Разбор задачи «Максимальное произведение»

Автор задачи: Даниил Орешников

В первой подзадаче можно просто перебрать все возможные точки разреза, для каждой найти сумму элементов, перемножить и выбрать оптимальное значение. При этом сумму можно каждый раз вычислять заново, получив время работы $O(n^2)$. Благодаря тому, что сумма элементов всего массива не превышает 10^9, все действия можно безопасно выполнить в 64-битном типе данных.

Для решения четвертой подзадачи можно заметить, что вместо суммирования элементов заново, каждый раз при перемещении границы вправо на один элемент можно пересчитывать сумму за $O(1)$. Следовательно решение работает за $O(n)$.

Для решения остальных подзадач метод просуммировать элементы и перемножить не подходит — произведение получается слишком большое и не влезает в 64-битный тип данных. Заметим, что решения на языке Python или в случае доступности 128-битного типа данных, в принципе, лишены этого недостатка. Но есть решение, которое не выполняет операций умножения.

Решим сначала подзадачи, где все значения равны. Пусть массив поделен на части размера k и l. Заметим, что произведение равно $(k \cdot a) \cdot (l \cdot a)$, где a — значение элементов массива, а $k+l=n$. Так как $(ka)(la) = (kl)a^2 = k(n-k)a^2$, достаточно максимизировать величину $k(n-k)$. Эта величина максимальна, когда k примерно равно $n/2$, а именно равно $n/2$ при чётном n и $(n \pm 1)/2$ при нечётном n. Таким образом получаем решение за $O(1)$: вывести $\lfloor n/2 \rfloor$. Это решение подходит для второй и пятой подзадачи, также можно не находить математически максимум $k(n-k)$, а перебрать значения k. Во второй подзадаче для вычисления $k(n-k)$ при этом достаточно 32-битного типа данных.

Наконец, для решения на полный балл заметим, что идея, что значение $k(n-k)$ максимально при k в районе $n/2$ обобщается следующим образом. Пусть s — сумма всех элементов массива, а t_k — сумма первых k элементов. Тогда произведение $t_k(s-t_k)$ максимально, когда t_k максимально близко к $s/2$. Можно перебирать k и вычислять t_k за $O(n)$ в первых трёх подзадачах, либо вычислять t_k за $O(1)$ с помощью префиксных сумм, или пересчитывать за $O(1)$ при увеличении k на 1, получая полное решение.

Разбор задачи «Планировка участка»

Автор задачи: Андрей Станкевич

Заметим, что если $a > n$, то $ab - cd > ab - ad = a(b-d) \geqslant a > n$ и никакие значения b, c и d не подходят. Следовательно $a \leqslant n$. Аналогично доказывается, что $b \leqslant n$.

Переберём a и b от 2 до n, а также c от 1 до $a-1$ и d от 1 до $b-1$. Проверив для каждого варианта, что $ab - cd = n$, получим решение первой подзадачи за $O(n^4)$, которое несложно модифицируется, чтобы убеждаться, что $a \neq x$ и $b \neq x$, тем самым решая подзадачу 2.

Пусть мы зафиксировали a, b и c. Тогда заметим, что d определяется единственным образом из уравненя $ab - cd = n$: если $ab - n$ кратно c, то $d = (ab-n)/c$, иначе никакие значения d не подходят. Следует также не забыть проверить, что $1 \leqslant d < b$. Получаем решение за $O(n^3)$, оно подходит для подзадачи 3, а добавляя проверки $a \neq x$ и $b \neq x$, также и для подзадачи 4.

Для решения подзадачи 5 можно применить забавный трюк. Давайте подсчитаем предыдущим алгоритмом за $O(n^3)$ решения для всех возможных значений n. Решение за $O(n^4)$ работает очень долго для n порядка 3000, но у нас почти все время тура. Все полученные значения поместим в константный массив. Такой метод известен как *предподсчёт*. К сожалению, сделать предподсчёт для шестой подзадачи не получается, это и слишком долго, и занимает слишком много места, такой большой исходный файл не удастся отправить на проверку.

Наконец, решение за $O(n^2 \log n)$, которое решает задачу полностью.

Вместо c будем перебирать $a - c$. Заметим, что $ab - cd = ab - bc + bc - cd = (a-c)b + c(b-d)$. Поскольку $c > 0$ и $b > d$, второе слагаемое положительно, и следовательно $(a-c)b < n$. Таким образом, достаточно перебирать только $O(n/b)$ значений. Значит для каждого a будет перебираться $n + n/2 + n/3 + \ldots + n/n = n(1 + 1/2 + 1/3 + \ldots + 1/n)$ пар значений. Сумма $1 + 1/2 + 1/3 + \ldots + 1/n$ хорошо известна — это частичная сумма гармонического ряда, она равна $O(\log n)$, следовательно суммарно

для фиксированного a перебирается $O(n \log n)$ значений, а общее время работы алгоритма равно $O(n^2 \log n)$. Финальное замечание, что так как a и b перебираются явно, нет проблемы убедиться, что $a \neq x$ и $b \neq x$.

Разбор задачи «Банкомат»

Автор задачи: Даниил Орешников

Рассмотрим сначала решение первых четырех подзадач, в которых $q \leqslant 5$. В этом случае каждый запрос можно обрабатывать независимо от других, поэтому проигнорируем факт, что у нас несколько значений b_i и будем решать задачу для одного значения b.

В первой подзадаче $b \leqslant 500$, $n \leqslant 500$, поэтому можно применить динамическое программирование. Обозначим как $dp[i]$ количество купюр, которое будет выдано, если запросить сумму i. Тогда $dp[0] = 0$, и если $m(i)$ — максимальный номинал купюры, которая не превосходит i, то $dp[i] = dp[i - m(i)] + 1$. Ответ — максимальное значение среди $dp[1] \ldots dp[b]$. Решение работает за за $O(nb)$.

Чтобы распространить описанное решение на третью подзадачу, избавимся от множителя n в времени работы. Для этого заметим, что $m(i)$ только возрастает при возрастании i, поэтому в процессе вычисления $dp[i]$ можно поддерживать текущее значение $m(i) = a_j$ и при переходе к следующему значению i проверять, нельзя ли перейти к следующему номиналу a_{j+1}. Время работы будет $O(b+n)$ и третья задача подрешена.

Во второй подзадаче номиналы купюр являются последовательными степенями двойки. Заметим, чтобы выдать сумму c банкомат выдаст купюры с номиналами, соответствующими позициям единиц в двоичной записи числа c. Значит задача свелась к поиску числа, не превосходящего b, содержащего максимальное число единиц в двоичной записи. Пусть двоичная запись числа b содержит k разрядов. Тогда число единиц в ответе не превышает k, и число $2^{k-1} - 1$ заведомо меньше b и содержит ровно $k - 1$ двоичную единицу. Значит ответ — либо b, либо $2^{k-1} - 1$, получившееся решение работает за $O(\log b)$.

Решение четвертой подзадачи уже приближает нас к полному решению: ограничения на n, a_i и b_i в ней максимальны и только $q \leqslant 5$. Научимся решать задачу для одного запроса b за $O(n)$. Правильное решение основывается на жадном алгоритме.

Сделаем инверсию задачи. Пускай mn_x — минимальное число c такое, что банкомат выдаст x купюр для суммы c.

Будем по-прежнему обозначать как $m(v)$ максимальный номинал купюры, не превышающий v. Докажем, что $mn_x = mn_{x-1} + m(mn_x)$.

С одной стороны $mn_{x-1} \leqslant mn_x - m(mn_x)$, поскольку из набора купюр, дающего сумму mn_x, всегда можно убрать самую большую купюру (равную $m(mn_x)$), и получим корректный набор из $x-1$ купюр, дающих в сумме $mn_x - m(mn_x)$. С другой стороны, предположим, что мы зафиксировали x, и $mn_{x-1} < mn_x - m(mn_x)$. Можно заметить, что в этом случае величина $mn_{x-1} + m(mn_x)$ меньше, чем mn_x, и банкомат вернет при запросе такой величины ровно x купюр — одну купюру $m(mn_x)$ (она максимальная, не превышающая этой суммы), и еще $x - 1$ купюр, на которые раскладывается сумма mn_{x-1}. Противоречие, исходное утверждение доказано.

Используя это наблюдение, можно посчитать все значения mn_x за время $O(n)$. Заметим, что для двух последовательных чисел a_i, a_{i+1} для всех значений x таких, что $m(mn_x) = a_i$ (иными словами, самая большая купюра, если банкомат выдаёт x купюр, равна a_i), значения mn_x образуют арифметическую прогрессию с разностью a_i и значениями в промежутке $[a_i; a_{i+1})$. Посчитав такую арифметическую прогрессию для промежутка $[a_i; a_{i+1})$, можно перейти к промежутку для $[a_{i+1}; a_{i+2})$: если последнее значение в прогрессии равно y ($y < a_{i+1}$), то следующая прогрессия начнется с y, и будет состоять из чисел $y + k \cdot a_{i+1}$, меньших a_{i+2}. Чтобы найти все эти числа и их количество можно просто поделить длины соответствующих отрезков нацело на a_{i+1}. Таким образом, последнее число mn_x, не превышающее b, и соответствующее значение x являются ответом на задачу.

Для решения последней группы можно заметить, что не обязательно решать независимо задачу для разных b_j, а достаточно найти двоичным поиском последнее $a_i < b_j$, и посмотреть на соответствующую арифметическую прогрессию из значений mn_x.

Разбор задачи «Плакаты»

Автор задачи: Алексей Толстиков

В первой подзадаче можно просто перебрать, кто из друзей поднимет плакаты, для каждого варианта проверить, что нет трёх поднятых подряд плакатов, и посчитать суммарную красочность поднятых плакатов. Время работы решения $O(2^n \cdot n)$.

Во второй подзадаче можно реализовать тот же алгоритм, но повторить его q раз, после каждого изменения. Время работы $O(2^n \cdot nq)$.

Третья и четвертая подзадача решаются с помощью динамического программирования. Заметим, что друзья 1, 2, 3 и 4 не могут одновременно поднять плакат. Переберём, кто из них не поднял плакат, и переставим его и друзей, которые идут до него, в конец нумерации. Теперь стоящий последним друг не поднял плакат, поэтому из задачи на окружности мы получили задачу на прямой. Используем для её решения динамическое программирование: обозначим как $dp[i][j]$ максимальную суммарную красочность плакатов, поднятых первыми i друзьями, причём последние j из них подняли плакат ($0 \leqslant j \leqslant 3$). Тогда $dp[i][0] = \max(dp[i-1][0], dp[i-1][1], dp[i-1][2], dp[i-1][3])$, а для $j > 0$ выполнено $dp[i][j] = dp[i-1][j-1] + a_i$.

В решении третьей подзадачи запустим этот алгоритм $q+1$ раз, для начальной позиции и после каждого изменения, получив решение за $O(nq)$. В четвертой подзадаче один запуск этого алгоритма работает за $O(n)$.

Наконец, для полного решения задачи необходимо научиться изменять величину красочность плакатов и пересчитывать значения $dp[i][j]$. Потребуется чуть более сложная конструкция: построим дерево отрезков на индексах от 1 до n. В узле, соответствующем отрезку от l до r будем хранить матрицу 4×4: $best[a][b]$ равно максимальной суммарной красочности плакатов, поднятых друзьями, с номерами от l до r, в начале a стоящих подряд друзей подняли плакат, а в конце — b подряд стоящих друзей. Тогда объединение двух соседних отрезков происходит по алгоритму, аналогичному пересчёту значений динамического программирования в одной из предыдущих задач. Изменяя значение красочности i-го плаката, нам необходимо пересчитать $O(\log n)$ значений в узлах на пути к листу, соответствующему отрезку от i до i.

Время работы получившегося решения есть $O(n + q \log n)$, напоследок отметим, что, несмотря на небольшие по меркам получившейся асимптотики ограничения, пересчёт значения в узле дерева отрезков получился довольно трудоёмким, поэтому константа, спрятанная в O довольно велика, и для того, чтобы уложиться в ограничение по времени, необходима достаточно аккуратная реализация.