альтернативно можно было воспользоваться формулой. Мы хотим, чтобы каждый из ребят получил значение, наиболее близкое к половине, то есть $\frac{ax+by}{2}$. Если $ax \geqslant by$, то Том должен был покрасить не более a столбиков, а именно — либо $\left\lfloor \frac{ax+by}{2x} \right\rfloor$, либо $\left\lceil \frac{ax+by}{2x} \right\rceil$. Симметричный случай аналогичен, но в нем удобнее считать количество столбиков, покрашенных Геком. Из двух возможных вариантов в любом случае надо явным образом выбрать наиболее оптимальный (либо же использовать `round` вместо округления в конкретную сторону).

Задача B. Work, Sleep, Repeat

Автор задачи: Мария Жогова, разработчик: Даниил Орешников

Важным фактом для решения задачи является то, что из-за периодичности цикла с периодом $x + y$ дней каждый d_i можно воспринимать как $d_i \bmod (x + y)$. Дальше останется только проверить существование такого дня D, при котором все d_i попадают в первые x дней цикла после него, то есть от D до $D + x - 1$.

Подзадача 1

При $x = 1$, как можно заметить, пользуясь описанным выше наблюдением, необходимо, чтобы все все дни d_i приходились на один и тот же остаток по модулю $x + y$, то есть $y + 1$. Таким образом, достаточно по каждому d_i посчитать $d_i bmod(y + 1)$ и проверить, что все полученные числа совпали. Если да, то как раз это полученное число и будет искомым ответом, иначе — ответа нет.

Подзадача 2

В этой подзадаче ограничения достаточно небольшие, чтобы можно было перебрать все возможные D от 1 до $x+y$, и для каждого из них проверить, подходит ли это значение. Пользуясь ключевым наблюдением, необходимо, чтобы для каждого d_i со дня D прошло сколько-то (возможно, 0, или в новом условии, -1) целых циклов по $x + y$ и еще не больше x дней после. То есть для каждого d_i должно выполняться $d_i = D + k(x + y) + w$, где $w < x$.

Для этого достаточно проверить, что для всех d_i верно $(d_i - D) \bmod (x + y) < x$. Таким образом, во второй подзадаче при переборе всех возможных D и проверке каждого, что он подходит, решение работает за $\mathcal{O}(n(x + y))$, чего было достаточно, чтобы пройти все тесты.

Подзадача 3

При ограничении $y = 1$ мы встречаемся с ситуацией, абсолютно противоположной первой подзадаче — в первой подзадаче каждый d_i накладывал ограничение, чтобы D имел такой же остаток по модулю $x + y$, а здесь наоборот, каждый d_i «запрещает» ровно один возможный остаток. Например, при $x = 5$ значение $d_i = 12$ запрещает $D = 1$. Иначе рабочие дни были бы 1 – 5 и 7 – 11, а день 12 приходился бы на отпуск.

Таким образом, достаточно завести множество всех «запрещенных» значений D, и для каждого d_i добавить в это множество $(d_i - x) \bmod (x + y)$. Если множество всех запрещенных значений будет размера меньше $x + y$, то гарантированно найдется число D, подходящее под все условия. Найти его можно за $\mathcal{O}(n)$, так как всего будет не более n запретов, то есть какое-то из чисел от 1 до $n + 1$ в множестве запрещенных будет отсутствовать.

Подзадачи 4 и 5

Для полного решения достаточно было обобщить предыдущие идеи. А именно, что каждый d_i вносит ограничение вида «D должен находиться в некотором отрезке остатков по модулю $x + y$». А именно, соответствующий цикл не может начаться раньше дня $d_i - x + 1$ и не может начаться позже d_i, то есть $D \in [(d_i - x + 1) \bmod (x + y), d_i \bmod (x + y)]$.

Поэтому полное решение — это выписать все такие ограничения и найти их пересечение. Единственная проблема — отрезок остатков может иметь вид $[7, 3]$, то есть «либо остаток хотя бы 7, либо остаток не меньше 3». В подзадаче 4 гарантируется, что $x < y$, а тогда можно доказать, что пересечение любых двух отрезков длины x будет тоже давать один отрезок длины не больше исходных, и пересечение отрезков можно выполнить за $\mathcal{O}(n)$.

Для подзадачи 5 можно было воспользоваться любым удобным методом пересечения циклических отрезков — например, отсортировать все $d_i \bmod (x + y)$ и двумя указателями проверить, что существует такой D, что все d_i лежат от него до $D + x - 1$, либо же представить каждый отрезок ограничения двумя событиями вида «ограничение началось» и «ограничение закончилось», отсортировать события, и найти остаток, при котором счетчик активных ограничений равен n. Время работы полного решения, таким образом, равно $\mathcal{O}(n \log n)$.

Задача C. Palindromization

Автор задачи и разработчик: Владислав Власов

Для начала воспользуемся некоторыми преобразованиями. Они нужны для решения не всех подзадач, однако, сделав их, затем будет проще думать о том, что в задаче требуется, и почему время работы, например, полного перебора, будет достаточным, чтобы уложиться в ограничения.

Итак, рассмотрим наш массив a.

1. Заметим, что нет смысла делать прибавления на отрезках, пересекающих середину массива. Действительно, если m — середина массива, то добавление x на отрезке $[m-p, m+q]$, где $p < q$, с точки зрения цели задачи равносильно добавлению x на отрезке $[m+p+1, m+q]$. Действительно, разница между этими двумя действиями только в прибавлении x на $[m-p, m+p]$, а это действие не изменяет свойство a быть палиндромом.

2. Теперь сведем задачу к другой: рассмотрим массив b вида

$$b = [a_1 - a_n, a_2 - a_{n-1}, \ldots, a_{\lfloor \frac{n}{2} \rfloor} - a_{\lceil \frac{n+1}{2} \rceil + 1}],$$

   то есть массив разностей противоположных элементов массива a. Если a — палиндром, то все элементы b — нули. А прибавление x на отрезке массива a равносильно прибавлению или вычитанию x на отрезке массива b.

3. Теперь посчитаем c — разностный массив массива b, к которому дописали 0 в начало и в конец, то есть

$$c = [b_1, b_2 - b_1, b_3 - b_2, \ldots, b_{\lfloor \frac{n}{2} \rfloor} - b_{\lfloor \frac{n}{2} \rfloor - 1}, 0 - b_{\lfloor \frac{n}{2} \rfloor}].$$

   В таком массиве почти ничего не меняется в плане нашей цели — когда все элементы b равны нулю, все элементы c равны нулю, и наоборот. Однако теперь мы свели прибавление x на отрезке b к прибавлению x к одному элементу c и вычитанию x из другого.

Подзадача 1

В первой подзадаче можно было написать аккуратный полный перебор возможных действий. Будем анализировать массив b вместо массива a. В нем числа лежат от -9 до 9, и за каждое действие достаточно выбрать два числа одного знака и прибавить или вычесть некоторое значение на отрезке между ними.

Заметим, что порядок действий не важен, а значит можно идти по массиву b от начала к концу и для каждого очередного b_i выбирать ему конец отрезка, элемент на котором того же знака, и делать прибавление на отрезке. Массив b имеет размер не более 5, поэтому различных отрезков для прибавления не больше $\frac{5 \cdot 4}{2} = 10$ (на самом деле еще меньше, если концы отрезков выбирать одного знака).

При чем видно, что эвристически делать прибавления меньших x выгодно только тогда, когда в b исходного много чисел $< k$, а в таком случае ответ будет меньше. Таким образом, у нас есть меньше 10 возможных отрезков для прибавления, на каждом есть несколько способов выбрать прибавляемое значение, и в конечном итоге полный перебор действий на тестах первой группы будет укладываться в ограничения. Единственное необходимое для этого наблюдение — что можно не рассматривать в a отрезки, пересекающие середину массива.

Подзадача 2

В этой подзадаче к каждому элементу нужно прибавить не больше чем 1. Снова будем рассматривать массив b, и заметим, что именно b_i операций нужно применить к i-му элементу. Можно доказать, что в таком случае ответом будет являться количество непрерывных отрезков из единиц и минус единиц в массиве b. Действительно, за каждое действие не получится уменьшить количество таких отрезков хотя бы на 2, значит ответ не может быть меньше.

Подзадача 3

Заметим следующий факт: если $a_i \leqslant a_{i+1}$ для всех i, то в массиве b все числа будут отрицательными, при чем неубывающими. В таком случае количество операций, применяемых к b_i, будет не меньше, чем количество операций, применяемых к b_{i+1}. Соответственно, применим следующий алгоритм:

1. Будем прибавлять k ко всем числам массива b, пока очередное прибавление не сделает его последний элемент положительным. Тогда вместо этого прибавим минус последний элемент b ко всем его элементам.

2. Мы добились того, что $b_{\text{last}} = 0$, за минимальное число действий. Теперь повторим те же действия, но уже для отрезка с первого элемента b до предпоследнего.

3. Когда последние два элемента b равны нулю, продолжим сужать префикс, на котором мы совершаем прибавления, постепенно придя к тому, что весь массив b состоит из нулей.

Факт, что мы использовали минимальное число действий, можно доказать конструктивно. Для этого достаточно показать, что любой способ, включающий в себя прибавление на двух непересекающихся отрезках, можно преобразовать в не худший способ, в котором этих отрезков нет. Это решается небольшим разбором случаев, который мы не будем здесь приводить, потому что это не является ключевой идеей, необходимой для решения.

Подзадача 4

Начиная с этой подзадачи, будем пользоваться массивом c для решения. Когда мы посчитали массив c, нашей задачей становится за минимальное число действий вида «прибавить x к одному элементу и вычесть из другого» получить полностью состоящий из нулей массив. В подзадаче с $k = 1$ это достигается достаточно прямолинейным образом: нужно за каждую операцию прибавлять 1 к отрицательному элементу и вычитать 1 из положительного.

Поскольку сумма элементов массива c равна нулю, то можно необходимое количество действий просто посчитать как сумму всех его положительных элементов. Время работы решения — $\mathcal{O}(n)$.

Подзадача 5

Аналогично с прошлой подзадачей, построим массив c, но обработаем немного по-другому. Теперь мы можем прибавлять и вычитать 1 или 2. На число c_i необходимо $\left\lceil \frac{|c_i|}{2} \right\rceil$ операций, меньше невозможно. Посчитаем две суммы таких величин: для положительных значений и для отрицательных. Ответом будет являться большая из них.

Для этого сначала заметим, что нет смысла вычитать из отрицательных чисел и прибавлять к положительным — тогда всегда можно либо уменьшить ответ, либо избавиться от такой операции, не увеличив его. А затем заметим, что меньшего количества операций добиться нельзя, а за такое количество операций получить все нули можно. В той части, в которой получилась максимальная сумма $\left\lceil \frac{|c_i|}{2} \right\rceil$, каждое c_i обработаем как $\left\lfloor \frac{|c_i|}{2} \right\rfloor$ операций прибавления или вычитания двойки, и еще возможно одну операцию прибавления или вычитания единицы. В другой части сумма получилась меньше, значит в ней было меньше нечетных чисел. Тогда каждому нечетному все еще хватит по единице, а «лишние» единицы сгруппируем по две, чтобы добавить к тем четным, которым не хватило двоек.

Подзадача 6

Для последней подзадачи необходимо было сделать еще несколько конструктивных наблюдений. Для начала переформулируем задачу как «дано множество положительных c_i и множество отрицательных c_i, требуется найти такое множество слагаемых от 1 до 3, что на них можно разложить все числа как первого, так и второго множества». Действительно, каждая операция — это прибавление к отрицательному и вычитание из положительного. По сути мы раскладываем на слагаемые множества положительных и отрицательных чисел среди c_i.

Теперь заметим, что:

- Любое разложение на слагаемые от 1 до 3 числа $c_i \geqslant 8$ всегда содержит в себе несколько слагаемых, дающих в сумме ровно 6 (доказывается аккуратным разбором случаев). Из этого следует, что вместо разложения $c_i \geqslant 8$ можно независимо раскладывать $c_i - 6$ и 6.

- В разложении чисел $c_i = 5$ или $c_i = 7$ всегда найдется несколько слагаемых с суммой 3. Значит можно раскладывать независимо $c_i - 3$ и 3.

Тогда разделим c на группу отрицательных и группу положительных чисел и преобразуем описанным образом. В каждой группе останутся только числа из множества $\{1, 2, 3, 4, 6\}$. Теперь приведем решение, позволяющее найти ответ за время $\mathcal{O}(n^2)$. Для этого заметим, что количество чисел в каждом таком наборе, не равных 6, линейно от n (из каждого исходного c_i получилось не более двух чисел от 1 до 4). Более того, не может быть так, чтобы в обоих наборах было разложение $6 = 2+2+2$, так как тогда его можно заменить на $6 = 3 + 3$ с меньшим числом слагаемых. Аналогичное верно для единиц.

Тогда хотя бы в одном наборе количество единиц не превосходит $\texttt{cnt}[1] + 2\texttt{cnt}[2] + 3\texttt{cnt}[3] + 4\texttt{cnt}[4] + 6$, а количество двоек — $\texttt{cnt}[2] + \texttt{cnt}[3] + 2\texttt{cnt}[4] + 3$. На самом деле будет видно, что необходимое количество единиц и двоек еще меньше, но пока нам хватит факта, что они линейны от n.

Переберем тогда количество используемых единиц (m_1) и двоек (m_2), найдем количество троек как $\frac{\texttt{total}-m_1-2m_2}{3}$, и проверим, можно ли оба множества разбить на такие слагаемые. Чтобы проверить, можно ли разбить множество на такие слагаемые, достаточно проверить следующие условия:

1. Если $m_1 < \texttt{cnt}[1]$, то нельзя, так как единицы получить по-другому нельзя. Иначе — уменьшим m_1 на $\texttt{cnt}[1]$ и забудем про них пока.

2. Если $m_2 \leqslant \texttt{cnt}[2]$, то на оставшиеся двойки и все четверки должно хватить единиц, все остальное разложится на слагаемые, равные 3.

3. Если $m_2 \leqslant \texttt{cnt}[2] + 2\texttt{cnt}[4]$, то на оставшиеся четверки должно хватить единиц. Более того, если $m_2 - \texttt{cnt}[2]$ нечетно, то оставшаяся двойка явно потребует лишней единицы. Действительно, либо в разложении какого-то числа придется поставить $2+1$, либо из текущего распределения двоек придется одно разложение убрать, и в нем использовать единицы вместо двоек. Небольшой разбор случаев того, как слагаемые 2 можно распределить по «целям» показывает, что одной лишней единицы хватает, чтобы оставшиеся числа распадались на слагаемые 3, а без нее разложить не получится.

4. Если $m_2 \leqslant \texttt{cnt}[2] + 2\texttt{cnt}[4] + 3\texttt{cnt}[6]$, то разобьем все как $2 = 2$, $4 = 2 + 2$ и $6 = 2 + 2 + 2$, и останется либо одна, либо две лишние двойки. Еще один небольшой разбор случаев показывает, что на каждую из оставшихся лишних двоек понадобится лишняя единица вне зависимости от того, распределять двойки показанным образом или нет.

5. Если $m_2 > \texttt{cnt}[2] + 2\texttt{cnt}[4] + 3\texttt{cnt}[6]$, то аналогично, на каждую лишнюю двойку понадобится по единице, чтобы объединить их в $3 = 1 + 2$.

6. Если $m_2 > \texttt{cnt}[2] + 2\texttt{cnt}[4] + 3\texttt{cnt}[6] + \texttt{cnt}[3]$, то не хватит «мест», в которые двойки можно распределить.

Таким образом, для каждой пары (m_1, m_2) можно определить, дают ли они корректное разбиение, и выбрать ту, которая минимизирует количество слагаемых в целом. Есть также альтернативное решение, сводящее эту задачу к рюкзаку, однако мы здесь его приводить не будем.

Задача D. Hackathon

Автор задачи: Егор Юлин, автор решения и разработчик: Даниил Орешников

Подзадача 1

Первая подзадача была расчитана на аккуратную реализацию описанного в условии процесса. Достаточно было посекундно перемещать участников на поле, добавлять новых в момент их прихода, и пересчитывать их цели. Для этого будем поддерживать текущую позицию каждого участника и его текущую «цель» — позицию минимума на его префиксе в массиве a.

После каждого «тика» времени (раз в секунду) достаточно пересчитать позиции участников и сгруппировать их по позициям в множества. Можно делать и более эффективно, заметив, что два участника, оказавшись на одной позиции, будут и дальше идти вместе, пока кто-то из них не заберет желаемый кусочек пицца, и поддерживать эти самые множества, а не пересчитывать каждый раз.

Для каждого множества участников, пришедших к своей цели, достаточно выбирать участника с минимальным номером, отдавать ему соответствующий кусочек, и после пересчитывать цели. Максимальное время в этой группе не будет превосходить $t_i + v \cdot s_i \leqslant 20\,000$, и в каждый момент надо обновить позиции всех участников ($\mathcal{O}(m)$) времени плюс пересчитать цели (линейным проходом по массиву за $\mathcal{O}(n+m)$ или даже за $\mathcal{O}(nm)$). Стоит отметить, что цели пересчитываются только при изменении массива a, что произойдет не более m раз. Таким образом, даже самая неэффективная реализация этого решения работает за $\mathcal{O}(T \cdot m + m^2 n)$, чего с запасом хватает для ограничений этой подзадачи.

Подзадача 2

Когда все участники стартуют в одно и то же время с одной и той же позиции, их стоит воспринимать как одно единое целое — к каждой цели они будут приходить вместе. И каждый раз участник с минимальным номером будет получать кусочек пиццы и уходить на место.

Будем поддерживать кучу или дерево поиска из пар $(a_i, -i)$, чтобы находить минимальный a_i, а при равенстве значений — самый правый из них. В куче будем хранить все элементы на позициях до стартовой позиции всех участников включительно, и до перемещения проверять, является ли текущая позиция позицией с кусочком пиццы максимальной толщины. При выдаче какого-то a_i очередному участнику достаточно положить в кучу обратно элемент $(a_i + 1, -i)$.

Операций с кучей будет не больше $n + m$ (каждый элемент плюс количество раз, которое мы выдали кусочек пиццы участнику), поэтому время работы такого решения равно $\mathcal{O}((n + m) \log n)$.

Подзадача 3

Это еще одна подзадача, позволяющая получить баллы, написав частичное решение, не оптимизируемое до полного. Условие $|a_i - a_j| \geqslant n$ означает, что никто из участников ни разу не поменяет свою цель. Действительно, смена цели означает, что участник шел к некоторому значению $a_j = x$, его забрали, и теперь его новый минимум — хотя бы $a_i = x + 1$. Но если какой-то a_i увеличился с одного из изначальных значений до другого a_j, то он увеличился на 1 ровно $a_j - a_i$ раз, что в данной подзадаче не меньше m. Поскольку увеличения происходят только после взятия очередным участником какого-то кусочка пиццы, не может быть такого, что кто-то из участников еще не получил свой кусочек, а значение уже увеличилось на m.

В таком случае достаточно для каждого участника в момент его появления около ряда найти минимальный a_i на префиксе — эта позиция и будет для него ответом. Это можно делать несколькими способами, в частности — с помощью дерева отрезков с операцией изменения элемента и поиска минимума на префиксе. Участников в таком случае стоит изначально отсортировать по времени их подхода к ряду, для каждого определить время, в которое он подойдет к конечному кусочку пиццы, и затем в порядке увеличения этого времени обработать изменения a_i, чтобы для каждого участника определить итоговый ответ. Время работы решения — $\mathcal{O}((n + m) \log n)$.

Подзадачи 4 и 5

Еще одним возможным подходом к решению задачи была сортировка событий. Заметим, что наивная симуляция не является эффективной по нескольким причинам:

1. мы тратим $\mathcal{O}(m)$ времени на обработку каждой секунды, даже тех, в которые ничего не происходит;

2. мы долго ищем новые цели для участников, даже не разделяя тех, для кого их надо пересчитать, и тех, для кого не надо.

Эти действия можно было реализовать эффективнее. Заведем кучу, хранящую события вида «участник пришел к своей (возможно, уже неактуальной) цели» и «участник подошел к ряду с пиццами». Упорядочивать события в куче будем по времени, а при одинаковой отметке времени сначала обработаем тех, кто подошел, а затем — тех, кто дошел до цели.

В момент подхода к ряду посчитаем текущую цель участника — это будет минимум на префиксе в текущем состоянии массива a. Вычисляем момент, в который он к этой цели подойдет, и добавляем соответствующее событие в кучу. В момент обработки события «участник подошел к своей цели» пересчитываем события для всех, кто шел к той же позиции. Это можно делать либо сразу, либо отложенно — пересчитать цель, когда участник подойдет к своей старой цели и увидит, что известное ему до этого значение неактуально. Такие два подхода дают одинаковый результат.

При эффективной реализации поиска минимума на префиксе (например, через дерево отрезков) такое решение проходит обе группы. При менее эффективной, но быстрее, чем за $\mathcal{O}(nm^2)$ в сумме — проходит четвертую. Альтернативно можно было решить четвертую подгруппу вариацией полного решения с худшей асимптотикой.

Подзадача 6

Идея обрабатывать «цели», то есть кусочки пиццы, с помощью кучи в порядке возрастания, которая появляется во подзадаче, является ключевой идеей для полного решения. Действительно, найти по позиции и времени начала движения участника его конечную точку будет сложно. Проще для каждой позиции найти участника, который на ней остановится.

Будем обрабатывать пары $(a_i, -i)$ в порядке возрастания. Для очередной такой пары найдем участника, который ее займет. Для этого участника j должно выполняться три условия:

1. $s_j \geqslant i$, то есть участник подошел к ряду не левее этой позиции;
2. к моменту подхода участника к позиции i, то есть $t_j + v \cdot (s_j - i)$, a_i является минимумом на соответствующем префиксе;
3. из всех таких участников выбирается подошедший к этой позиции раньше всех, или, если таких несколько — с минимальным номером из них.

Преобразуем условие на время: то, что к моменту подхода к i он является минимумом на префиксе, равносильно тому, что время подхода не меньше, чем любое время взятия кусочка пиццы, находящегося левее (так как мы обрабатываем позиции по возрастанию значений a_i). Назовем момент времени, когда a_i становится минимумом на префиксе, $\texttt{stable}(a_i)$. Тогда мы хотим $t_j + v \cdot (s_j - i) \geqslant \texttt{stable}(a_i)$ или же $t_j + v \cdot s_j \geqslant \texttt{stable}(a_i) + v \cdot i$. Слева стоит функция от j, справа — от i и текущего состояния a_i.

Переформулируем теперь задачу как «найти участника с $s_j \geqslant i$ и минимальным $t_j + v \cdot s_j$, не меньшим $\texttt{stable}(a_i) + v \cdot i$». Это можно делать несколькими способами, один из которых — отсортировать участников по s_j, для каждого вычислить искомую величину $\texttt{target}_j = t_j + v \cdot s_j$, и разбить на блоки размера $\sqrt{m \log m}$, после чего отсортировать участников в каждом блоке по $\texttt{target}$. Тогда для поиска участника, у которого $\texttt{target}$ минимален среди всех, не меньших $\texttt{stable}_i + v \cdot i$, достаточно сделать двоичный поиск в нескольких блоках, и еще часть блока возможно обработать отдельно, посмотрев на каждого участника. Такой запрос работает за $\mathcal{O}(\sqrt{m \log m})$.

Осталось только научиться помечать участника, получившего кусочек пиццы, как ушедшего из ряда. Для этого можно либо перестроить заново его блок, либо воспользоваться техникой сжатия путей — запомнить для каждого человека следующего по величине $\texttt{target}$ в его блоке, который еще не удален, и при нахождении участника переходить по этим ссылкам, параллельно сжимая их в более короткие пути.

Отдельно отметим, что если не нашлось участников, забирающих себе кусочек на позиции i, то эта позиция уже ни для кого никогда не станет целью, поэтому ее можно не добавлять обратно в кучу. Полное решение, таким образом, работает за $\mathcal{O}((n + m)\sqrt{m \log m})$. Есть и более эффективное асимптотически решение с деревом отрезков на множествах для поиска участника с теми же критериями, но у него константа больше, и физическое время работы дольше.

Задача E. Tree Trisection

Автор задачи и разработчик: Даниил Орешников

Для начала введем несколько обозначений. Обозначим минимальное связное множество, содержащее все листья с l по r и имеющее корень в v как $\texttt{cover}(v, l, r)$. Также обозначим минимальный лист в поддереве вершины v за $\text{left}(v)$, а максимальный — за $\text{right}(v)$.

Теперь немного проще переформулируем условие и заметим несколько важных фактов. Дано полное двоичное дерево, и поступают запросы, указывающие отрезок его листьев $[l, r]$. Для начала выбирается множество $V = \texttt{cover}(\text{lca}(l, r), l, r)$. После требуется выбрать в этом множестве две вершины x и y так, чтобы если «отрезать» само V и в нем — поддеревья x и y, то максимальный из весов трех полученных поддеревьев будет минимален.

Соответственно, два полезных факта:

1. Вершину $\texttt{root}(\texttt{V})$ можно найти как $\text{lca}(l, r)$, при чем в нумерации с 1 левый ребенок вершины i имеет номер $2i$, а правый — $2i + 1$, поэтому $\text{lca}(l, r)$ имеет номер, двоичная запись которого является наибольшим общим префиксом двоичных записей l и r.

2. Для того, чтобы минимизировать максимальный из весов трех деревьев, достаточно максимально приблизить их веса к трети веса V. Действительно, сумма их весов фиксирована и равна весу V, а значит максимальный из них будет не меньше $\frac{\texttt{weight}(V)}{3}$.

Подзадача 1

Как и обычно, первая подзадача расчитана на достаточно прямолинейную реализацию того, что просят в условии задачи, и в ней нам даже не понадобятся факты, рассмотренные выше. Найти $\texttt{root}(V)$ можно за время $\mathcal{O}(\log n)$, просто рассмотрев всех предков l и r. А затем достаточно в явном виде выписать все вершины, которые входят в V: это вершины, отрезок листьев которых пересекается с $[l, r]$, и перебрать все варианты выбор x и y.

Даже если для каждого запроса перебирать все возможные пары x и y, и для обеих вершин считать вес их поддерева между листьями l и r за время $\mathcal{O}(n)$, мы получим решение за время $\mathcal{O}(mn^3)$, что проходит при ограничениях этой подзадачи. Требовалось также любым способом аккуратно для каждой перебираемой пары (x, y) проверять, что они не являются предком и потомком — это можно сделать за $\mathcal{O}(\log n)$ перебором предков каждой из этих двух вершин.

Подзадача 2

Чуть более аккуратное вычисление весов поддеревьев x и y за время $\mathcal{O}(\log n)$, например, с предподсчетом веса каждого полного поддерева и спуском со сложением нужных весов из x и y вниз позволяло сдать подзадачу 2, в которой асимптотики $\mathcal{O}(mn^2 \log n)$ было достаточно. Более подробно — чтобы вычислить вес некоторого поддерева в V, достаточно было спускаться из корневой вершины этого поддерева вниз, и для каждой вершины

- если отрезок ее листьев не пересекается с $[l, r]$, возвращать ноль;
- если отрезок ее листьев содержится в $[l, r]$, возвращать целиком вес ее поддерева;
- иначе — спускаться в ее детей и возвращать полученные из них суммы весов.

Тогда на каждом слое дерева будет посещено не более четырех вершин, и не более, чем из двух из них, будет осуществлен спуск еще на слой ниже, то есть время работы такого подсчета веса пропорционально высоте дерева.

Можно было также для каждого листа a и его предка p посчитать суммарный вес $\texttt{cover}(p, a, \text{right}(2p))$ или $\texttt{cover}(p, \text{left}(2p + 1), a)$, то есть поддерева, содержащего либо все листья между a и «серединой» поддерева p. Тогда для $v = \text{lca}(l, r)$, вес $\texttt{cover}(v, l, r)$ раскладывается в вес $\texttt{cover}(v, l, \text{right}(2v))$ и вес $\texttt{cover}(v, \text{left}(2v + 1), r)$, что можно посчитать за $\mathcal{O}(1)$, используя предподсчитанные значения.

Подзадача 3

В случае, когда рассматриваются два соседних листа, V всегда будет разбиваться на два вертикальных пути из $\texttt{lca}(l, r)$ в l и в r. Это означает, что суммарный размер множества V не превосходит $2\log_2 n$, и в нем можно за время $\mathcal{O}(\log^2 n)$ перебрать x и y, либо же воспользоваться методом двух указателей для минимизации максимального из весов трех поддеревьев за время $\mathcal{O}(\log n)$.

Насчитывать размеры поддеревьев x и y можно по ходу перебора, или же можно воспользоваться описанным выше предподсчетом.

Подзадача 4

В подзадаче, в которой веса всех вершин равны 1, не приходится думать о сложном вычислении весов поддеревьев, а можно просто заметить, что любое поддерево в V либо является полным и имеет вес $2^k - 1$ для некоторого целого k, либо имеет корень на пути либо из V в l или из V в r. Следовательно, различных весов поддеревьев в V не более $3\log_2 n$.

Дальше достаточно аккуратно определять, для каких k в V найдется полное поддерево веса $2^k - 1$ (для этого надо быстро находить самую высокую вершину с полным поддеревом в V простым спуском из $\text{lca}(l, r)$ влево и вправо), и для каждого запроса смотреть на $\mathcal{O}(\log n)$ возможных размеров поддеревьев, которые можно от V отрезать, что даст время работы решения $\mathcal{O}(m \log^2 n)$ или $\mathcal{O}(m \log n)$, если также воспользоваться двумя указателями.

Надо только аккуратно обрабатывать тот факт, что запрещается выбирать x и y, являющиеся предком и потомком, поэтому стоит отдельно запомнить все возможные k из левого и правого поддеревьев v, и перебирать в качестве x опции из левого поддерева, а в качестве y — из правого. Отдельно надо учесть опции, в которых x и y располагаются по одну сторону — либо аккуратным рассмотрением вариантов структур деревьев, либо воспользовавшись трюком, описанным в конце решения следующих подзадач.

Подзадачи 5 и 6

Для полного решения или близких к нему требовалось быстрее считать различную вспомогательную информацию или сами ответы. Поскольку запрещено выбирать x и y, которые являются предком и потомком, мы хотим независимо вырезать из V два поддерева, чтобы их веса и вес того, что останется, были как можно ближе друг к другу. Как мы уже отметили, для этого достаточно выбирать поддеревья x и y веса, наиболее близкого к трети веса V, хотя это утверждение требует уточнения (см. ниже).

А пока рассмотрим два случая — либо x и y лежат по одну сторону от $v = \texttt{root}(V)$, либо по разные. Если они находятся по разные стороны, то, не теряя общности, x в левом поддереве, а y — в правом. Ответим тогда сразу на все запросы, для которых $a = \texttt{lca}(l, r)$ равен текущей вершине. Для этого независимо найдем слева x, наиболее близкий к трети веса V, в левом поддереве v, и y — в правом.

Для этого запомним для каждого запроса его l, и будем двигаться по листьям от «центрального левого» ($c_1 = \text{right}(2v)$) влево, и для каждого листа i будем пересчитывать $\texttt{cover}(v, i, c_1)$. Среди всех вершин этого множества мы хотим уметь в произвольный момент выбирать дерево, наиболее близкое по весу к определенному значению. Заметим, что все множество $\texttt{cover}(v, i, c_1)$ состоит из «полных» вершин, для которых каждая вершина поддерева тоже лежит в этом множестве, и «неполных», для которых какие-то из вершин поддерева пока еще в это множество не вошли. «Неполных» вершин не более $\log_2 n$, а остальные можно хранить вместе с весами их поддеревьев в каком-нибудь дереве поиска, позволяющем искать ближайший по значению элемент за логарифм (например, в декартовом).

Тогда для каждого листа i: проходим по его предкам, обновляем веса их поддеревьев, ставшие «полными» закидываем в дерево поиска. После, если у какого-то запроса $l = i$, ищем поддерево нужного веса в дереве поиска, и проверяем еще $\mathcal{O}(\log n)$ «неполных» поддеревьев вручную. Аналогичный процесс выполняем и для правых границ, только идем, начиная с листа $c_2 = \text{left}(2v + 1)$ вправо до $\text{right}(v)$.

Уточнение про веса: на самом деле слова про «вес, наиболее близкий к трети», не до конца верные. Можно доказать, что хотя бы у одного из x и y вес поддерева должен быть одним из двух ближайших к $\frac{\texttt{weight}(V)}{3}$ значений. Но если одна из вершин уже выбрана, например, x, то y надо выбирать с весом поддерева, наиболее близким к $\frac{\texttt{weight}(V)-\texttt{subtree}(x)}{2}$. Поэтому:

1. сделаем проход по листьям в левом поддереве v, найдем кандидатов на x с весом, наиболее близким к трети веса V (ближайший не меньший и ближайший не больший);
2. сделаем проход по листьям в правом поддереве v и для каждого соответствующего кандидата x найдем наиболее подходящий y; параллельно найдем кандидатов на y с весом, наиболее близким к трети веса V;
3. сделаем еще один проход по левому поддереву, и найдем для каждого кандидата y наиболее подходящий x;
4. из всех найденных пар кандидатов (x, y) выберем оптимальную пару.

Время работы такого решения — $\mathcal{O}(n \log^2 n + m \log n)$, первое слагаемое берется из того, что мы каждую вершину положим один раз в дерево поиска во время обработки каждого из ее предков, а второе — из ответа на запросы.

Осталось только находить ответы на запросы, для которых x и y лежат по одну сторону $v = \text{lca}(l, r)$. Для этого заметим, что такое может быть выгодно только если вес какой-то из частей V по одну сторону от v, включая вес v, меньше текущего найденного ответа. Такое может случиться только для одной из двух сторон (левая или правая), поэтому «отложим» этот запрос в соответствующего ребенка v.

Для этого будем представлять наши запросы не как (l, r), а как (l, r, Δ), где для запросов из входных данных $\Delta = 0$. Это будет означать, что мы минимизируем не $\max(w_v, w_x, w_y)$, а $\max(w_v + \Delta, w_x, w_y)$. «Откладывая» запрос ниже по дереву, увеличим его Δ на вес той части, которую мы оставили «сверху» — потом эта часть будет присоединена к верхней из трех частей, полученных из отложенного запроса. Отвечать на такие запросы можно тем же образом, который описан выше, но теперь надо искать веса, наиболее близкие к $\frac{\texttt{weight}(V)+\Delta}{3}$. Время работы ответов на запросы тогда становится $\mathcal{O}(m \log^2 n)$, потому что каждый запрос порождает не более $\log_2 n$ новых запросов, отложенных вниз по дереву.