

Analysis: Fairland

We need to find a salary X such that all salaries at the company are between X and $X+D$, and the number of remaining employees is maximized.

If employee i is kept, then X must be in the interval $[S_i-D, S_i]$. Not only that, but X must be in the interval $[S_j-D, S_j]$ for each employee j who is the manager of employee i , or the manager of the manager of employee i , and so forth. So for each employee, we want to find the intersection of all those intervals. This is easily done using a preorder traversal of the tree, computing the intersection of all the intervals in the path from the root node to the current node. (If the intersection is empty, we can't keep that employee.)

Now we want to find the maximum number of intervals that overlap at any point. Let the interval we get for employee i be $[A_i, B_i]$. Create an array of pairs of integers, with two pairs for each employee: one containing $(A_i, +1)$ and the other containing $(B_i, -1)$. Sort this array by the first numbers of the pairs, breaking ties using the second number, by putting entries with $+1$ before those with -1 . Then we start a counter at zero, and iterate through the array in sorted order. Add the second number of each pair to the counter. The maximum value of the counter is the answer to the problem.

Analysis: Smoothing Window

One approach towards a solution is to construct the temperature sequence that begins with $K-1$ zeroes, followed by a single pass through all the $N-K+1$ smoothing window sums to fill in the rest of the sequence. For example, running this construction for the sum array $\{0, 12, 0, 12, 0\}$ with $K = 3$ (from the third sample case) produces the sequence $\{0, 0, 12, -12, 12, 0\}$, where the difference between the maximum and minimum temperatures is 24. To reach a sequence that minimizes this difference, we tweak some values in the sequence as explained below.

Let's define $\text{GROUP}(i)$ where $(0 \leq i < K)$ as the group containing the $(i \% K)$ -th temperatures in the sequence. The sequence $\{0, 0, 12, -12, 12, 0\}$ above is grouped as follows:

1. $\{0, 12, 0\}$ from the first, fourth, and seventh temperatures
2. $\{0, -12\}$ from the second and fifth temperatures
3. $\{0, 12\}$ from the third and sixth temperatures

Changing the i -th temperature by Z degrees means that we need to compensate for this addition with other temperatures in the sequence. One way is by adding Z to the other temperatures in the same group, and subtracting Z from all the temperatures in another group. For example, to increase the first temperature reading by 3 degrees, we also increase the fourth and seventh temperature readings by 3, and reduce the temperatures in the second (or third) group by 3. The resulting sequence is $\{3, -3, 0, 15, -15, 12\}$. Notice that its sum array stays the same: $\{0, 12, 0, 12, 0\}$.

Now let's define:

- $\text{lo}(i)$ as the minimum element in $\text{GROUP}(i)$
- $\text{hi}(i)$ as the maximum element in $\text{GROUP}(i)$
- $\text{interval}(i)$ as a range $[\text{lo}(i), \text{hi}(i)]$
- $\text{SHIFT}(i, y)$ as an operation that adds y to each member of $\text{GROUP}(i)$.

We can remodel the original problem as follows:

You are given K intervals, where the i -th interval spans from $\text{lo}(i)$ to $\text{hi}(i)$. You can perform any number of interval adjustments by choosing i, j , and y ($0 \leq i, j < K$), shifting the i -th interval by y (i.e., $\text{SHIFT}(i, y)$), and shifting the j -th interval by $-y$ (i.e., $\text{SHIFT}(j, -y)$). Find a sequence of adjustments (where order doesn't matter) that minimizes the covering range of all intervals, where the covering range of the intervals is $\max\{\text{hi}(i)\} - \min\{\text{lo}(j)\}$ for all $0 \leq i, j < K$. The minimum covering range is equivalent to the smallest possible difference between the minimum and maximum temperatures in the original problem.

Another important insight is that the shifting can be done independently and can be aggregated. That is, we can accumulate all positive shifts and do it in bulk, similarly with the negative shifts. Therefore, we can "normalize" all intervals by shifting them such that the lower bound of all intervals becomes 0. Denote Q to be the total sum of shifts, which is the sum of all $\text{lo}(i)$ where $0 \leq i < K$. For example, if we have two intervals $[-10, -8]$ and $[333, 777]$, after normalization they become $[0, 2]$ and $[0, 444]$ with $Q = -10 + 333 = 323$. Finally, we have to shift back the normalized intervals by Q degrees, but with the flexibility to distribute the shifts such that the covering range of all intervals is minimized.

Notice that if we increment (or decrement) all the intervals by one, the relative positioning of each interval to one another won't change and thus it does not change the covering range of the intervals. Using this insight, we can reduce Q to $Q \% K$ by distributing the excess shifts Q evenly among the intervals without affecting the final answer. When Q is negative we can keep adding K to Q until it is non-negative.

Suppose we have three normalized intervals $[0, 4]$, $[0, 9]$, and $[0, 7]$, and the total sum of shifts Q is 40. We can distribute back 39 shifts evenly to all three intervals where each interval is shifted by 13, ending with $[13, 17]$, $[13, 22]$, and $[13, 20]$. Then, we can shift all of them down back to $[0, 4]$, $[0, 9]$, $[0, 7]$ without changing their relative positioning and we are left with $Q = 1$.

Finally, we must distribute back the remaining shifts to the intervals and minimize the covering range of all intervals.

Suppose that L is the size of the largest interval. If $Q = 0$, then there is no excess shift to distribute, and the minimum covering range is L . Otherwise, we can assign $L - (hi(i) - lo(i))$ excess shifts to interval i without changing the answer. Using the previous example, the current minimum difference is $L = 9$. Without changing the minimum covering range, we can assign 1 excess shift to interval $[0, 4]$, and get $[1, 5]$.

If there are still some excess shifts remaining, it means we cannot distribute any more excess shifts without increasing the minimum covering range. Increasing the minimum covering range by 1 will make room for distributing K more excess shifts, which is enough for the remaining Q (since $Q = \text{sum of shifts} \% K$, and thus it is less than K).

The complexity of this solution is $O(N)$.

Analysis: Runaway Quail

The optimal strategy will consist of running either to the left or right until we catch a particular quail, changing direction, then running until we catch a particular quail on the other side, changing direction again, etc., until we have caught every quail. Changing direction at a time other than when catching a quail would be wasteful.

Consider a partial solution where we have caught some of the quail on each side, and returned to the origin. For each side, let Q be the fastest quail we have not caught. (To simplify the analysis, if there is more than one quail on the same side with the same speed, we ignore all but the farthest one.) Each other quail R on that side must fall into one of these categories:

- R is faster than Q . In this case, R has already been caught.
- R is slower than Q , and is farther away. R cannot have been caught in this case, because we would have had to run past Q to get to R .
- R is slower than Q , and is not farther away. In this case, we might have already caught R , or we might not. But it doesn't affect our solution — if we have not yet caught R , we will run past R when we go to catch Q anyway.

So we can solve this problem with dynamic programming, where the states only need to include the identity of the fastest uncaught quail in each direction. We will keep the fastest time at which we can achieve each state we generate.

For each state, starting from the initial state where we have caught nothing, we try running in either direction until we catch the fastest uncaught quail in that direction, and then run back to the origin. We also try running to catch each quail farther than the fastest uncaught quail in each direction, and then running back. For each of these options, we compute the new fastest uncaught quail in the direction we ran, which will tell us what the new state is when we return to the origin. If we have achieved the new state with a faster time than we had for it before, we update the time for that state.

Whenever we generate the state where every quail has been caught, we check the time at which we caught the last quail. The fastest of these times is the answer.

Analysis: Log Set

Let x be the largest element of S' . Then x is the sum of all of the positive elements of S , and none of its negative elements.

Let y be the second-largest element of S' . Either this is the sum of all but the smallest of the non-negative elements, or it is the sum of all of the non-negative elements and the smallest negative element. So if $d = x - y$, then the smallest magnitude of any element in S is d — either $d \in S$ or $-d \in S$.

These two cases are basically identical. Consider the process of building every subset of S , and finding the sum of each of those subsets. To build a subset, we consider each element of S , and either include it in the subset or not. If we include it, the sum increases by the value of the element. If we don't include it, the sum increases by zero. So in the case where d is an element of S , we will be choosing between adding 0 and adding d to the sum. If $-d$ is an element of S , we will be choosing between adding $-d$ and adding 0. These situations are equivalent, except that every element of S' is offset by a constant depending on whether d or $-d$ is in S .

We can now scan through S' and match elements into pairs that are d apart, and keep only the smallest of each pair. Then we can recursively apply this procedure to get the magnitudes of all the other elements of S .

When we are done, we have the magnitudes of the elements of S . Next we need to figure out which elements are positive and which are negative. First, make them all negative. The largest element of S' will be zero in this case. Then we need to choose which elements to change from negative to positive in order to make the largest element of S' equal to x . This is a Subset Sum problem, with required sum x , and elements equal to the magnitudes. This is easily solvable with dynamic programming — the number of unique sums, and hence the number of states in the dynamic program, is guaranteed to be small because the number of unique values in S' is small.

Analysis: River Flow

Assume the farmers are not cheating. The river flow data is periodic, with period $2D$.

Let x_i be $d_i - d_{i-1}$ when $i > 1$, and $x_1 = d_1 - d_{2D}$. x_i represents the difference in flow between day i and the previous day in the $2D$ -day cycle.

Each farmer's actions change x_i in the following way. Let the first time a farmer changes between diverting water and allowing it to flow, or vice-versa, be T , and let the number of days for which that farmer performs the same action be P . At times $T, T+P, T+2P$, etc., the farmer will alternate between adding 1 to x_i and subtracting 1 from x_i . (Whether it adds 1 or subtracts 1 first depends on whether the farmer is diverting water or allowing it to flow at the start of the $2D$ -day cycle). For example, a farmer that starts diverting water at time 3, and changes every 8 days, contributes -1 to x_3 , $+1$ to x_{11} , -1 to x_{19} , $+1$ to x_{27} , etc.

We can assume that if two farmers share the same T and P , they either both start the cycle diverting water or both start the cycle allowing it to flow. If they were performing different actions, we could remove the two farmers and their tributaries, and get the same data.

Now consider the quantity $F(T,P) = x_T - x_{T+P} + x_{T+2P} - x_{T+3P} + x_{T+4P} - \dots$ for some T and P . Any farmers with those values of T and P will each contribute $2D/P$ or $-2D/P$ to $F(T,P)$, depending on which action they perform first. Any farmers with a different value of T or P will contribute zero to $F(T,P)$. So the number of farmers for T and P is $|F(T,P)| * P / 2D$, and the sign of $F(T,P)$ tells us their initial action.

We can try every valid value of T and P to find out how many farmers of each type there are. Once we know this information, we can check that the original data matches it for some number of tributaries. If it does not, we output "CHEATERS!", otherwise we output the number of farmers.