A: Gold rush

Let r_i, c_j and d_{j-i+n} be the estimated depths of gold deposits for row, column and diagonal of sector (i, j). To solve the problem we would like to know the value

$$R = \sum_{i=1}^{n}\sum_{j=1}^{m}\left[\texttt{max}(r_i, c_j, d_{j-i+n}) - \texttt{min}(r_i, c_j, d_{j-i+n})\right].$$

To achieve this we will calculate two expressions

$$S = \sum_{i=1}^{n}\sum_{j=1}^{m}\left[r_i + c_j + d_{j-i+n}\right]$$

and

$$T = \sum_{i=1}^{n}\sum_{j=1}^{m}\left[2 \cdot \texttt{min}(r_i, c_j, d_{j-i+n}) + \texttt{mid}(r_i, c_j, d_{j-i+n})\right],$$

where `mid` returns median of given values. Note that $R = S - T$.

To obtain S we can iterate over all estimations and multiply them by the number of sectors they apply to. So we would multiply row estimations by m, column estimations by n and diagonal estimations by an appropriate value between 1 and $\texttt{min}(n, m)$.

Note that

$$T = \sum_{i=1}^{n}\sum_{j=1}^{m}\left[\texttt{min}(r_i, c_j) + \texttt{min}(r_i, d_{j-i+n}) + \texttt{min}(c_j, d_{j-i+n})\right].$$

To obtain T we can solve the following problem. Given only two types of estimations calculate sum of minimums over every sector.

Let's resolve this subproblem given estimations for rows and diagonals. Other pairs are solved similarly. We use two segment trees. One for rows and one for diagonals. Each segment tree will store the information on how many sectors in a given row (diagonal) was not assigned a smaller diagonal (row) estimation, e.g. segment tree for rows will start with value m in every leaf. We iterate over all the estimations for rows and diagonals in an ascending order. Let r_i be an row estimation under consideration. We query segment tree for rows (i-th position) to find the number of sectors that were not assigned a value yet. Let x be the result of the query. We add $r_i \cdot x$ to the result of the subproblem. Before moving to the next estimation we have to update the segment tree for diagonals. We add -1 to all the diagonals that have a common sector with considered row. We follow a similar procedure when considering diagonal estimations.

Solving this subproblem for every pair of estimation types gives as T. So in the end we can return R as the result.

Time complexity: $O((n + m)\, log\, (n + m))$.

B: Krzysztof and UZI

One can notice that if n is a leap year, $n + 400$ is also a leap year. The opposite is also true, i.e., if n is not a leap year then $n + 400$ is also not a leap year. Since the structure of leap years repeats every 400 years, and the structure of UZI competitions repeats every k years, we can see that the structure of "UZI and leap years" repeats every $400k$ years. Because of this, it is enough to brutally check the first $400k$ years, because the later ones have exactly the same gaps between consecutive competitions.

C: Henry Porter and the Hedge of Secrets

First, notice how a spell of power k increases exactly k hedge heights by at least a factor of 2. Hence, the sum of powers of all spells we can ever perform without some of the height exceeding $C = 10^{18}$ is $\mathcal{O}(n \log C)$. This means that if we're able to detect a spell which (without loss of generality) has to be performed in any optimal solution, we can just naively iterate through the affected hedge segments and apply the spell; as long as we break if some height exceeds C, the total time spent on applying spells will be acceptable.

It seems that some kinds of spells "interfere" with each other, but we can make a simple observation to resolve that: if we ever want to perform a spell of power $a \cdot b$ (for some $a, b > 1$), we can instead perform b^2 smaller spells of power a (or, equivalently, a spells of power b and then b spells of power a). Since we don't care about the exact number of spells, this means we can assume we only ever perform spells of power k if k is a prime number. Now, the problem simplifies significantly, as each prime number that divides some of the a_i's could be treated independently. One approach is thus to try to factorize everything, and go from there, and the problem can indeed be solved in this way; however, here we'll focus on the shortest solution, which doesn't "separate" the different primes up front.

Let's choose an index i, and define $b = \frac{a_i}{gcd(a_i, a_{i+1})}$. Note that b contains all the prime divisors that appear in a_i more than they do in a_{i+1}, and the only spell that affects a_{i+1} without affecting a_i is one that starts at position $i + 1$. Thus, we can simply iterate through all prime divisors of b, and for each divisor p apply a spell of power p starting at position $i + 1$ (if a divisor appears in a power larger than one, then we'll just apply the spell several times). We can now go through all i's from left to right and perform this operation, and then again from right to left (simply done by reversing the sequence and re-running the same procedure); we can see that either something will break (a hedge segment height will exceed C, or we will try to apply a spell that "doesn't fit") or all a_i's will end up having the same value, at which point that value is our answer.

The only question is, how efficient is the above approach? First of all, we need to factorize values of the form $b = \frac{a_i}{gcd(a_i, a_{i+1})}$. We can do this naively by iterating through all potential divisors, as long as we break if we get to n (if we haven't fully factorized b at this point, that means we would need to perform some spells of power larger than n, which is never going to be possible). If we do end up breaking after reaching n, then the answer is NO, so this case can only happen once per test case. Otherwise, our naive factorization loop will go up to the largest prime divisor of b, which may be relatively large. However, note that after we loop for p iterations to find $p \mid d$, we also apply a spell of power p, and (as said at the beginning of this editorial), the total work spent applying spells is bounded. Thus, this naive factorization is actually enough! (As a side note, it is possible to factorize all a_i's up front by computing their LCM, which has to be no larger than C for the answer to exist, factorizing that using an algorithm such as Rho-Pollard, and then using the obtained set of prime numbers to factorize a_i's. But all this is just not necessary.)

Summing up, we get an algorithm with running time $\mathcal{O}(n \log C)$, which ends up being remarkably simple to implement (with the core solution taking up less than 25 lines of code).

D: Average problem

Let $\texttt{pref}(i)$ return the prefix of s of length i. We start by determining the length of the maximal pufix for every prefix of s (result stored in $\texttt{pufix}[i]$). We build a directed tree by adding an edge $(\texttt{pufix}[i] \to i)$ for every prefix $(i \in [1, |s|])$.

Note that iterating internal vertices of the path from root to the vertex i iterates all lengths of pufixes for the i-th prefix.

1°. There is an internal vertex which value is not the length of pufix of the i-th prefix. That cannot be true since pufix of a pufix is a pufix.

2°. There is a pufix p of i-th prefix which length is not covered by the internal vertices. Suppose its length is between two vertices u and v connected by an edge $(u < v)$. Then p has to be a pufix of $\texttt{pref}(v)$. But $\texttt{pref}(u)$ was the longest pufix of $\texttt{pref}(v)$, thus leading to a contradiction.

To get the result for every vertex v we want to know the value of the vertex that is in the middle of the path from root to v. To do so we run DFS maintaining all the vertices on the path from root to the current position.

Time complexity: $O(|s|)$.

E: The Defense of Aiur

There are two ways that two optimal Templar may meet:

1. Both Templar go straight towards each other.
2. Each Templar goes towards the Pylon closest to them. The one that reaches their respective Pylon first teleports to the second Pylon. After teleporting, the Templar move towards each other.

First, let's solve the easier case, where the Templar don't use Pylons. Here, the answer is the distance between the closest pair of Templar (the distance should be divided by two, but without loss of generality we will skip this division during the rest of the solution). We can compute the distance between the closest pair of points using either the classic divide-and-conquer algorithm, or a modified sweep-line, both of which work in $O(n \log n)$ where n is the number of points. From now on, let's assume that the two closest Templar are d units away from each other.

In the second case, if we compute the closest Pylon for each Templar, we just need to add the two smallest of these Templar-Pylon distances to get the answer. This is a well known problem which we can solve using either kd-trees or a modified Delaunay

Triangulation. Both of these approaches work in $O(n \log n)$ but are hard to implement well within the time of the competition.

To simplify our solution, we may notice that, for a given Templar, there is no need to consider Pylons farther than d units away, because they for sure won't let us improve upon the bound of d. This means that it is sufficient, for a single Templar, to only check Pylons within a square of side length $2d$ centered on the Templar. It turns out that if we do this, a single Pylon will be checked by a constant number of Templar (proof below). This can be implemented with a sweep-line in $O(n \log n + m \log m)$ time (m denotes the number of Pylons).

Now, let's prove why each Pylon will be checked a constant number of times. Without loss of generality, let's assume that, for a given Templar, we will check for Pylons in a circle of radius $\sqrt{2}d$ around it, instead of a square of side length $2d$ (the square is completely within the circle, so in the worst case we will over count).

Let's fix a Pylon P. Now, let's draw two circles of radii $\sqrt{2}d$ and, $(\sqrt{2}+\frac{1}{2})d$ respectively, that are centered on P. For a Templar to check P it must be inside the smaller circle. For each Templar inside the aforementioned circle, let's draw a circle of radius $\frac{d}{2}$ around it. Since there are no Templar closer than d to each other, it follows that the circles of radii $\frac{d}{2}$ don't intersect. These circles are completely inside the circle of radius $(\sqrt{2}+\frac{1}{2})d$. Since the circle of radius $(\sqrt{2}+\frac{1}{2})d$ has an area of $(\sqrt{2}+\frac{9}{4})\pi d^2$, and each circle of radius d has an area of $\frac{1}{4}\pi d^2$ we can see that there can be at most $\lfloor \frac{(\sqrt{2}+\frac{9}{4})\pi d^2}{\frac{1}{4}\pi d^2} \rfloor = 14$ smaller circles within the circle of radius $(\sqrt{2}+\frac{1}{2})d$. This in turn implies that at most 14 Templar can check the Pylon P, concluding the proof.

F: Numbers

One obvious way in which $n \mid \texttt{rev}(n)$ may hold is $n = \texttt{rev}(n)$, i.e. that n is a palindromic number. It may be tempting to assume there are very few non-palindromic solutions, or even that 2178 is the only one, but that's not true: some of the first non-palindromic numbers of interest are $1089, 2178, 10989, 21978, 109989, 219978, 1099989, 2199978$. While there is certainly some structure to them (notice how the 9 digits start to appear in the middle?), some other "kinds" of solutions emerge for larger numbers of digits, e.g. note how 2178 being a solution implies 217821782178 is also one. Thus, let's leave this and try a more general approach.

Denote $x = \frac{\texttt{rev}(n)}{n}$, and notice that $x \in \{1, 2, \ldots, 9\}$. We will fix the value of x and the number of digits d, and generate all d-digit numbers n such that $\frac{\texttt{rev}(n)}{n}$ equals x. Note that checking d up to 14 is enough to find 10^7 solutions, as there are $\approx 10^{d/2}$ palindromic numbers with d digits.

Given x and d, we can start to try all possible options for the digits of n, starting from the least significant one. Once we fix the last digit of d, we can multiply it by x, and get the last digit of $x \cdot n$, together with the carry to the second-to-last position. Note how the last digit of $x \cdot n = \texttt{rev}(n)$ has to be the first digit of n, so we can fill that in. Continuing in this manner, we will check all ways of choosing the bottom half of digits of n, and at that point the top half will already be determined, so what is left is to check that everything is correct there. This process takes time roughly $\mathcal{O}(10^{d/2})$ for a fixed x and d, and is fast enough in practice (especially if we break early after some things go wrong, e.g. if we get that the first digit of n has to be 0).

Using the approach above we generate the first 10^7 numbers of interest, and after sorting them can answer any queries about the k-th smallest one in $\mathcal{O}(1)$ time.

G: Hostile takeover

Let u be the company with maximal daily cost c_u. We claim that there exists an optimal solution in which u is bought right after its parent company. This is true because if u were not bought right after its parent, we can swap it with a previously bought company and achieve a solution with at most the same cost (we save c_u by buying u day earlier, lose some other c_j, but $c_j \leq c_u$).

Now that we know that u is directly after its parent p, we can "join" both of the companies into one big node u' in the tree with $c_{u'} = c_p + c_u$. (Now there is a 1-day gap, but this will not change the further argument). We can now repeat this reasoning, pick the current company v which now has the maximal c_v and join it with its parent. If we repeat this process $n-1$ times, we will have constructed the optimal solution. We can simulate picking a new node with a priority queue, and join nodes together with disjoint set union in $O(n \log n)$.

H: Hero skills

Let us draw the skills as a diagram, where the number of `x` in row k denotes the number of skills of level k:

```
x
xx
xxx
xxxx
xxxx
xxxxx
xxxxxx
```

We will call a diagram *reachable* if it can be obtained following the Fate rules – our goal is to check if a given diagram is reachable. The solution bases on the following fact: *A diagram is reachable if and only if for every x-cell in it, the diagonal below this cell is full.* For example, the presence of the red cell below requires all blue part of the diagonal below to appear:

```
x
xx
xxx
xxxx
xxxx
xxxxx
xxxxxx
```

First, observe that any skill development must preserve this condition. This is because when a skill is raised, its `x` moves exactly one row up, but also at least one column to the left – if not, the target row would end up being longer. Then it either moves to the same diagonal, or to a lower one – neither can violate the diagonal condition.

It remains to prove that any such diagram can indeed be constructed. We split the diagram into diagonals and fill them one-by-one, starting with the lowest one. On every diagonal, we move the `x`'s to their places, starting from the highest ones. Every time we want to push an `x` up, we have a guarantee that the diagonal below is full, so the move is always possible.

Finally, the reachability condition is possible to check in $\mathcal{O}(n)$ – we iterate through rows and keep the number of the lowest diagonal which has had an empty space. No `x` may appear above it.