Problem A. Arithmetics and That's It

Idea:	Yafim Klimasheuski, Ivan Lukyanov
Development:	Andrey Lukashevich, Aleksey Tolstikov
Tutorial:	Dzianis Kim

Expected difficulty:	easy-medium
Success rate:	12.0 % (21/175)

First to solve:	0:54:09 (Belarusian SU 4: kitties)
Shortest solution:	1173 bytes (Belarusian SUIR #60: DIE)
Shortest jury solution	1541 bytes

Keywords: pigeonhole principle

As $2(k+1) \leq n$, we can split the array into $k+1$ continuous segments of length at least 2. If we manage to obtain an arithmetic progression by removing at most k elements, at least one segment will remain intact.

Let's iterate over segments. Suppose the current segment remains intact, then we can find the difference of the progression from the first two elements of the segment and check if the segment itself is an arithmetic progression with that difference. If yes, we can continue the sequence to the left and to the right of the current segment by picking the next (previous) number greedily.

Pick the best sequence of those obtained by the algorithm above, and don't forget to check that at most k elements are removed.

The total time complexity is $\mathcal{O}(nk)$.

Problem B. Byte Pair Encoding

Idea:	Aleksey Tolstikov
Development:	Ilya Malochka, Andrey Lukashevich, Dzianis Kim
Tutorial:	Dzianis Kim

Expected difficulty:	easy-medium
Success rate:	30.0 % (9/30)

First to solve:	2:00:03 (Belarusian SU 9: MIP solver)
Shortest solution:	2725 bytes (Belarusian SU 9: MIP solver)
Shortest jury solution	2630 bytes

Keywords: heap, set, linked list, data structures

Solutions that erase or insert elements, or count the most frequent pair of numbers in linear time, would work in at least quadratic time, which is too slow.

To erase and insert elements quickly enough once we know their location, we can store the array elements as a doubly linked list. That will also help us track which pair of numbers has the most occurrences, or, in terms of a list, the most links between neighboring elements.

Let's maintain a mapping M (using `std::map`) from a pair of numbers to the set of all its links, and to store a link, let's store the pointer to its left end. Let's also use any data structure S that allows us to obtain the maximum, insert, and delete an arbitrary element in logarithmic time (such as `std::set` or `std::priority_queue`) to maintain elements of type (number of occurrences, (l, r)) to find the most frequent pair.

How would the process look like? Initially, we form M and S for the input array by iterating over all neighboring element links. On each of at most k steps, we'll use S to obtain the pair of numbers (l, r) to replace, then take all its occurrences from M. If there are at least two occurrences, we can iterate over them, find their elements in constant time using a pointer, remove their r-elements, and replace (that is, we need to recreate its links) their l-elements with elements containing a new number. While doing that, the links between neighboring elements will change; we have to carefully consider changes and apply them to M accordingly. Then, if we change an element of M for a certain pair (l, r), we need to update the element for (l, r) also in S. We don't have to consider the case of intersecting occurrences, as $l \neq r$.

Don't forget to not put pairs of type (l, l) in both data structures as we cannot replace them. However, it would be wrong to think that once elements become equal, they will never change or will even stay in the array, because if one of them is erased together with its second neighbor, there won't be two ls anymore.

The total time complexity is $\mathcal{O}(n \log n)$, as the number of erasures is at most $n - 1$, and at all times we have to maintain data structures with size at most n whose queries have logarithmic complexity.

Problem C. Confusion

Idea:	Dzianis Kim
Development:	Dzianis Kim
Tutorial:	Dzianis Kim

Expected difficulty:	hard
Success rate:	5.7 % (5/88)

First to solve:	1:32:51 (Belarusian SU 2: Komaru, Komugi and Cocoa)
Shortest solution:	1173 bytes (Belarusian SU 3: Medium rare)
Shortest jury solution	1303 bytes

Keywords: bits, number theory, observations

Let's denote the number of bits in unsigned int type in this problem as M.

We can check that $b = 0$ is only suitable for $a = 1$.

If $b \neq 0$, then a^b and a have the same lowest bit, and so b has to be even. Then, there are two different cases.

If a is even, then for $b \geq M$, the power part becomes zero modulo 2^M, and so should the xor part, so b can be either a positive even number not exceeding $M - 2$, or equal to a. We can simply check all of those options. For most numbers, it suffices to check only $b = a$; however, doing nothing else would be a mistake, as for all even $k \leq M - 2$ there is an even a for which $b = k$ is suitable. (Fun fact: those values of a can apparently be obtained in almost the same way as the solution for odd a.)

If a is odd, we cannot obtain zero in the power part, and it seems a bit harder to solve. In a more general language, for an M-bit **odd** number a, we would like to find an M-bit **even** number b so that $a^b \equiv a \oplus b \mod 2^M$. Let's call this problem $P(a, m)$ with the value $m = M$ and abstract ourselves from a fixed value of m.

Let's try to solve the problem $P(a, m)$ for $m \geq 1$. From Euler's theorem, $a^{\varphi(2^m)} \equiv 1 \mod 2^m$, as a and 2^m are coprime. As $\varphi(2^m) = 2^{m-1}$, it means that the bit 2^{m-1} of the number b doesn't affect the power part and affects only the corresponding bit of the xor part. That means that we can first solve the problem $f(a \bmod 2^{m-1}, m - 1)$, and then adjust the bit 2^{m-1} by picking that bit in b as we like. Continuing like that, we can proceed to a problem $P(1, 1)$ with $m = 1$, and it has the only solution $b = 0$. Going back from that process, we can restore b uniquely by just trying both options for the next bit of b.

In total, we'll check $\mathcal{O}(M)$ options (possibly using different power-of-two modulos if a is odd, but that's not really a problem as we can calculate everything modulo 2^M and then take the lowest bits only). To check if a and b are suitable for each other modulo 2^m ($m \leq M$), we can use binary exponentiation in $\mathcal{O}(m)$ time, which gives us a solution with $\mathcal{O}(M^2)$ complexity per query.

We can also precalculate values of type a^{2^x}, and considering that in both cases we can calculate bigger powers of a using smaller powers of a, we can obtain a solution with $\mathcal{O}(M)$ complexity per query.

Problem D. Desired Distance

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

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

First to solve:	- (unsolved)
Shortest solution:	- (unsolved)
Shortest jury solution	5573 bytes

Keywords: segment tree, sweep line, implementation

We can notice that, with the given distance function $d(i, j)$, the distance between the two points p_i and p_j is equal to half of the side of the square with the centre at p_i that contains p_j on its' side. This observation is crucial to the further solution.

In total, after adding a new point there will be $\frac{n(n-1)}{2}$ pairs of points. It means that, to achieve the median equal to k, we need to have less than $a = \frac{n(n-1)+2}{4}$ distances that are strictly less than k, and at least a that are less than or equal to k.

First, we need to find how to calculate how many of the distances between already present $n - 1$ points are strictly less than k and how many are exactly equal to k. For that, we can make a function $f(P, t)$ that, given a multiset of points P and given distance t, calculates how many pairs of given points have the distance $\leq t$. Executing this function for k and $k - 1$, we can find the desired values.

Value of $f(P, t)$ can be calculated in the following way:

We can sort the points by x coordinate and process them in this order. Utilizing a Segment Tree with operations of addition on a segment and finding value in a point, when reaching a point with coordinates (x_i, y_i), we do the following: undo addition for all points with $x_j < x_i - t$ by adding -1 at $[y_j - t, y_j + t]$, add value at point y_i to the answer, add 1 to segment $[y_i - t, y_i + t]$ in this order. Undoing addition can be done quickly either by storing all subtractions in the scanline, or by storing all currently added points in a queue.

If $f(P, k - 1) > a$, it's impossible to add a point to achieve the desired median value. Otherwise, if $f(P, k) \geq a$, we can easily achieve the desired answer by adding a new point far away from all existing points, for example at $(10^6, 10^6)$. In the case of $f(P, k) < a$, we need to investigate further.

We can find a point on the plane that has the highest possible amount of given points from P within the distance k of it. Again, this can be done by sorting the points by coordinate x and utilizing a Segment Tree, this time with operations of adding on a segment and finding a point with the maximal value in the entire tree. With this, we can find both the point (x_0, y_0) and the number of points close to it b.

If $f(P, k) + b < a$, then achieving the desired median is impossible. Otherwise, we can place a point (x_n, y_n) at coordinates $(10^6, y_0)$ and gradually decrease its' coordinate x_n. Tracing other points with a scanline, when it for the first time has $a - f(P, k)$ given points within distance k of it, we find an answer.

Problem E. Enter the Museum

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

Expected difficulty:	easy-medium
Success rate:	29.7 % (44/148)

First to solve:	0:31:56 (Belarusian SUIR #2: Nu chto)
Shortest solution:	899 bytes (Belarusian SUIR : STICKY FINGERS)
Shortest jury solution	808 bytes

Keywords: dfs

The museum is represented by a graph, where the rooms are nodes, and the corridors are edges. Because the number of edges is 1 less than the number of nodes and the graph is connected, this is a tree. We can represent it as a rooted tree with the root located in node 1.

Because the desired path is a cycle, we will need to pass through each edge an equal number of times in both directions. We can calculate the number of times each edge needs to be passed through in the following way:

Let's look at a leaf in the tree. If the number in it is a, that means that the path will need to go from this leaf's parent to it and back exactly a times. We can subtract a from the value in the parent node and remove the leaf from the tree.

If at a certain point while there are at least 2 nodes remaining in the tree, we find a leaf with value $a \leq 0$, it means that it's impossible to construct a desired path, as we will need to pass through one of the edges ≤ 0 times. And if, when only the root remains, the value in it is not 0, it also means that it was impossible to pass through the tree.

If, when only 1 node remains, it has value 0 in it, this means that a path exists and can be constructed with depth first search (DFS). To do this, you can start in node 1, and, as long as one of the values in a child node is greater than 1, go to the child node i and subtract 1 from a_i. And, if all the values in the child nodes are equal to 0, go back to the parent node p and subtract 1 from a_p.

An important note: if you go through all neighbors every time when choosing the next node in the path, the solution will take too long.

Problem F. Fairly Easy Problem

Idea:	Dzianis Kim
Development:	Dzianis Kim
Tutorial:	Dzianis Kim

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

First to solve:	- (unsolved)
Shortest solution:	- (unsolved)
Shortest jury solution	2759 bytes

Keywords: geometry, derivatives, optimization

Without loss of generality, we can assume that $D = O(0, 0)$ as otherwise we can subtract it from all the points.

For the purpose of this tutorial, let's denote as "β-periodic angle α" a set of angles that differ from α by a multiple of β, with the meaning that all angles in that set are equivalent in the meaning they are used.

For each non-zero point T, let's take a directed angle α between the ray Ox and the ray OT, and call 2π-periodic angle α the angle of this point. For each line passing through O, let's take a directed angle between the ray Ox and any of the rays lying on the line and starting at O, and call π-periodic angle α the angle of this line.

Let the periodic angles corresponding to points C_i and lines l_j be denoted as (2π-periodic) α_i and (π-periodic) β_j, respectively. Let d_i be the distance between O and C_i.

Each circle has exactly two tangents passing through O (unless O lies on the circle, then they coincide). If the angle corresponding to one of the tangents to ω_i is π-periodic x, then the angle corresponding to the second tangent is π-periodic $(2\alpha_i - x)$, since the ray OC_i bisects the angle formed by the tangents.

Let l_0 be the other tangent to ω_1 than the common tangent of ω_1 and ω_2, and let l_n be the other tangent to ω_n than the common tangent of ω_{n-1} and ω_n (unless ω_1/ω_n respectively pass through O, then the two tangents are the same). Then if $\beta_0 = x$, the following conditions hold for π-periodic angles:

$$\begin{cases} \beta_0 = x; \\ \beta_1 = 2\alpha_1 - x; \\ \beta_2 = 2\alpha_2 - 2\alpha_1 + x; \\ \ldots \\ \beta_n = 2\alpha_n - 2\alpha_{n-1} + \ldots + (-1)^{n-1}2\alpha_1 + (-1)^n x; \end{cases}$$

The radius r_i of circle ω_i can be found as $|d_i \sin(\alpha_i - \beta_{i-1})|$ (yes, we cannot really subtract 2π- and π-periodic angles, but in this expression, the value never changes regardless of the value we pick from the set of angles). We need to maximize the sum of the squares of the radii, that is, $\sum\limits_{i=1}^{n} d_i^2 \sin^2(\alpha_i - \beta_{i-1})$, where the values of β also depend on x.

Finding the maximum may seem complicated, but in fact, each of the functions under the sum has the form $d_i^2 \sin^2(A \pm x)$, where A is some number. Without loss of generality, we can assume that the sign in front of x is a plus (otherwise, we can just change the sign of the sine argument entirely). The derivative of such a function with respect to x has the form $2d_i^2 \sin(A+x)\cos(A+x) = d_i^2 \sin(2A+2x)$. This expression is exactly the y-coordinate of the point $(d_i^2, 0)$ if we rotate it around the point $(0, 0)$ by angle $2A$ (then it becomes $(d_i^2 \cos 2A, d_i^2 \sin 2A)$), and then also by angle $2x$.

To find such a value x that the sum of these derivatives equals 0, we now need to find such an angle $2x$ that the rotation of the sum of points $(d_i^2 \cos 2A, d_i^2 \sin 2A)$ by this angle around $(0, 0)$ produces a point with a zero y-coordinate and a negative x-coordinate (this is important, as we want to find a local maximum, so the derivative should decrease). This can be done by summing those pre-rotated points and then using `atan2` function. The resulting angle $2x$ will be 2π-periodic, therefore, x will be π-periodic, as required for the angle corresponding to the line.

Once we found the desired value of x, it is fairly easy to find all the angles, radii, and the answer to the problem.

From the above solution, we can also conclude that the answer can be found using ternary search over the angle of l_0. But even though finding an extremum for a trigonometrical function like this requires fewer iterations (as its derivative is around zero at that point), "usual" ternary search is not intended to pass. The input data was given in a compressed form, so that the time spent by the solution depends less on input data size, and the logarithmic factor for the ternary search becomes too big to ignore. You could probably bypass the time limit by using golden ratio $(\sqrt{5} - 1)/2$ to determine the next points (which allows you to calculate only one value per iteration and not two), and at the same time reaching the lower bound of the number of iterations that doesn't get precision-related Wrong Answer.

Problem G. Gorgeous Summation

Idea:	Yahor Dubovik
Development:	Yahor Dubovik
Tutorial:	Dzianis Kim

Expected difficulty:	hard
Success rate:	9.7 % (3/31)

First to solve:	2:25:45 (Belarusian SU 1: sos repos)
Shortest solution:	1248 bytes (Belarusian SU 3: Medium rare)
Shortest jury solution	1335 bytes

Keywords: palindromes, Manacher's algorithm, inclusion-exclusion

Let $A = 100$ be the upper bound for the values of array elements.

To find the value for a segment of even length, the problem asked you to take all pairs of elements at the same distance from the center of the segment, sum up numbers in those pairs, and take the greatest common divisor (GCD) of the resulting values. The chosen pairs of elements correspond exactly to those elements that have to be equal if the whole segment was a palindrome. Let's try to find a solution that uses a similar idea.

For now, let's consider a segment $[L, R]$ of the array. If its value is divisible by d, for all $i = 0, 1, \ldots, \lfloor (R-L)/2 \rfloor$, $a_{L+i} + a_{R-i}$ is also divisible by d. As the segment length is an even number, if we multiply a_i by $(-1)^i$ for all i, then all those conditions will change to "a_{L+i} and a_{R-i} have the same remainder modulo d", which means that the whole segment is a palindrome when all the elements of the modified array are taken modulo d.

Let's multiply a_i by $(-1)^i$ for all i. Now, to determine the number $F(d)$ of segments with even length whose value is divisible by d, we can take the remainders of all array elements modulo d, then find all the even palindromes using Manacher's algorithm in linear time. There are only 200 possible values of d, as this is the upper bound for a sum of two numbers from the input.

To determine how many segments have value equal to a certain number d (and not just divisible by d as above), we can use the inclusion-exclusion principle, which is easier to summarize using the Möbius function μ in this case. Let the desired value be denoted as $G(d)$, then $G(d) = \sum_{d|m} \mu(m/d) F(m)$.

The answer is equal to $\sum_d^{2A} G(d) \cdot d = \sum_d^{2A} \sum_p^{\lfloor 2A/d \rfloor} d \cdot \mu(p) \cdot F(pd)$. As for this double sum we'll get $2A + \frac{2A}{2} + \frac{2A}{3} + \ldots + \frac{2A}{2A}$ iterations in the inner for loop, we can calculate it in $\mathcal{O}(A \log A)$ time.

The total time complexity of this solution is $\mathcal{O}(nA + A \log A)$.

A few other solutions based on the same idea with palindromes were intended to likely pass too (as long as the complexity is subquadratic with respect to n). Some of the variations:

- Instead of applying $a_i \leftarrow a_i \cdot (-1)^i$, we can replace the array with the array of its prefix sums, and do everything in the same fashion otherwise (apart from the fact we need to look for odd palindromes in this case).
- Let's find the palindromes using hashes and binary search around the target center of the palindrome. That would slow down the corresponding part of the solution by a factor of $\mathcal{O}(\log n)$; however, we don't have to know Manacher's algorithm for that.
- Let's not use the inclusion-exclusion principle with $2A$ numbers. Instead, let's only consider the primes and their degrees that do not exceed $2A$ (for $A = 100$, there are only 60 of them). For every potential center of a palindrome and for every considered number x, let's find the greatest palindrome with that center for an array of remainders modulo x, and obtain events of type "at width w, it stopped to be a palindrome modulo x". Then, if we start expanding the segment from that center, we can process all of these events one by one, and for all segments with this center we can calculate the sum of their values.

Problem H. Huh? Oh, Yes, Welcome to the Contest!

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

Expected difficulty:	trivial
Success rate:	96.3 % (78/81)

First to solve:	0:06:30 (Belarusian SUIR #2: Nu chto)
Shortest solution:	290 bytes (Minsk br of Plekhanov RUE: SKaM)
Shortest jury solution	290 bytes

Keywords: implementation, joke problem

This problem required you to:

- copy or retype the dialogue using either HTML or PDF problem statement;
- read all characters of the first line of the input;
- print the input inside the dialogue in the place of the team name twice;
- replace all characters `a`–`z` in it with `A`–`Z` (for example, by subtracting 32 from their ASCII codes, or using any conversion function in your programming language), and printing the result inside the dialogue one more time;
- be careful to not print extra characters, not convert extra characters, and perform the other steps correctly.

Problem I. Imperial Decree

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

Expected difficulty:	medium-hard
Success rate:	100.0 % (3/3)

First to solve:	3:38:30 (Belarusian SU 2: Komaru, Komugi and Cocoa)
Shortest solution:	1966 bytes (Belarusian SU 1: sos repos)
Shortest jury solution	2379 bytes

Keywords: dynamic programming, probabilities

Note that if the squares of the points are fixed, the distance between them in the x direction and the distance in the y direction become independent from each other. This allows us to assert that the expected value of the product is equal to the product of the expected values.

Next, we will find the expected value of the distance in the x direction depending on the coordinates of the squares. Let them be in rows x_1, x_2. If $x_1 \neq x_2$, then $E = |x_1 - x_2|$; otherwise, $E = \int_{a=0}^{1} \int_{b=0}^{1} |a-b| db da = \frac{1}{3}$. The same can be applied for the distance in the other direction.

For quick calculation of all distances, it is possible to find the sum of all weighted distances horizontally and vertically for each pair of squares using dynamic programming:

Going through columns left-to-right, for each square we can calculate the sum of weighted distances to the squares on the left (it can easily be maintained by storing sums of prefix weights for each row), and we can find the same for vertical distances.

By multiplying sums of weighted vertical and horizontal distances in a square, we consider the case where a corner of the rectangle that is not one of the two selected points belongs in it. To consider the case where the selected points fall in the same row or column, we can use one of the calculated weighted distance sums and consider another distance to be equal to $\frac{1}{3}$.

In the end, the solution works in $O(nm)$.

Problem J. Jolly Polygon

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

Expected difficulty:	medium
Success rate:	28.9 % (22/76)

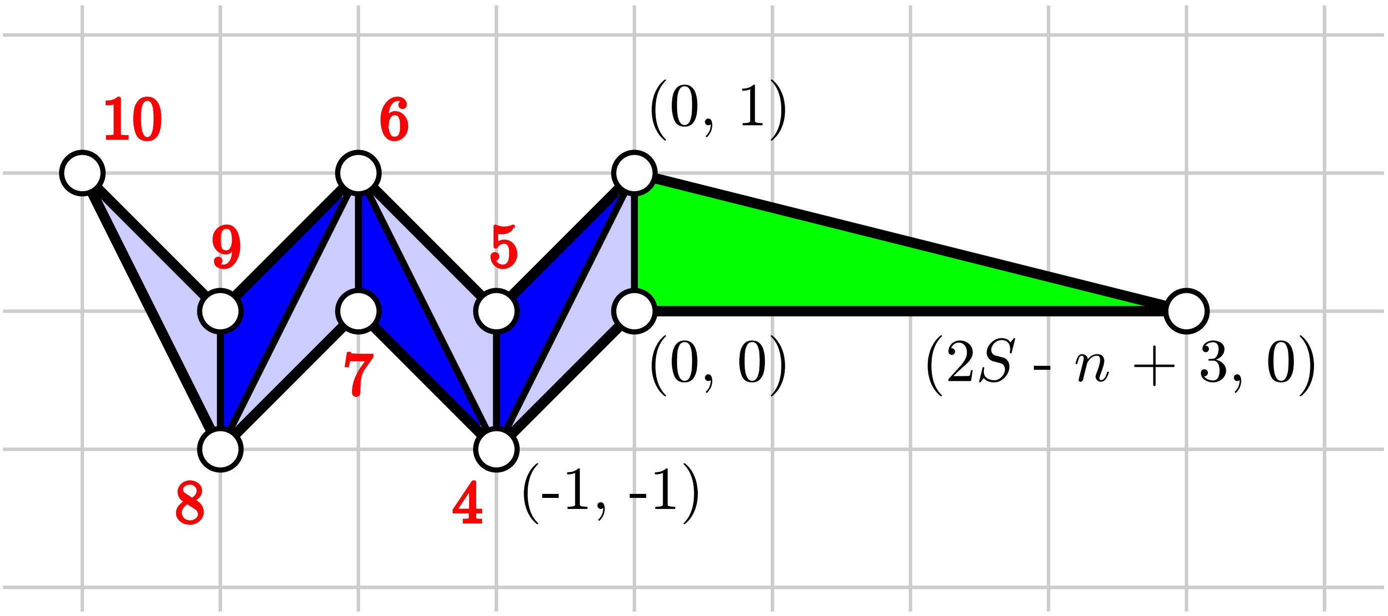
First to solve:	1:26:25 (Belarusian SU 2: Komaru, Komugi and Cocoa)
Shortest solution:	721 bytes (Belarusian SU 20: Khan Zamay)
Shortest jury solution	468 bytes

Keywords: geometry, construction, Pick's formula

As this is a constructive problem, it's likely that there are many different solutions. Here is one of them.

It is widely known that for a non-degenerate triangle whose vertices have integer coordinates, the smallest possible area is $\frac{1}{2}$. Let's build a chain of $n-2$ triangles, so that they don't share non-boundary points, all of them except one have the area of $\frac{1}{2}$, and the remaining triangle allows us to set the total area to S. Also, we have to make sure no three consecutive points are collinear.

The configuration depicted below (and moved along some fixed vector, so that all the coordinates are between 0 and 10^6) satisfies all of those conditions. The explanation of this configuration follows.



Instead of numbering points in the order of polygon traversal, let's start with three points which form a green triangle with area $S - \frac{n-3}{2}$, which is exactly the desired area minus $\frac{1}{2}$ per every point that is not one of those three. Then, let's add points one by one, and change polygon for the new points, and number the points accordingly. Thus, the 6th point on the picture is the one we added to obtain a polygon for $n = 6$, not the sixth point in any order.

Let's add points as follows: for each $k \geq 2$, the $(2k)$-th added point has coordinates $(-k, (-1)^{k+1})$, and the $(2k+1)$-th added point has coordinates $(-k, 0)$. It means that every time we add a new point, we form a new triangle with area $\frac{1}{2}$. On the picture, the bold red numbers near the points indicate the order of addition, and the newly added triangles are filled with different shades of blue.

It's pretty clear that neither the upper part nor the lower part of the polygon's perimeter contain three consecutive collinear points, and that the total width of the picture is at most 10^6. Without careful thought, it's easy to run into one of these two problems.

Problem K. Know Your Duration of Stay

Idea:	Dzianis Kim
Development:	Dzianis Kim
Tutorial:	Dzianis Kim

Expected difficulty:	very easy
Success rate:	41.6 % (62/149)

First to solve:	0:27:31 (Belarusian SU 4: kitties)
Shortest solution:	516 bytes (Minsk br of Plekhanov RUE: SKaM)
Shortest jury solution	547 bytes

Keywords: prefix sums

You can construct in linear time the array $P = (p_0 = 0, p_1, p_2, \ldots, p_n, p_{n+1}, p_{n+2}, \ldots, p_{2n})$ of prefix sums of month durations for two consecutive years, that is, $p_i = p_{i-1} + a_i$, $p_{i+n} = p_{i+n-1} + a_i$ for $i = 1, 2, \ldots, n$.

Then, if you are given a query $(s_d, s_m), (e_d, e_m)$, if the departure day is strictly earlier in the calendar than the arrival day, increase its month by n. Now you can calculate the answer in constant time per query as $P[e_m - 1] - P[s_m - 1] + e_d - s_d$.

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

Problem L. Late Autumn Set of Cards

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

Expected difficulty:	very easy
Success rate:	29.1 % (62/213)

First to solve:	0:12:17 (Belarusian SU 7: loverov lovers)
Shortest solution:	646 bytes (Minsk br of Plekhanov RUE: SKaM)
Shortest jury solution	1189 bytes

Keywords: cases, bitmask dynamic programming, divisors, number theory

The sample tests tell us that $16112024 = 2^3 \cdot 269 \cdot 7487$, and the numbers 269 and 7487 are prime, so the desired number has exactly 16 positive integer divisors. Only cards whose numbers are divisors of 16112024 could be useful for us, and all the other cards can be removed. Also note that the numbers in the input are limited to 10^4, so not all divisors of 16112024 could even appear there.

Then, we can either consider all cases manually, or use dynamic programming.

A manual consideration of cases could be as follows: the answer is 0 if there's no number 7487 (as no other number in the input could have 7487 as a divisor). Otherwise, we can check for numbers 269, $269 \cdot 2$, $269 \cdot 2^2$, $269 \cdot 2^3$, and if we have enough numbers 2, 4, 8 to get a desired product with that number, output the solution; otherwise, there's no solution.

A typical dynamic programming solution involves having states of type (a, b, c), $0 \le a \le 3$, $0 \le b, c \le 1$, which correspond to numbers $2^a \cdot 269^b \cdot 7487^c$. For each such number x, we can store, for example, -1 if it's impossible to obtain it yet, or any number in the product that obtains x (or 1 for the number 1, as it corresponds to an empty product).

Initially, $dp[0][0][0] = 1$, and all the other values are equal to -1. We process all the divisors of 16112024 from the input one by one. If the current divisor is $2^a \cdot 269^b \cdot 7487^c$, and currently $dp[x][y][z] \neq -1$ and $dp[x + a][y + b][z + c] = -1$, then we can assign the latter value to this divisor, provided that $x + a \le 3$, $y + b \le 1$, $z + c \le 1$. The states should be processed in descending order to avoid using newly changed values.

At the end, if the answer exists, we can restore the product itself by starting in $dp[3][1][1]$, and then repeatedly taking the divisor in the current state and going to another state corresponding to the current number divided by that divisor.

This solution is a more general one and wasn't required to solve the problem.