Ловушка для Джерри

Автор задачи и разработчик: Даниил Орешников

Для начала отметим какими способами можно было получить частичные баллы за решение отдельных групп тестов, а затем рассмотрим полное решение:

1. В первой подгруппе $n \cdot k$ не превосходит 10^5, что означает, что каждый запрос можно обрабатывать за $\mathcal{O}(n)$, просто целиком проходя по массиву.

2. В случае, когда $|a_i|$ изначально не превосходят 10, можно посчитать количество платформ каждой высоты в начальный момент времени, и аккумулировать суммарное изменение высот — тогда ответ на запрос суммы потребует один раз пройти по массиву количеств, который имеет размер 21.

3. Следующая группа уже достаточно приближена к полному решению. Во-первых, следовало отсортировать высоты платформ. Во-вторых, как и в предыдущей группе, мы не будем в явном виде изменять эти высоты, а будем в отдельной переменной `add` хранить, на какую величину высоты изменились с начального момента времени.

   Теперь стоит заметить, что если $a_i + \texttt{add} < 0$, то соответствующая опасность равна $-(a_i + \texttt{add})$, а иначе — $a_i + \texttt{add}$. Таким образом, за счет того, что высоты платформ теперь отсортированы, для нескольких первых i будет использована первая формула, а для всех оставшихся — вторая. Если ровно для t первых элементов a выполняется, что $a_i < -\texttt{add}$, мы получим формулу для суммарной опасности

   $$(-a_1 - \ldots - a_t - t \cdot \texttt{add}) + (a_{t+1} + \ldots + a_n + (n - t) \cdot \texttt{add})$$

   Суммы первых или последних a_i с нужным знаком можно получать за $\mathcal{O}(1)$, если заранее посчитать префиксные суммы на массиве a. Остается только вопрос: как быстро находить t? Для третьей группы было достаточно заметить, что если `add` не уменьшается, то t тоже не уменьшается, поэтому достаточно было завести указатель, и проверять, не надо ли его сдвинуть вправо, после каждого действия.

4. И это же приводит нас к полному решению — абсолютно все действия повторяем, как и в предыдущей группе, однако пользуемся двоичным поиском, чтобы искать это самое t (количество чисел, меньших $-\texttt{add}$) при каждом запросе.

В конечном итоге мы получили решение с $\mathcal{O}(n \log n)$ времени на предподсчет (за счет сортировки массива) плюс $\mathcal{O}(k \log n)$ из-за двоичного поиска, используемого после каждого запроса.