Разбор задачи «Шаттл до Рутнока»

Заметим, что можно добиться того, чтобы в шаттле не было нескольких пустых капсул подряд. Если после нескольких пустых капсул есть капсулы, занятые повстанцами, то можно сдвинуть этих повстанцев в капсулы с меньшими номерами. Если далее нет занятых капсул, то можно разместить в свободные капсулы еще как минимум одного повстанца.

Также можно добиться того, чтобы первая капсула была занята. Если она свободна, можно сдвинуть всех повстанцев на одну капсулу влево, и это не ухудшит ответ.

Два этих замечания позволяют написать простое линейное решение, которое пройдет группы тестов с первой по пятую. Поместим повстанца в первую капсулу, а дальше в цикле будем идти с шагом через одну капсулу и помещать повстанцев туда, пока не дойдем до капсулы n или $n-1$.

Заметим, что каждая вторая капсула в нашем линейном решении оказывается свободной. Если n четно, тогда и свободных, и занятых капсул будет по $\frac{n}{2}$. При нечетном n мы получим $\frac{n+1}{2}$ занятых и $\frac{n-1}{2}$ свободных капсул. Запрограммировав эти формулы, можно пройти шестую и седьмую группы тестов соответственно.

В последней группе тестов число n не помещается в 32-битный целый тип, поэтому для ее прохождения нужно использовать 64-битный целый тип.

Разбор задачи «Побег в империю»

В первой группе тестов доказательство лояльности и оформление удостоверения беженца не требуют времени, поэтому ответ $= x_3$.

Во второй группе тестов нужно сначала получить удостоверение беженца и только после этого подать заявление на оформление паспорта, поэтому ответ $= x_2 + x_3$.

В общем случае, чтобы и доказать лояльность, и оформить удостоверение беженца, нужно $\max(x_1, x_2)$ дней. После этого нужно еще x_3 дня для оформления паспорта. Итого, ответ $= \max(x_1, x_2) + x_3$.

Разбор задачи «Башни из контейнеров»

В первой группе тестов исходная высота единственной башни равна 1, и понадобится $m-1$ контейнер, чтобы сделать ее высоту равной m. Во второй группе тестов $m \geqslant h_1$, поэтому понадобится $m - h_1$ контейнеров.

Посчитаем суммарное количество контейнеров в башнях: $S = h_1 + \cdots + h_n$. Если высота каждой башни не менее m, то в сумме в них не менее nm контейнеров. Так как мы можем распределять контейнеры как угодно, то при $nm \leqslant S$ нам не нужны дополнительные контейнеры (ответ 0). Иначе ответ $= nm - S$.

Для прохождения последней группы тестов нужно учесть переполнение и воспользоваться 64-битным типом данных.

Разбор задачи «Круговая по рукам»

В первой группе тестов тиif забирает единственную монету из первого мешка, и количество ЗЕТов в остальных двух мешках не меняется. Ответ: «0 1 1»

Во второй группе тестов во всех мешках по одному ЗЕТу. Положим x_i равным 1 для всех i от 1 до n и будем обнулять x_i, встречая i в запросах. Тогда после всех запросов x_i будет количеством ЗЕТов в i-м мешке.

Для прохождения третьей группы тестов нужно проэмулировать описанный в условии процесс: забирать из q_i-го мешка один ЗЕТ, а остальные ЗЕТы раскладывать в (q_i+1)-й, (q_i+2)-й и так далее мешки. Каждый запрос мы обрабатываем за $O(x_{q_i})$, где x_{q_i} — количество ЗЕТов в q_i-м мешке к моменту этого запроса.

Для прохождения четвертой группы тестов нужно было заметить, что в процесе эмуляции не обязательно раскладывать все ЗЕТы из мешка по одному: если в мешке осталось x_{q_i} ЗЕТов, то мы можем добавить $\lfloor \frac{x_{q_i}}{n} \rfloor$ во все мешки, а оставшиеся $x_{q_i} \mod n$ разложить в мешки с номерами q_i+1, q_i+2, …, $((q_i - 1 + x_{q_i} \mod n) \mod n) + 1$. Так каждый запрос мы обрабатываем за $O(n)$.

Заметим, что мы совершаем операции трех типов: обнуляем количество ЗЕТ-ов в одном мешке, прибавляем ко всем мешкам одно и то же число ЗЕТ-ов или прибавляем один ЗЕТ к какому-то

отрезку мешков (если происходит переход от n-го мешка к первому, то нужно прибавить один ЗЕТ к двум отрезкам). Вторую и третью операции можно выполнять эффективнее, чем за $O(n)$, если воспользоваться какой-нибудь древовидной структурой данных. Например, дерево отрезков умеет выполнять все три операции за $O(\log n)$. Так можно было пройти пятую группу тестов.

Разбор задачи «Ящик Пандоры»

Наивное решение, которое для каждого k_i проходит по массиву кнопок с шагом k_i и меняет состояние каждой встреченной кнопки на противоположное, проходит первые четыре группы тестов.

Перейдем к полному решению задачи. Для начала заметим, что ответ не зависит от порядка k_i. Кроме того, если два ученых высказали предположения с одинаковым k_i, то применение этих предположений по очереди дважды меняет состояние некоторых кнопок, что равносильно тому, что состояние никаких кнопок не меняется. Тогда можно сгруппировать предположения с одинаковым значением k_i — с помощью сортировки или структуры данных map / dictionary — оба способа работают за $O(n \cdot \log(n))$. После этого останется применить по одному разу лишь те предположения, которые были высказаны нечетное количество раз.

Теперь обработаем оставшиеся предположения k_i наивным способом, описанным выше. Один проход по массиву кнопок совершит $\lfloor \frac{n}{k_i} \rfloor$ шагов. Поскольку все k_i теперь различны, суммарное количество шагов такого алгоритма можно оценить сверху величиной $\sum_{i=1}^{n} \lfloor \frac{n}{i} \rfloor$. Известный факт, что такая сумма имеет порядок $O(n \cdot \log(n))$ — вы можете найти его доказательство в Интернете в статьях про гармонический ряд. Такая оценка позволяет пройти последнюю группу тестов.

Разбор задачи «Кофейная церемония»

Чтобы пройти первые три группы тестов, было достаточно перебрать все возможные подотрезки массива a, количество которых имеет порядок $O(n^2)$. Для каждого отрезка нужно циклом посчитать сумму чисел на нем, после чего проверить, делится ли она на k. Такое решение работает за $O(n^3)$

Оптимизируем данное решение до $O(n^2)$, для этого избавимся от линейного подсчета суммы чисел на отрезке.

Заполним массив p *префиксных сумм* массива a, определенный следующим образом: $p[i]$ = сумме всех чисел в массиве a на отрезке $[0, i]$, где $0 \leqslant i < n$. Это можно сделать за линейное время по правилу: $p[0] = a[0]$; $p[i] = p[i-1] + a[i]$, при $0 < i < n$. С помощью префиксных сумм мы можем за $O(1)$ вычислять сумму на произвольном отрезке массива следующим образом: $sum([0, right]) = p[right]$, $sum([left, right]) = p[right] - p[left-1]$ при $left > 0$. Решение, перебирающее все подотрезки массива a и считающее сумму по описанным формулам, проходит и четвертую группу тестов.

Чтобы пройти пятую и шестую группу тестов, нужно применить ту же самую идею, но рассматривать префиксные суммы по модулю k. Заметим, что для любых i, j, где $i < j$, верно $p[j] \mod k = p[i] \mod k$ тогда и только тогда, когда сумма чисел на отрезке $[i+1, j]$ делится на k.

Значит, задача сводится к тому, чтобы найти количество пар таких, что $p[i] \mod k = p[j] \mod k$ и $i < j$. Это можно сделать, например, двигаясь слева направо и поддерживая в массиве cnt с индексами от 0 до $k-1$ количество каждого остатка префиксных сумм. Встретив префиксную сумму $p[i]$, добавим к ответу $cnt[p[i] \mod k]$, после чего увеличим значение $cnt[p[i] \mod k]$ на единицу. К итоговому ответу нужно не забыть прибавить финальное значение $cnt[0]$ — оно соответствует подходящим отрезкам, начинающимся в нулевом индексе.

Такое решение работает за $O(n)$ по времени и требует $O(k+n)$ памяти. Шестая группа тестов отличается от пятой тем, что значения префиксных сумм в ней не помещаются в 32-битный целый тип.

В последней группе тестов большое значение k не позволяет хранить все возможные остатки в массиве cnt. Здесь вместо массива нужно использовать структуру данных map / dictionary из стандартной библиотеки языка программирования. Эта структура занимает $O(n)$ памяти и, в зависимости от реализации, обеспечивает доступ к элементу за время $O(1)$ или $O(\log(n))$. Таким образом, мы получаем сложность итогового решения $O(n \log(n))$ по времени и $O(n)$ по памяти.