Problem A. Station Module Hierarchy

It can be shown that the depth of the station module hierarchy is $O(\sqrt{n})$. All paths of the hierarchy are no longer than twice the depth, so after any re-rooting, the depth is $O(\sqrt{n})$.

We will run a DFS that does a re-rooting along the way, and also maintain $cnt_{v,d}$: the amount of vertices d edges away from v. Then having a center v determined, we add $\sum \binom{cnt_{v,d}}{k-1}$ to the answer.

One important consideration in this problem is the amount of memory used. The size n is rather large for a typical sqrt-based problem, and that's because a lot of modules have very small subtrees. We can say that a vertex has depth d only if it has at least d children. So when we create original depth counters, the total amount of memory used is $\sum d_v \le \sum deg_v = O(n)$. This observation doesn't affect time complexity, as after re-rooting, the station module hierarchy property is destroyed, but all entries to counters are just moved entries from the original linear time.

The time complexity is $O(n\sqrt{n})$, and the space complexity is $O(n)$.

Problem B. Energy Cell Selection

We need some sort of a knapsack for the problem, but we can't do a classic one as the weights are too high. Fortunately, we can split the energy cells into two groups, where each group has small values in one of the dimensions.

Now we solve knapsacks independently for both sets, finding optimal $dp_{k,\sum a} = \sum b$. Having this dp and symmetric dp', we can perform the merging:

$$ans = \max_{i=0}^{k} \max_{w=0}^{W} \max_{w'=0}^{W} (dp_{i,w} + w') \cdot (dp'_{i,w'} + w).$$

Problem C. Championship And Buses

For each bus, you can check whether it arrives on time (that is, is not late) using the condition $s + t \le x$. Then, among such buses, you need to choose the one that departs the latest. If there is no bus that arrives on time, output -1.

Problem D. Gateway Code

Preliminaries

First, let's decompose $k = 2^a \cdot 2^b \cdot k'$, where k' is coprime with 10. If we solve the problem for k', then we can add $max(a, b)$ zeros to the end of the answer: the sum of digits doesn't change and the number becomes divisible by $10^{max(a,b)} \implies$ also by k.

The answer exists for any k', we can always find such x that $10^x \equiv 1 \pmod{k}$, so a correct gateway code n could be $\sum_{i=1}^{i=k} 10^{ix}$. The sum of digits is k and the number is divisible by k (as the sum of k terms that are $\equiv 1 \pmod{k}$).

This approach gives an answer of the length k^2 (or $\frac{k^2}{9}$), while we are required to use at most $20k$ digits.

For convenience, let's now use k instead of k'

Main idea

First idea is that we can concatenate number k to itself multiple times, getting $\overline{kk\dots kk}$. This can be written as $\sum 10^{\alpha_i} k$, so it's obviously divisible by k.

If k is divisible by sum_digits(k), the problem is solved. Sometimes it's not, and that's what we have to solve.

Original idea

We can use not only k, but also $a_i k$ for different a_i.
Once we came up with a *good* set of a_i, we can just use knapsack to get $\sum_{i_1,i_2,\ldots i_p} \text{sum\_digits}(a_i k) = k$.

A sufficient set of a_i was $[1, 3, 5, 11, 33, 55, 111, \ldots, 555555]$.
It just works, besides several small cases that can be bruteforced.
Note: we should be careful with picking a_i, for example if $k = 10\,001$ then $\forall a_i \le 9999 : a_i k = \overline{a_i a_i} \implies \text{sum\_digits}(a_i k) = 2\,\text{sum\_digits}(a_i) \implies$ knapsack is unsolvable since we want to reach an odd number $k = 10\,001$ using only even numbers.

Alternative approach 1
We can find such x that $[1, x]$ is sufficient.
We need such x that $\text{sum\_digits}(xk) \equiv k \pmod{\text{sum\_digits}(k)}$.
We can just run a search starting from $x = 1$. It will find such x quite fast since $\text{sum\_digits}(k) < 50$ and $\text{sum\_digits}(xk)$ is quite random.

Alternative approach 2
Let's generate some numbers $n_1, n_2, \ldots, n_p$ ($p \approx \sqrt{k}$) such that $\text{sum\_digits}(n_i) = \frac{k}{2}$.
We need to find n_i and n_j such that $n_i + n_j \equiv 0 \pmod{k}$.
Again, we will likely find such n_i and n_j thanks to the birthday paradox.

The problem here is that we cannot (or don't want to) generate $\sqrt{k}$ numbers of length k.
Instead, we can start with n_1, then do some modifications, one digit by one digit, keeping a persistent trace of the numbers that we have until we find required n_i and n_j.

Alternative approach 3
Let's generate a number of length $2k$ with k zeros and k ones.
Let's do random modifications to it. The results that we get modulo k are quite random, so we should find a correct gateway code in $O(k)$ steps.

Problem E. Euclid Wants to Eat

This problem can be solved in at least two ways.

- Case analysis

  Let $n = 2k + 1$, that is, odd. Then the pair $(k, k + 1)$ is a valid answer.

  Let $n = 4k$, that is, divisible by 4. Then we can output $(2k - 1, 2k + 1)$.

  Let $n = 4k + 2$, that is, when divided by 4 the number leaves remainder 2. Then we can output $(2k - 2, 2k + 2)$. One should be careful here, since 2 is a special case.

- Brute force

  Set $a = \lfloor n/2 \rfloor, b = \lfloor n/2 + 1 \rfloor$. We will run a loop while $\gcd(a, b) > 1$, after each check decreasing l by 1 and increasing r by 1. The loop will finish quickly, as proved in the method above.

Problem F. Parity Game

Let us start by analyzing when Alice definitely wins.

A simple case — when all numbers are equal to 1.

Now let us ask: when does Alice definitely lose?

For example, if all numbers are even — then the second player has a symmetric strategy. Namely, after the first player's move, they subtract 1 from every pile. Since an odd number of berries was taken from each even pile, there is still at least one berry left in each such pile, and since all piles had even sizes before the first player's move, after both the first player's move and the opponent's move all piles become even again.

Thus, Alice wins if she can reduce the game to a losing position.

Consider two cases:

- There is at least one odd pile of size at least 3

  Then Alice removes all piles of size 1, and also takes one berry from every odd-sized pile. As a result, the second player starts from a losing position in which the total number of berries is still positive.

- All odd piles have size 1

  After Alice's move, all such piles become empty, and all remaining even piles become odd. If there are no such piles, then Alice has already won; otherwise — on a sequence consisting only of odd numbers, the player who moves first wins (because either all are ones, or we are in the first case considered above).

Therefore, you should print 1 if and only if all elements of the array are equal to 1, or there exists an odd number greater than or equal to 3.

Problem G. Signal Messages

In the problem, we are asked to match the two-line signal blocks against a fixed pattern and then find the maximal consecutive segment that repeats this pattern with the same signal key S.

We can implement the parsing manually line by line, but a convenient way is to use regular expressions.

Basic grouping can be used to parse one block. The captured group is the signal key S, and backreferences ensure that all four occurrences on the first line and all four occurrences on the second line are the same word.

```
r'([a-z]+), \1 \1ity \1\n'
r'i said \1, \1 \1ity \1\n'
```

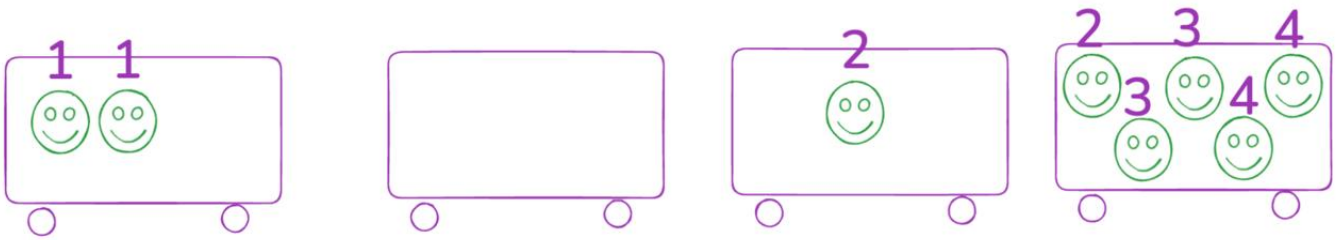
After that, we only need to keep the length of the current run of consecutive blocks with the same captured key and update the best answer.

A full regular-expression solution is also possible. It needs look-behind checks so that a match does not start in the middle of another line or overlap with another signal block:

```
r'(?<![^ \n])([a-z]+), \1 \1ity \1\n'
r'i said \1, \1 \1ity \1\n'
r'(\1, \1 \1ity \1\n'
r'i said \1, \1 \1ity \1\n)*'
```

Problem H. Capsule Passenger Distribution

Preliminaries
Let's number all passengers from left to right (arbitrarily inside one capsule).
If one passenger has a smaller number than another one, they will end up also to the left. Otherwise, we could just swap them at the moment of overtaking.
For every passenger, we know where they will end up.



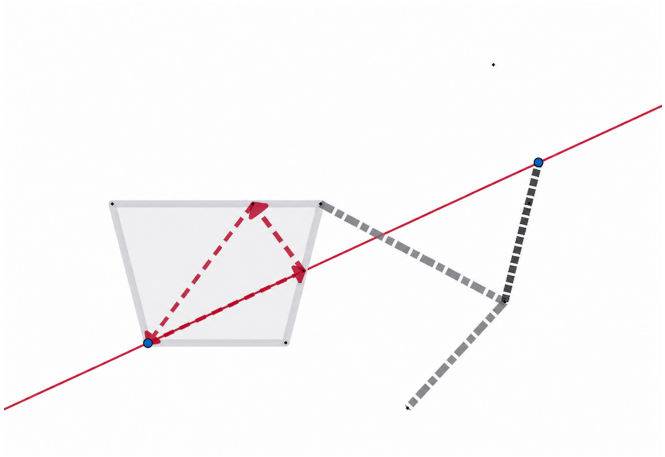
Solution

For every capsule, all passengers would end up in some segment of capsules: from l_i to r_i. We care only about the leftmost and the rightmost.

So we just compute l_i and r_i. The answer is the maximum time to reach them over all capsules.

Problem I. Optical Calibrator

The key observation is the classical unfolding/mirroring trick: instead of bending the calibration beam at each reflection, we reflect the chamber across the wall, so that in the unfolded plane the trajectory becomes a single straight segment. After reflecting across walls $2, \ldots, n-1$ (in order), the original vertex p_1 has an image q_n. A valid beam path exists iff the straight ray $q_0 q_n$ (where $q_0 = p_1$) intersects every unfolded edge $[q_i, q_{i+1}]$ strictly inside. So the solution is to unfold the polygon into a segment and verify the ray found.



Unfolding.

We'll unfold the polygon sides one by one. To do that, we think of each side as of (rotation, length) pair relative to the previous side. We just need to stack them together, additionally mirroring the rotation on every even side.

Suppose in the original polygon we know p_{i-2}, p_{i-1} and in the unfolded plane their images q_{i-2}, q_{i-1}. We want to recover the rotation, as the length doesn't change. Let

$$\text{old} = p_{i-1} - p_{i-2}, \quad \text{dir} = q_{i-1} - q_{i-2}.$$

We can recover the cos and sin of the needed angle with dot and cross products: $(\frac{old \cdot dir}{|old|^2 \cdot |dir|^2}, \frac{old \times dir}{|old|^2 \cdot |dir|^2})$. And now we can perform a new rotation using given cos and sin.

Intersection test. For each unfolded edge $(a, b) = (q_i, q_{i+1})$ and ray $(c, d) = (q_0, q_n)$, we want relative coordinate of an intersection point w_i:

$$w_i = a + t \cdot (b - a) \quad w_i = c + u \cdot (d - c)$$

We'll explain how to derive t.

$$a+t\cdot(b-a)=c+u\cdot(d-c) \quad (a-c)\times(d-c)+t\cdot((b-a)\times(d-c))=u\cdot((d-c)\times(d-c)) \quad t=\frac{(c-a)\times(d-c)}{(b-a)\times(d-c)}$$

We require $t \in [\varepsilon, 1-\varepsilon]$ with $\varepsilon = 10^{-5}$ (strictly interior) and $u \geq 0$ (forward along the ray). Failure means "NO". If all pass, the firing direction is simply $q_n - q_0$, so the answer is "YES" and the angle is $\theta = \text{atan2}(d_y, d_x)$ in degrees.

Intended complexity is $O(n)$. The low limits are due to numerical instability and the fact that it is hard to verify the model solution precisely enough. We tried to have rather forgiving tests and room for variance in computations.

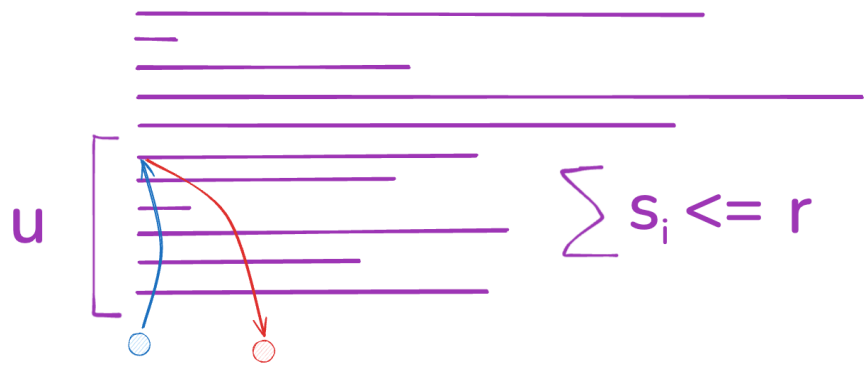
Problem J. Broken Navigation Console

Preliminaries

Let $U = u$ and $R = r$. The top $U + 1$ rows are always successful: the cursor reaches the first row during the first macro round.
For lower rows this is not immediate.
For every row, either all or none of its positions are successful. We'll prove it further.

Solution

First, let's solve the problem without updates.
Let's consider some block of exactly U rows that has the sum of lengths $\sum_i s_i \leq R$



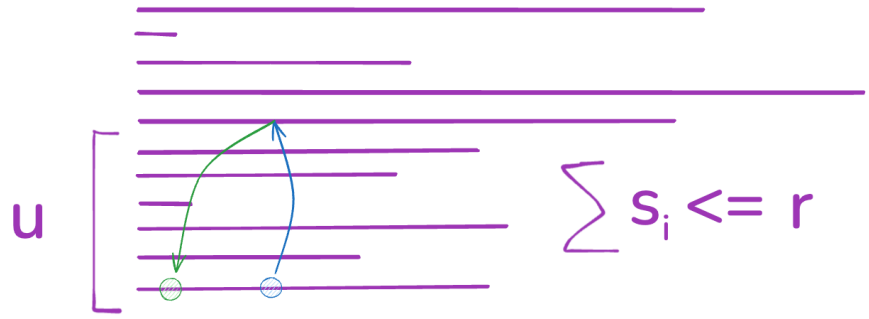
Let's consider a blue console position in any row below these U rows.
In one macro round, doing Up key presses (blue arrow), we can reach at most the first position of the top row of our U rows.
Easy to see that doing R Right key presses (right arrow) we will end up below our U rows again.
This proves that all positions below such a block of U rows are not successful.
Let's drop all rows below the very first such block (if it exists).

From now,

$$\sum_i s_i > R \tag{1}$$

(1) holds for any U consecutive rows.

Let's prove that all remaining positions are successful.
Let's pick any blue console position and prove that after every macro round we move closer to the first row (end up above or to the left).



Let's denote our blue position as (i, j). After U Up key presses we end up at $(i - U, \min(j, s_{i-U}))$.
Let's prove that we need **more than** R Right key presses to go from $(i - U, \min(j, s_{i-U}))$ to (i, j).

- To reach the end of row $i - U$ we need: $s_{i-U} - \min(j, s_{i-U})$ key presses.
- To descend $U - 1$ rows down we need: $\sum_{i=i-U+1}^{i-1} s_i$ key presses.
- To reach (i, j) we need: j key presses.

In total we need $(s_{i-U} - \min(j, s_{i-U})) + \sum_{i=i-U+1}^{i-1} s_i + j$ key presses.

$j - \min(j, s_{i-U}) \geq 0 \implies$
$\implies (s_{i-U} - \min(j, s_{i-U})) + \sum_{i=i-U+1}^{i-1} s_i + j = (j - \min(j, s_{i-U})) + \sum_{i=i-U}^{i-1} s_i \geq \sum_{i=i-U}^{i-1} s_i > R$ by (1).

So the solution is to find first such $p > 1$ that $\sum_{i=p}^{p+U-1} s_i \leq R$ and take $\sum_{i=1}^{p+U-1} s_i$ as the answer.

Adding updates
We need to solve 2 tasks:

- Find the first block of U rows that has a sum of lengths $\leq R$.
- Take the sum of some prefix.

The second task is easy enough; use any tree you like that supports position updates and prefix sums.
The first one is a bit more tricky.
Let's make a segment tree that at position i stores the sum of lengths of U rows starting from i.
And make this segment tree answer prefix minimum queries.
When we get an update, we need to update some segment of values (those that contain us in their block of U rows).
Now, to find the first such block, we can binary search the prefix that has a value $\leq R$.
The complexity is $O(n \log^2 n)$. It can also be done in $O(n \log n)$, but we didn't ask for that; binary searching passes freely.

Binary search approach

Without updates, you can binary search the answer (by rows or even purely by positions) using some simulation of the macro rounds.
If you know how to adapt this solution to the case with updates, please let us know.

Problem K. Power Consumption Optimization

It can be seen that throttling operations could be applied one by one in a greedy way. And for a single throttling operation, it is optimal to subtract from the minimal power level left in the sector.

So, basically, k throttling operations decrease some module levels to 1, leave all the large module levels untouched, and decrease one middle module level by the rest. So we need to put module levels in sorted order in a way, find the biggest prefix with $\sum(a_i - 1) \le k$, and then decrease the last module level by $k - \sum(a_i - 1)$.

To do this, you can have a persistent segment tree that stores a counter for each consumption value in some prefix of the module array. To answer a query on station sector $[\ell, r]$, one can take $t_{\ell-1}$ and t_r, then have a "virtual" tree that has the difference in prefix sums for two trees. After you go down in this tree, you find the middle module level, get the product of all bigger module levels, and update the middle module level.

Everything can be (carefully) implemented in $O((n+q)\log n)$.

Problem L. Letter Game and Wise Flea

If the total number of words is odd, then the king wins; if the total number of words is even, then the queen wins (the parity of the move number does not change).

Therefore, the problem reduces to determining the parity of the number of required city names in the interval from l to r inclusive. Note that there are no words of even length with the required property (when alternating letters over an even number of positions, one of the ends will be a vowel, which together with one of the terminal `A`'s creates an invalid situation). Therefore, we can replace even l by $l+1$, and even r by $r-1$.

First, let us check whether $l > 3$. If not, then the cases of lengths 1 and 3 are considered separately.

There is exactly one word of length 1 — `A`. So if words of length 1 are allowed, the parity changes.

Words of length 3 have the form `AXA`, where `X` is a consonant, so there are c such words; if c is even, adding such names does not change the parity, and if c is odd, it does.

Having considered the boundary cases, let us move to the main part. For $k > 1$, the number of words of length $2k+1$, where $k > 1$, is $c^k \cdot v^{k-1}$ (because the positions of vowels and consonants are fixed by alternation, starting with a consonant since it is adjacent to `A`, so the first and last internal letters must be consonants; thus we have k consonants and $k-1$ vowels. Each consonant can be chosen in c ways, each vowel in v ways, which gives the formula). Therefore, if c or v is even, then all these terms are even and do not affect the parity of the sum; if both c and v are odd, then the parity of the sum of such numbers is equal to the parity of the count of numbers of the form $2k+1$ in the interval.

Problem M. Magic of Art

Let us simplify the formula for magicness.

$$\begin{aligned}(a_2 - a_1)\cdot 1 + (a_3 - a_2)\cdot 2 + (a_4 - a_3)\cdot 3 + \ldots + (a_{i+1} - a_i)\cdot i + \ldots + (a_n - a_{n-1})\cdot (n-1)\\ = a_1(0-1) + a_2(1-2) + a_3(2-3) + \ldots + a_{n-1}((n-2)-(n-1)) + a_n(n-1)\\ = a_n \cdot n - (a_1 + a_2 + \ldots + a_n)\end{aligned}$$

Let us denote $s = a_1 + a_2 + \ldots + a_n$ — the sum of the array.

Then $a_n \cdot n - s = x$. Hence $a_n = \frac{x+s}{n}$.

Therefore, the magicness depends only on the sum and on the last element. Since the sum does not change under permutation, we only need to place an element with the required value in the last position.

Thus, if the array contains an element with this value, we place it in the last position. Otherwise, there is no solution.