Problem A. Additional Tabs

Idea: Ilya Malochka
Development: Ilya Malochka
Tutorial: Dzianis Kim

Expected difficulty: trivial
Success rate: 88.7 % (63/71)

First to solve: 0:03:28 (BSUIR 9 : Aonbel haters)
Shortest solution: 108 bytes (Belarusian NTU: Wrong Answer on Test 1)
Shortest jury solution 73 bytes

Keywords: simple math

The current tabs use $k \cdot a + (n - k) \cdot b$ megabytes, and the new ones add $\ell \cdot a$ megabytes, so print `YES` if and only if $k \cdot a + (n - k) \cdot b + \ell \cdot a \le m$.

The complexity is $\mathcal{O}(1)$.

Problem B. Block Encoding

Idea: Andrey Lukashevich
Development: Andrey Lukashevich
Tutorial: Dzianis Kim

Expected difficulty: easy
Success rate: 86.8 % (33/38 + 3 pending)

First to solve: 1:12:48 (Belarusian SU #2: static_cast<AC>)
Shortest solution: 1570 bytes (* Belarusian SU: Ra16bit)
Shortest jury solution 1367 bytes

Keywords: implementation, matrices

We should not invent a direct conversion from every format to every other format. There are five formats, and a special case for each pair is unnecessary.

Pick one normalized form, for example `ND`, or any other form you prefer. Each format is a permutation of the $n \cdot m$ sequence positions. It is enough to convert an arbitrary format to this form and back. A conversion from s to f is the inverse permutation of s followed by the permutation of f, applied to the given sequence.

Of course, the problem is solvable by putting elements into a two-dimensional matrix directly, instead of operating on flat arrays.

The complexity is $\mathcal{O}(\sum nm)$.

Problem C. Coloring of a Permutation

Idea: Andrey Lukashevich
Development: Andrey Lukashevich
Tutorial: Dzianis Kim

Expected difficulty: easy-medium
Success rate: 44.9 % (35/78 + 16 pending)

First to solve: 0:18:07 (Belarusian SU #6: nice HOOK team)
Shortest solution: 528 bytes (Belarusian NTU: Skill Issue)
Shortest jury solution 717 bytes

Keywords: greedy, LIS, dynamic programming

The conditions in the statement imply that a subsequence of the elements with the same color must be decreasing. Then, if we take the longest increasing subsequence (LIS), its elements have to be colored differently, and the length of the LIS is the lower bound for the number of colors.

Consider the standard $\mathcal{O}(n \log n)$ dynamic programming approach to calculating LIS. In it, we iterate over elements from left to right and maintain a non-decreasing array d (initially filled with $+\infty$), where d_i is the smallest ending element of an increasing subsequence of length i on the prefix considered so far. To calculate the length ans_i of the longest increasing subsequence ending at the current element p_i, we use binary search to find the rightmost d_j that is smaller than our current element. Then, $ans_i = j + 1$, and we can update the array with the current element: $d_{j+1} \leftarrow p_i$, and proceed further. After the iteration, we take the maximum of the values ans_i to get the length of the LIS.

Take ans_i as the color for p_i. If we consider any $j < i$ such that $p_j < p_i$, then we have $ans_i \geq ans_j + 1$, due to the fact that the continuation of an increasing subsequence of length ans_j ending at p_j is included in the calculation of ans_i. But that means ans is a valid coloring, since all pairs (i, j) such that $i < j$ and $p_i < p_j$ have different colors.

The total complexity is $\mathcal{O}(n \log n)$ per test case.

Problem D. Digital Display

Idea:	Ivan Lukyanov
Development:	Ivan Lukyanov, Dzianis Kim
Tutorial:	Ivan Lukyanov

Expected difficulty:	hard
Success rate:	0.0 % (0/0)

First to solve:	- (unsolved)
Shortest solution:	- (unsolved)
Shortest jury solution	1051 bytes

Keywords: combinatorics

Let $R = 10^d$. If a filled row has x in the left cell, there are $R - x + 1$ options for the right cell: empty, or any integer from x to $R - 1$. For $x = 1, 2, \ldots, R - 1$, there are exactly $R, R - 1, \ldots, 2$ options. An empty row has only one trivial option.

Let $f(r, m)$ be the number of sequences of m rows with no more than r options. Then $f(r, m) = f(r - 1, m) + r \cdot f(r, m - 1)$: either we discard r, or we append one more row with exactly r options (r choices for the right cell). The base is $f(r, 0) = 1$ and $f(0, m) = 0$ for $m > 0$.

This matches the recurrence and the base cases of the Stirling numbers of the second kind $S(m + r, r)$. (One can verify that by manual inspection of $f(r, m)$ values for small r and m.) Therefore, the answer is $f(R, n) = S(n + R, R)$.

The recurrence is unusable directly because n is too large. So we use the explicit formula derived from the inclusion–exclusion principle, counting the surjections from $n + R$ to j:

$$S(n + R, R) = \frac{1}{R!} \sum_{j=0}^{R} (-1)^{R-j} \binom{R}{j} j^{n+R}.$$

Compute it modulo $10^9 + 7$. It is enough to sum the terms $(-1)^{R-j} \cdot (j!)^{-1} \cdot ((R - j)!)^{-1} \cdot j^{n+R}$ for $j = 1, 2, \ldots, R$ (the $j = 0$ term vanishes because $n + R > 0$). Factorials and their inverses are precomputed in $\mathcal{O}(R)$, and each j^{n+R} is a binary exponentiation. Don't forget 64-bit integers.

The complexity is $\mathcal{O}(10^d \log n)$.

Problem E. Equivocal Logic

Idea: Yafim Klimasheuski
Development: Yafim Klimasheuski
Tutorial: Dzianis Kim

Expected difficulty: easy-medium
Success rate: 44.0 % (22/50 + 19 pending)

First to solve: 1:29:15 (BSUIR 2 : ëksel moksiki)
Shortest solution: 2562 bytes (Gomel SU: C++_++))
Shortest jury solution 2314 bytes

Keywords: bruteforce, bigint, precalculation

Let us compose the list of all pairs (x, k) that satisfy all the conditions.

Let us denote the sum of digits of x as s. Then, the equality we have can be rewritten as $x = (s-k)^k$.

Note that if $x = 1$, then $s = 1$, and $\sqrt[k]{x} = s - k = 1$, so the requirements cannot be satisfied. From now on, we can assume $x \geq 2$, and so its supposed k-th root $s - k$ is also at least 2 (as it cannot be equal to 1).

From the fact that the desired value of x has at most 300 digits, we can infer that $s \leq 9 \cdot 300 = 2700$. Thus, $s - k \leq 2700$.

Since $s - k \geq 2$, values of k greater than 1000 can only give us values of x greater than $2^{1000} > 10^{300}$, which is useless for us.

We can iterate over all ~ 2700 values of $s-k$ and all ~ 1000 values of k, calculate $x = (s-k)^k$, check whether the sum of digits of x is indeed s, and if so, add (x, k) to the list of solutions. That might be difficult because x can be too large, and because the number of considered pairs $(s - k, k)$ is also quite large right now.

The problem of x being too large can be solved either by using languages with built-in long arithmetics (Java, Python), or by writing it by hand. Fortunately, the latter case is not too complicated either. We can consider all cases with the same $s - k$ in the order of increasing k, and then it is enough to only multiply x by $s - k$ several times, to be able to find the sum of digits of x, and to stop when it has at least 301 digits.

The problem of a big number of considered cases is actually not a problem, as we can simply **precalculate** all the numbers locally, and then submit a program that answers the queries using the hardcoded answers. However, even without that, there are not many cases to consider, and we can submit a solution that forms the list **in runtime** well within the time limits.

Again, if we consider all cases with the same $s - k$ in the order of increasing k, we just need to stop when x grows too much, and that will take $\left\lfloor \frac{\log 10^{300}}{\log(s-k)} \right\rfloor$ iterations for a fixed value of $s - k$. If we sum this up for all possible values of $s - k$, we will get a value similar to a logarithmic integral, which is $\Theta(M \log 10^{300} / \log M)$ iterations, where M is the maximal value of $s - k$.

Denote $A = 10^{300}$, the base of the decimal system as $B = 10$, and Ans as the size of the list of answers. Since $M = \mathcal{O}(B \log A)$, we can obtain that the calculation has $\mathcal{O}(B \log^2 A)$ iterations, during each of which we perform linear operations on numbers with length up to $\mathcal{O}(\log A)$. After that, we have to sort the list of answers, which takes $\mathcal{O}(\text{Ans} \log \text{Ans} \log A)$. Therefore, the full complexity is $\mathcal{O}(B \log^3 A + \text{Ans} \log \text{Ans} \log A)$ if we do everything in runtime.

The complexity per test case is still $\mathcal{O}(1)$, as regardless of when we do the precalculation, we prefer to form the list first and then answer the queries.

Problem F. Faulty Machine

Idea: Yafim Klimasheuski
Development: Yafim Klimasheuski
Tutorial: Yafim Klimasheuski

Expected difficulty: medium-hard
Success rate: 0.0 % (0/1 + 4 pending)

First to solve: - (unsolved)
Shortest solution: - (unsolved)
Shortest jury solution 3471 bytes

Keywords: combinatorics, probabilities, linear algebra

You might be more familiar with C-notation: $C_n^k = \binom{n}{k}$.

Initially, each light bulb turns on with probability $\frac{1}{2}$. This means that for any $a \in [0, n]$, the probability of exactly a light bulbs turning on in the first second is exactly $p_a = \frac{\binom{n}{a}}{2^n}$.

We can calculate those initial p_a in the following way: store a pair of a floating-point number in the range $[0.5, 1)$ and a power of 2 as an integer. $p_0 = \frac{1}{2^n}$. $p_i = p_{i-1} \cdot \frac{\binom{n}{i}}{\binom{n}{i-1}} = p_{i-1} \cdot \frac{(i-1)!(n-i+1)!}{i!(n-i)!} = \frac{p_{i-1} \cdot (n-i+1)}{i}$. With this formula, we can iterate through i and calculate them all in linear time with reasonable absolute precision.

Now, how can we calculate the expected time to reach k from a? We can express this value as t_a. For t_k, it's obviously $t_k = 0$. For the rest, we can derive recursive formulas with variable d: the number of light bulbs among m switched that were already turned on:

$$t_a = 1 + \sum_{d=0}^{m} \left(\frac{\binom{a}{d} \cdot \binom{n-a}{m-d}}{\binom{n}{m}} \cdot t_{a-d+(m-d)} \right)$$

Note that, because $m \le 10$, the number of elements in each such sum is no more than 11.

Because $d \le m \le 10$, each of the binomial coefficients presented in this formula can be calculated via no more than 10 multiplications and divisions of integers within $2 \cdot 10^5$, so we can calculate the formula for each t_a in $\mathcal{O}(m^2)$ and maintain good relative precision of the result.

Each such coefficient in the sum where $\binom{a}{d} \cdot \binom{n-a}{m-d}$ is not zero represents a possibility of moving from a turned on light bulbs to $a - d + (m - d)$. We can build a graph with $n + 1$ nodes and edges corresponding to those entries. If node a doesn't have a path to node k, it means that exactly k light bulbs can never be reached from it, and $t_a = \infty$.

Now, how can we calculate the remaining t_a that can eventually reach k?

We can utilize Gaussian elimination. We utilize the equation described above to construct an equation

$$t_a - \sum_{d=0}^{m} \left(\frac{\binom{a}{d} \cdot \binom{n-a}{m-d}}{\binom{n}{m}} \cdot t_{a-d+(m-d)} \right) = 1$$

for each $a \neq k$ and an equation $t_k = 0$. We can place the t_a on the diagonal and use a simple pass forward and backward on the Gaussian matrix.

By having k close to $\frac{n}{2}$, we can expect that the values of non-infinite t_a will stay within a reasonable range.

However, the naive approach would require $\mathcal{O}(n^2)$ memory and $\mathcal{O}(n^3)$ time. To fix this, we can notice that each row i only contains columns in the range $[i - m, i + m]$. As a consequence of this, during the Gaussian elimination, no new coefficients outside of this range would appear in those rows, and it's enough to only use $\mathcal{O}(n \cdot m)$ memory and $\mathcal{O}(n \cdot m^2)$ time.

The final answer is:

$$\sum_{a=0}^{n} [t_a \neq \infty] \cdot p_a \cdot (t_a + 1).$$

Total solution complexity is $\mathcal{O}(n \cdot m^2)$.

Problem G. Going Along the Circle

Idea:	Andrey Lukashevich
Development:	Dzianis Kim
Tutorial:	Dzianis Kim

Expected difficulty:	very easy
Success rate:	62.9 % (56/89 + 6 pending)

First to solve:	0:10:13 (* Belarusian SU: Ra16bit)
Shortest solution:	432 bytes (Belarusian STU: BelSTU2)
Shortest jury solution	545 bytes

Keywords: inverse permutation

For all values x, find their positions b_x in the permutation by iterating over the permutation once and setting $b_{p_i} \leftarrow i$.

The length of the walk between i and $i+1$ is $b_{i+1} - b_i$ if $b_{i+1} > b_i$, and $(n - b_i) + b_{i+1}$ otherwise, as in that case we have to start a new cycle on the circle. We can also combine these two formulas into one: $(b_{i+1} - b_i + n) \bmod n$.

Solutions that spend the same number of operations as the number of steps, as well as those that find positions in quadratic time, will not pass.

The complexity is $\mathcal{O}(\sum n)$.

Problem H. Hotkeys

Idea:	Andrey Lukashevich
Development:	Dzianis Kim
Tutorial:	Dzianis Kim

Expected difficulty:	medium
Success rate:	35.0 % (7/20 + 10 pending)

First to solve:	1:11:32 (* Belarusian SU: sos repos)
Shortest solution:	1901 bytes (* Belarusian SU: Ra16bit)
Shortest jury solution	2865 bytes

Keywords: Hungarian algorithm, flows

If a menu item i is assigned a character c, the leftmost occurrence of c in s_i should be taken: any later occurrence only increases the sum and does not help other picks. Thus, the cost of the pair "menu item i, character c" is the minimum position of c in s_i, and if c does not occur in the string, the assignment is forbidden.

It remains to assign pairwise distinct characters to the menu items with the minimum total cost. This is an assignment problem: n menu items on the left, an alphabet of $|\Sigma| = 36$ characters on the right (lowercase English letters and digits). An edge between a menu item and a character has cost equal to the position of the first occurrence of that character in the name; if the character does not occur, there is no edge. Find a minimum-cost matching that covers every menu item. If none exists, the answer is `NO`.

Otherwise, the positions are recovered from the matching: for each menu item, take the first occurrence of the assigned character. If there are several optimal matchings, any of them is fine.

Since n and $|\Sigma|$ are small, the Hungarian algorithm (with complexity $\mathcal{O}(|\Sigma|^3)$) or a minimum-cost flow of value n (for example, successive shortest paths) works. The cost matrix is built with a single pass over the strings.

Problem I. Intersecting Spirals

Idea:	Yafim Klimasheuski
Development:	Dzianis Kim
Tutorial:	Dzianis Kim

Expected difficulty:	medium-hard
Success rate:	0.0 % (0/3 + 12 pending)

First to solve:	- (unsolved)
Shortest solution:	- (unsolved)
Shortest jury solution	2225 bytes

Keywords: number theory, segment tree

Let $a_i = 1/d_i$ (the sign matches the winding direction) and $x_i = Ra_i$. The numbers x_i may be negative when $d_i < 0$; process them together with the positive ones, without splitting the spirals by sign. Two spirals intersect at polar radius $r > 0$ if and only if $r(a_i - a_j)$ is an integer. The number of such points with $0 < r \le R$ is $\lfloor |x_i - x_j| \rfloor$. The origin lies on every spiral, so each pair adds one more point. The answer is

$$\sum_{1 \le i < j \le n} \lfloor |x_i - x_j| \rfloor + \frac{n(n-1)}{2}.$$

Write $x_i = k_i + f_i$ with $k_i = \lfloor x_i \rfloor$ and $0 \le f_i < 1$; for negative x_i the integer part is still taken downward. If $k_i > k_j$, then $x_i > x_j$ and $\lfloor x_i - x_j \rfloor = k_i - k_j$, decreased by one when $f_i < f_j$; if $k_i = k_j$, then $\lfloor |x_i - x_j| \rfloor = 0$. Thus the sum of $\lfloor |x_i - x_j| \rfloor$ equals $\sum_{i<j} |k_i - k_j|$ minus the number of pairs whose k-order and f-order disagree. After sorting $k_1 \le \ldots \le k_n$, the sum of absolute differences is $\sum_i k_i(2i - 1 - n)$. The correction is computed with a Fenwick tree or a segment tree on all k_i at once: order the indices by increasing f_i and, for each new k, query how many already inserted values are strictly larger. Pairs with equal fractional parts are not part of the correction. Compare fractional parts exactly, without floating point: f_i is expressed through $R \bmod |d_i|$ (for negative d_i the integer part is $\lfloor -R/|d_i| \rfloor$).

Use a 64-bit integer type.

The complexity is $\mathcal{O}(n \log n)$.

Problem J. Jiggly Sequence

Idea:	Yafim Klimasheuski
Development:	Yafim Klimasheuski
Tutorial:	Yafim Klimasheuski

Expected difficulty:	medium
Success rate:	7.1 % (1/14 + 26 pending)

First to solve:	3:45:33 (Belarusian SU #3:Team spilis)
Shortest solution:	2796 bytes (Belarusian SU #3:Team spilis)
Shortest jury solution	2276 bytes

Keywords: cases, ad-hoc

The initial sequence is $\{a_i\}$, so let's define all unique numbers as x_i, where x_i is encountered c_i times, and $c_1 \ge c_2 \ge \ldots \ge c_m$.

We want to construct a sequence with all n elements, where identical elements differ in position by at least $k+1$.

If $k+1 \geq n$, the criterion is just that all the elements need to be in different positions, so we can just check for $c_1 = 1$. From now on, let's investigate the case $k+1 < n$.

Let's divide the positions into $k+1$ groups: $A_i = \{i + (k+1) \cdot t, ...\}$. Each of those groups represents the greedy selection of possible positions for identical elements, starting from i.

Suppose that $n = a \cdot (k+1) + b$, where $0 \leq b \leq k$. In this case, groups $\forall i \in [1; b] : |A_i| = a + 1$, $\forall i \in [b+1, k+1] : |A_i| = a$.

We can investigate two cases: $b = 0$ and $b > 0$.

Suppose $b = 0$. In this case, we have exactly $k+1$ groups of size a. There are 3 sub-cases:

1) $c_1 > a$. In this case, it is impossible to construct any valid arrays.

2) $c_1 = ... = c_{k+1} = a$. In this case, we can assign any of the x_i to any of A_i to construct a valid array.

3) $c_{k+1} < a$. In this case, we have some (possibly zero) elements that can be encountered exactly a times, and at least one element that can be encountered fewer than a times.

For the elements that have $c_i = a$, we can assign them to any subsets A_i. For the rest, consider the following sequence: $\{k+1, (k+1) \cdot 2, ..., n, k, (k+1) \cdot 2 - 1, ..., n-1, ...\}$.

This sequence goes through all elements for each A_i, starting from A_{k+1} in decreasing order of i. Also, if we remove any subset of A_i from it, any continuous subsequence of length $a-1$ (including the cyclic shift) contains only indices that differ by at least $k+1$.

Therefore, we can assign the remaining x_i to unoccupied positions in this order and apply any cyclic shift.

As such, if $c_1 \leq a$, it is possible to construct a valid array with any x_i in any position.

Now, suppose $b > 0$, and let's investigate this case:

1) $c_1 > a + 1$. In this case, it is impossible to construct any valid arrays.

2) $c_{b+1} > a$. In this case, it is impossible to construct any valid arrays because we can only fit b multi-sets of equal elements of size $a+1$.

3) $c_b = a + 1$. In this case, all the multi-sets of size $a+1$ can only be assigned some permutation of $\{A_1, ..., A_b\}$. For the rest of the elements, they can be assigned to elements from $\{A_{b+1}, ..., A_{k+1}\}$ under the same logic as defined for case $b = 0$.

Therefore, in this case positions in $\{A_1, ..., A_b\}$ can be assigned any values from $\{x_1, ..., x_b\}$ and only them, while positions in $\{A_{b+1}, ..., A_{k+1}\}$ can be assigned any values in $\{x_{b+1}, ..., x_{k+1}\}$ and only them.

4) $c_b \leq a$. In this case, all the multi-sets of size $a+1$ can only be assigned any subset of $\{A_1, ..., A_b\}$.

For the rest of the elements, we can consider an ordering of indices from $\{A_b, A_{b-1}, ..., A_1, A_{k+1}, ..., A_{b+1}\}$, each taken in increasing order. If we remove any non-full subset of $\{A_1, ..., A_b\}$ from this sequence, we can notice that any remaining a consecutive elements in it (including the cyclic shifts) have pairwise distances at least $k+1$; therefore, just assigning x_i in this order and applying any cyclic shift results in a valid array.

As such, in this case positions in $\{A_1, ..., A_b\}$ can be assigned any values from $\{x_i\}$, while positions in $\{A_{b+1}, ..., A_{k+1}\}$ can be assigned any values in $\{x_{b+1}, ..., x_{k+1}\}$ and only them.

Problem K. Keep the Sum

Idea:	Andrey Lukashevich
Development:	Andrey Lukashevich
Tutorial:	Dzianis Kim

Expected difficulty:	easy
Success rate:	41.4 % (36/87 + 12 pending)
First to solve:	0:16:29 (Belarusian SU #2: static_cast<AC>)
Shortest solution:	863 bytes (* Belarusian SU: Ra16bit)
Shortest jury solution	1029 bytes

Keywords: constructive, remainders, cases

From $a_i + a_{i+1} + \ldots + a_{i+k-1} = s$ and $a_{i+1} + a_{i+2} + \ldots + a_{i+k} = s$, it follows that $a_{i+k} = a_i$. Therefore, all elements with the same index remainder modulo k must be equal. If this is not the case for some known values, there are no solutions.

Otherwise, we have reduced the problem to having an array a_1, a_2, ..., a_k, some elements of which might be missing, and the sum of all these elements has to be s.

Let s' be the value of s minus the sum of all known elements in that array. If $s' < 0$, there are no solutions. Let us assume $s' \geq 0$ from now on.

If we have no missing elements, the solution is correct if and only if $s' = 0$. If we have missing elements, we know that $s' \leq s \leq 10^9$. Thus, we can replace the first missing element with the value of s', and the other missing elements with zeroes. Of course, there are other ways to restore the missing elements.

Do not forget to restore the whole sequence at the end. The complexity is $\mathcal{O}(\sum n)$.

Problem L. Leisure Geometry

Idea:	Yafim Klimasheuski
Development:	Dzianis Kim
Tutorial:	Dzianis Kim
Expected difficulty:	medium
Success rate:	21.6 % (11/51 + 57 pending)
First to solve:	2:07:59 (* Belarusian SU: sos repos)
Shortest solution:	792 bytes (BSUIR 8 : std::out_of_range)
Shortest jury solution	346 bytes

Keywords: geometry, cases, trigonometry

If $\alpha = \beta$, every pair of segments doesn't change the direction of the pencil, and moves the pencil by a nonzero vector, so the pencil never returns to $(0, 0)$.

Now assume $\alpha \neq \beta$. Each pair of segments is the same two-step vector, rotated by $\beta - \alpha$ degrees from the previous pair. We claim that these vectors sum to zero if and only if $k(\beta - \alpha)$ is a multiple of $360°$ for some integer $k > 0$.

A return to the origin after an odd number of segments is impossible: it would mean that a unit segment starts at a distance 1 from the origin after k pairs, and writing those pairs as a geometric progression and comparing magnitudes forces $\cos(\alpha/2) = \cos(\beta/2)$, hence $\alpha = \beta$.

If we now assume that we need an even number of segments, then the first return is a closed equilateral polygon whose exterior angles alternate as $-\alpha$ and $+\beta$. A simple closed polygon has turning number ± 1, so the exterior-angle sum must be $\pm 360°$. That holds exactly when $|\alpha - \beta|$ divides 360; then there are $n = 360/|\alpha - \beta|$ pairs of segments. Otherwise, the turning number is larger than 1, and the polyline intersects itself.

It remains to find the area. You could use a pen and paper to obtain the exact formula, or calculate the needed points for one of $\dfrac{360°}{|\alpha - \beta|}$ symmetric parts of the figure and calculate the area by splitting it into triangles, or even build all the points and calculate the whole area. And if you had problems doing

that within the time limit, you could even precalculate all the answers.

The exact formula is

$$\frac{360}{|\alpha-\beta|}\cdot\frac{\cos(\alpha/2)\cos(\beta/2)}{\sin(|\alpha-\beta|/2)}.$$

The complexity is $\mathcal{O}(t)$ ($\mathcal{O}(1)$ per test case).

Problem M. Multiply and Add

Idea:	Andrey Lukashevich
Development:	Andrey Lukashevich
Tutorial:	Dzianis Kim

Expected difficulty:	trivial
Success rate:	17.3 % (47/271 + 43 pending)

First to solve:	0:08:58 (Belarusian SU #1: past the castle walls)
Shortest solution:	326 bytes (Belarusian STU: BelSTU2)
Shortest jury solution	565 bytes

Keywords: simple math, division, cases

If $m = 0$, the only candidate is $(0, 0, 0)$, so the answer is `NO` when $n > 0$, and $(0, 0, 0)$ when $n = 0$. From now on, assume $m \geq 1$.

Divide n by m and find the quotient and remainder (that is, find q and r such that $n = m \cdot q + r$, $q \geq 0$, $0 \leq r \leq m - 1$). If $q \leq m$, then a triple (m, q, r) is a suitable answer; otherwise, we can assume that $q \geq m + 1$, and $n \geq m(m + 1)$. But $m(m + 1) = m \cdot m + m$ is the greatest possible value of n that has a solution, so we can just print (m, m, m) in case $q = m + 1$, $r = 0$, and `NO` otherwise.

Note that you should not calculate the product $m \cdot (m + 1)$, as it can not fit into the 64-bit integer type when m is too large. Instead, use integer division as in the solution above. Also, you should not use floating-point number division due to precision losses.

The complexity is $\mathcal{O}(t)$ ($\mathcal{O}(1)$ per test case).