Разбор задачи «Основная причина»

Решать эту задачу можно было разными способами.

Поскольку ограничения позволяют не задумываться об эффективности, допустим, например, следующий подход. Заведем список причин $problems$, который будем пополнять каждый раз, когда появляется причина, отличная от уже имеющихся в нем. Также будем поддерживать еще два списка, синхронизированных со списком причин — $frequency$ и $lastnum$. Таким образом, $problems[i]$ будет содержать собственно причину, $frequency[i]$ — количество раз, которое эта причина встретилась во входных данных, $lastnum[i]$ — номер последнего вхождения этой причины.

Обновление будет происходить следующим образом: пусть мы прочитали некоторую очередную причину $\#j$. Пойдем по списку $problem$ и будем последовательно сравнивать ее со всеми имеющимися в списке. Если причина совпала с одной из имеющихся, например с причиной, записанной в списке под номером $\#z$ (обратите внимание, что номер причины во входных данных и номер причины в списке — это разные сущности), то соответствующее ей значение $frequency[z]$ должно быть увеличено на 1, а $lastnum[z]$ получит значение j.

Когда все причины будут обработаны, достаточно будет просмотреть список причин и найти там максимальное значение $frequency$; если же таких значений окажется несколько, то еще одним проходом можно найти то, для которого $lastnum$ будет наибольшим.

Разбор задачи «Меньшее из зол»

Посчитаем суммарный рейтинг для каждого объекта — для этого достаточно просуммировать числа в соответствующих столбцах.

Далее, зафиксировав два объекта $\#i$ и $\#j$, мы можем посчитать количество голосов, поданных за снос каждого из них. Будем перебирать пары, сохраняя ту, в которой разность между количеством голосов окажется наибольшей. Если очередная пара имеет такую же разность между количеством голосов, что и сохраненная, сравним их по разнице суммарных рейтингов.

Асимптотика решения — $O(n^2 \cdot m)$, что вполне подходит для данных ограничений.

Разбор задачи «Консервативная стратегия»

Задачу можно отнести к классу реализационных.

Ограничения позволяли перебрать все сезоны строительства в цикле (в худшем случае, когда каждый из 100 этапов делался бы порядка 10^5 сезонов).

Для ответа будем хранить количество сезонов и список из номеров этапов в каждом сезоне (этот список мы перестанем обновлять, если количество сезонов превысит 10^5).

Также для каждого сезона будем поддерживать количество единиц времени, которое еще не израсходовано.

Теперь будем рассматривать очередной этап строительства. В этом рассмотрении можно выделить три ключевых момента:

- если оставшегося в сезоне времени не хватает на то, чтобы начать и завершить этап целиком, и потребуется консервация, то нам нужно проанализировать, стоит ли тратить время на консервацию и последующую расконсервацию или же сразу начать этап со следующего сезона. Как понятно, если в текущем сезоне мы потратим времени на строительство (учтите, что еще придется тратить время на консервацию) меньше, чем на расконсервацию в следующем сезоне, то начинать этап в текущем сезоне не стоит.

- если этап достаточно длинный и в принципе не может быть завершен за один сезон, то у нас будет не менее одного сезона, каждый из которых придется начинать с расконсервации, затем выполнять какое-то количество работы, после чего (при необходимости) снова консервировать объект. Если в течение целого сезона мы не сможем потратить на выполнение работы хотя бы одну единицу времени, мы должны будем вывести -1.

- когда этап завершен, нужно обновить количество единиц времени, оставшееся в текущем сезоне.

Разбор задачи «Типичная ситуация»

Будем решать задачу с помощью бинарного поиска по ответу. Предположим, что типовой проект бассейна рассчитан на некоторое количество жителей b и выясним, можно ли распределить жителей всех районов так, чтобы они могли посещать бассейн в своем или одном из соседних районов.

Распределять жителей будем с помощью жадного алгоритма.

Начнем с первого района. Если $a_1 \leqslant b$, то «отправим» всех жителей этого района в бассейн в этом же районе. Если же $a_1 > b$, то оставшихся $a_1 - b$ жителей «отправим» в бассейн района #2.

С жителями второго района поступим следующим образом. Если в бассейне района #1 еще остались вакантные места ($a_1 < b$), то займем все вакантные места жителями района #2. Затем, если есть еще жители, которые не записаны в бассейн, запишем их в бассейн района #2 . Если же есть еще жители, которые в бассейн не записаны, их придется «отправить» в бассейн района #3.

Будем продолжать этот процесс, учитывая следующее: если окажется, что мы уже исчерпали вместимость бассейна в районе $\#(i+1)$, а среди жителей района $\#i$ у нас все еще остались те, кто в бассейн не записан, то предлагаемая вместимость бассейна b не может считаться подходящей.

Описанный алгоритм распределения работает за $O(n)$, поэтому нам и понадобится бинарный поиск: перебирать все возможные варианты вместимости бассейна может оказаться очень долго. Однако поскольку функция «можно ли распределить жителей всех районов, если типовой бассейн будет вместимости b» является монотонной неубывающей относительно параметра b. В самом деле, если мы смогли распределить жителей в случае, когда бассейн имеет вместимость B, то и любой больший бассейн тоже будет подходящим. Если же не смогли — то бассейн меньшей вместимости тем более не годится. Этот факт и позволяет нам воспользоваться бинарным поиском; в качестве начальных границ бинарного поиска можно взять 0 и $a_{max} = \max\limits_i a_i$.

Асимптотика решения $O(n \cdot log a_{max})$.