

Problem A. Eye Color

Idea: Nikolay Dubchuk
Development: Nikolay Dubchuk

The jury suggests solving the problem as follows. First, we will write a function `isSame` that takes two strings s_1 and s_2 as input and checks whether the input string s_1 can represent the color written in string s_2 . In C++, such a function can be implemented as follows:

```
bool isSame(string s_1, string s_2) {  
    return s_1 == "." || s_1 == s_2;  
}
```

Now we will write a function that determines the eye color of a child based on the eye colors of the parents. It turns out that this color can be determined unambiguously according to the rules written in the statement. The implementation in C++ is:

```
string getChildColor(string s_1, string s_2) {  
    if (s_1 == "brown" || s_2 == "brown")  
        return "brown";  
    if (s_1 == "green" || s_2 == "green")  
        return "green";  
    return "blue";  
}
```

Using these functions, it is already easy to solve the problem. We will count all three eye colors. With three nested loops, we will iterate through the eye colors: first for the first parent, then for the second, and finally for the child. We will iterate in lexicographical order, that is, in the order [blue, brown, green]. Inside the innermost loop, we will check with three calls to the function `isSame` that the eye colors correspond to the read strings. Finally, we will check that the child's eye color matches the color returned by the function `getChildColor`, and only if all these checks pass, we will output the three eye colors.

Problem B. Big World Politics

Idea: Roman Pervutinskiy
Development: Roman Pervutinskiy

Consider the countries that have territorial claims from country i . They either lie entirely within the minimum rectangle of country i , intersect its border, or do not intersect with it at all. We need to count the total number of the first and second types of countries.

Note that the perimeter of the minimum rectangle of country i is no greater than four times the number of cells belonging to country i , since one can traverse from the left to the right border and from the top to the bottom border of the minimum rectangle using the cells of country i . Therefore, the sum of the perimeters of all minimum rectangles is $O(nm)$.

Let's understand how to account for the countries that lie entirely within the minimum rectangle. We will choose some representative cell for each country and count the number of representatives in all prefix rectangles. Thus, we can find the number of countries whose representatives lie within the minimum rectangle.

Finally, we need to account for the countries that intersect the border of the minimum rectangle but whose representatives lie outside of it. For this, we can traverse the cells along the border and count all such colors. This gives us a solution in $O(nm)$.

Problem C. Puzzle

Idea: Roman Gusariev
Development: Ekaterina Vedernikova

The allowed operations permit independently rearranging elements within each column, while the number

of zeros and ones in the column remains unchanged.

This means that in each column, we can choose which rows will have the value 1 and which will have the value 0, but the total number of ones is fixed.

If there are k ones in column j , then the maximum number of identical rows in that column is $\max(k, n - k)$ — we can either make all those rows have a one or a zero.

For the rows to be completely identical, they must match in all columns simultaneously. Therefore, the total number of identical rows cannot exceed the minimum value among all columns.

It is sufficient to count the number of ones in each column and take the minimum of the values $\max(k, n - k)$. This will be the answer.

The algorithm runs in $O(nm)$ and fits within the constraints.

Problem D. New Balls

Idea: Ivan Safonov

Development: Egor Ulin

To solve this problem, it is necessary to construct n trees that do not have overlapping edges.

We will show that if any two trees share at least one common edge, then there will exist a tree that can be obtained from both trees in $n - 1$ operations or less.

We will demonstrate the sequence of steps for the first tree:

1. During the reconstruction of the tree, we will choose such (c, d) that there is no such edge in the current tree, but there is such an edge in the second tree. Then we will add the edge (c, d) to the current tree. A cycle will appear in our graph; we note that on such a cycle, there will be at least one edge that is not in the second tree, so we will take this edge as (a, b) ;
2. We will repeat this until we have n common edges between the trees;

After that, the difference will be no more than $n - 1$ edges between the second tree and the tree from the query, and then we can transform the second tree into the one just obtained using the same operations, which means we have found a tree that can be obtained from two different trees, which does not allow us to uniquely restore from which tree it was obtained.

Note that the complete graph on $2n$ vertices will contain $\frac{2n \cdot (2n - 1)}{2}$ edges, while a tree on $2n$ vertices contains $2n - 1$ edges, so the maximum number of trees we can obtain is $\frac{2n \cdot (2n - 1)}{2 \cdot (2n - 1)} = n$. That is, we must distribute all the edges of the complete graph among the trees.

We will solve a slightly different problem; suppose we want to split the complete graph into non-overlapping Hamiltonian paths, this can be done as follows:

1. Construct the path $1, 2, 2n, 3, 2n - 1, \dots, n + 1$;
2. All other paths will be obtained from the first one by shifting all vertices modulo $2n$ by values from 1 to $n - 1$. For example, by shifting by 2, we get the path $2, 3, 1, 4, 2n, \dots, n + 2$.

It is not difficult to notice that such paths will not have overlapping edges, and their union will yield the complete graph. And since we were able to split the graph into paths, we were able to split the graph into trees, which will be bamboo.

After constructing such trees, to answer the queries, it is sufficient to find a tree that has at least n common edges with the tree from the query.

Problem E. Array Depletion

Idea: Pavel Skobelin
Development: Pavel Skobelin

We will use a greedy algorithm to solve the problem, which will consist of the following: in each turn, we will choose the leftmost index i such that $a_i + a_{i+1} = x$, and remove this pair of elements. It is claimed that Ksyusha can complete the level if and only if she can complete the level using this greedy strategy.

First, let's analyze the simpler part — implementing the greedy strategy. This can be done by processing the array from left to right: we will maintain a stack of elements that have not yet been removed. If the current number in the array, when added to the last element of the stack, equals x , we will remove the last number from the stack; otherwise, we will add the current number to the stack. It is not difficult to notice that this algorithm implements the greedy algorithm described in the solution.

It remains to show that this greedy algorithm is correct. It is obvious that if the stack is empty at the end, then the array can be emptied—indeed, we have reached an empty array by removing adjacent (at the moment) elements. Now, let's show the implication in the other direction. Suppose index j was chosen in the solution, while the greedy algorithm chose index i such that $i < j$. We will consider two cases:

1. $j = i + 1$. Then $a_i + a_{i+1} = x$, and $a_{i+1} + a_{i+2} = x$. From this, it follows that $a_i = a_{i+2}$ — which means that the array will be the same when removing the i -th and j -th elements, so the order of operations can be changed.
2. $j > i + 1$. In this case, it is not difficult to show that the order of these operations can be changed without worsening the solution.

Thus, it was sufficient to implement the greedy algorithm described above to solve the problem.

Problem F. Fibonacci Chocolates

Idea: Ildar Gainullin
Development: Ildar Gainullin

Consider the states of the chocolates independently of each other. Each chocolate at any moment in time is described by a pair of sets

$$(L, R),$$

where L is the set of possible states that Alice can transition to, and R is the set of states that Bob can transition to.

Any state of the game is similarly described by a pair of such sets. Thus, each position in the game has the form

$$G = \{L \mid R\}.$$

It is claimed that for each chocolate, there exists a rational number equivalent to the game with that chocolate in the following sense: if the sum of the rational numbers corresponding to all chocolates is strictly greater than zero, then Alice wins; otherwise, Bob wins.

Consider the rule for summing two such games. Let

$$G_1 = \{L_1 \mid R_1\}, \quad G_2 = \{L_2 \mid R_2\}.$$

Then their sum is defined as

$$G_1 + G_2 = \{L_1 + G_2, G_1 + L_2 \mid R_1 + G_2, G_1 + R_2\}.$$

To compare two games x and y , it is sufficient to check whether the inequality $x \leq y$ holds. It is claimed that $x \leq y$ if there does not exist a move by Alice $x^L \in L_x$ such that $y \leq x^L$, and there does not exist a move by Bob $y^R \in R_y$ such that $y^R \leq x$.

Note that in analyzing the game, we actually only need to know the largest element of the set L and the smallest element of the set R . Moreover, to find the numerical value of a certain game x , whose value is an integer, it is sufficient to be able to check comparisons of the form $x \leq n$ and $x \geq n$, where

$$n = 1 + 1 + 1 + \dots + 1$$

— the sum of n unit games.

If the value of the game is not an integer, it can be proven that it is always a rational number with a denominator that is a power of two. Therefore, the value can be successively doubled (by adding the game to itself) until it becomes equal to some natural number n .

Comparisons and the calculation of the sum can be implemented using memoization independently for each chocolate.

The operation of removing all parts of one chocolate can be described as transitioning from the current state s of the game for one chocolate to the state 0 (that is, $0 \in L$ if the chocolate belongs to Alice, and $0 \in R$ otherwise).

In fact, this game can be described as a red-blue Hackenbush — a Hackenbush game in which both players can remove any edge from the tree on their turn, leaving only the connected component containing the root — with the distinction that the edges are colored blue and red, where Alice can only remove blue edges and Bob can only remove red edges. For each chocolate, exactly one edge of the color belonging to the player is connected to the root.

After calculating the numbers corresponding to each chocolate (which can be pre-computed locally), one can notice that the number for each chocolate under the given constraints has the form

$$\pm \frac{a}{2^b},$$

where a, b are small natural numbers (not greater than 50).

To count the number of subsets with a positive sum among numbers of this form, one can use bitwise dynamic programming on the binary digits.

Problem G. Magic Ritual

Idea: Nikolay Vedernikov

Development: Nikolay Vedernikov

To solve the problem, we need to minimize the cost of rearranging the array into non-decreasing order.

Swapping elements at positions with a difference of 2 has zero cost. This means that elements at even positions can be rearranged among themselves for free, and the same is true for odd positions. Any swap that changes the parity of an element's position costs at least 1.

Let's sort the array. We will see which values should be at even and odd positions in the sorted array.

We will divide the elements of the original array into two multisets: elements from even positions and elements from odd positions. Then we will go through the sorted array and take the required elements from the corresponding multiset. If an element is found, we remove it.

After this, there will be elements left in the multisets that are not in their correct parity. Each paid swap corrects two such errors, so the minimum number of paid operations is half the number of remaining elements.

This is the answer for this test case.

Sorting and working with multisets give a complexity of $O(n \log n)$ per test, which fits within the constraints.

Problem H. Beaver Dam

Idea: Pavel Skobelin

Development: Pavel Skobelin

We will perceive the maze as a tree. From the fact that it is laid out on a table, we will use only the fact that each vertex has at most 4 neighbors.

To solve the problem, we will prove a supporting lemma. Consider any edge $u - v$. We will then send beavers to vertices u and v . For the sake of generality, let the beaver from vertex u reach the exit no later than the beaver from vertex v . We will divide the graph into two sets: U — the subtree of vertex u , including itself, and V — all other vertices. In this case, two things are asserted:

1. There is at least one exit in the set of vertices U .
2. If a beaver is sent to any vertex in set U , then the nearest exit for it will be located in set U .

The first assertion is clear: if all exits in the current tree are located in set V , then the beaver sent from vertex v would have reached it strictly earlier, since the shortest path from u to any vertex in set V goes through vertex v , which means it has a strictly greater length.

Now, let's prove the second assertion. Let the shortest path from vertex v to the exit have length x . Thus, the shortest path from vertex v to the exit in set V has length $y \geq x$. Therefore, the shortest path from vertex v to the exit that lies in set U has length $z \geq y + 1 = x + 1$.

Thus, the strictly nearest exit from vertex u lies in set U . Now, suppose the optimal exit for some vertex $w \in U$ is located in set V . In this case, it passes through vertex u . This means that this path could have been strictly shortened by going from vertex u to a vertex in set U . It follows that we have reduced the optimal answer, which leads to a contradiction — hence, this could not be the case, and the optimal exit for w lies in set U . The second point of the lemma is proven.

Let's come up with a solution for the original problem. It turns out that if we choose some edge $u - v$, after the query, using the first point of the lemma, we reduce the considered set to one of the sets U or V . Note that if we have reduced the considered set to U — then in it, using the second point of the lemma, we can solve the problem recursively, that is, choose an edge $u' - v'$, and carry out similar reasoning for them, since in this set there will be at least one exit according to the first point of the lemma.

The size of the considered set decreases each time. When it becomes 1, the only vertex in this set will contain the exit. It remains to understand how to choose the edge $u - v$ each time.

It is clear that we want to find such an edge $u - v$ that the size of the smaller of the sets U and V will be maximized. Let's find the centroid of the tree. It will satisfy the condition that all its children have size at most $s/2$, where s is the current size of the tree. It can also be understood that there will be at least one child whose size is at least $s/4$, since each vertex of the tree has at most four neighbors. Thus, if we consider the edge from the largest child to the centroid as the edge $u - v$, the size of the smaller subtree will be from $s/4$ to $s/2$, that is, at least $s/4$. Therefore, by choosing such an edge, we will discard at least $1/4$ of the size of the tree.

It is clear that with such actions we will make at most $\log_{4/3}(nm) \leq \log_{4/3}(nm) = 48$ actions until we obtain a set of size 1. In fact, such a solution easily fits within the query limit, as this is a rather rough estimate.

Problem I. Predicting a Position

Idea: Grigoriy Khlytin
Development: Grigoriy Khlytin

Let the initial sorted array of keys in ascending order be denoted as $X = x_1, x_2, \dots, x_n$. Let the maximum allowable prediction error for the position be denoted as \mathcal{E} .

The formal problem is to partition the array of keys into the minimum number of parts such that for each part $[l, r]$, for the set $\{(x_i, y_i)\} = \{(x_l, 0) \dots (x_r, r - l)\}$, there exists a line $y = k \cdot x + b$ that satisfies the error inequalities for all $i \in [l, r]$:

$$\begin{cases} k \cdot x_i + b \leq y_i + \mathcal{E} \\ k \cdot x_i + b \geq y_i - \mathcal{E} \end{cases}$$

We can transform the system as follows:

$$\begin{cases} b \leq (-x_i) \cdot k + (y_i + \mathcal{E}) \\ b \geq (-x_i) \cdot k + (y_i - \mathcal{E}) \end{cases}$$

Consider each inequality of the system separately, temporarily disregarding \mathcal{E} :

$$b \leq (-x_i) \cdot k + (y_i) \tag{T1}$$

$$b \geq (-x_i) \cdot k + (y_i) \tag{T2}$$

Inequality (T1) in the coordinates (k, b) describes a certain set of points Q , which is bounded from above by the *lower envelope* of the lines, while inequality (T2) describes a set of points P , which is bounded from below by the *upper envelope* of the lines defined by the equations $b = (-x_i) \cdot k + (y_i)$, with slopes $-x_i$ and intercepts y_i for all $i \in [l, r]$.

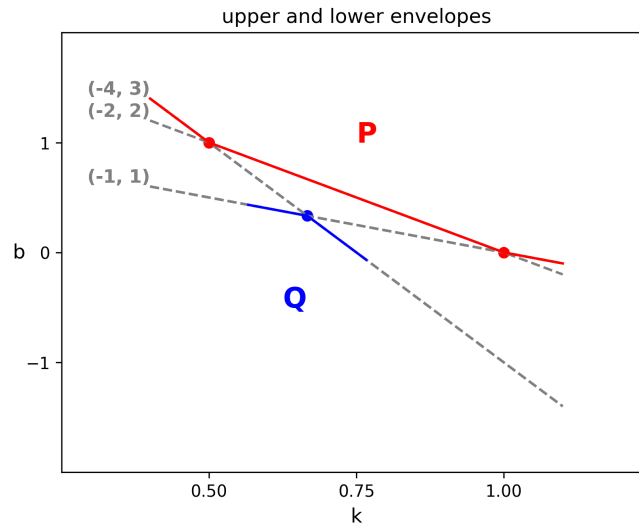


Рис. 1: Upper and lower envelopes for the set $\{(x_i, y_i)\} = (1, 1), (2, 2), (4, 3)$

Note that the system has a solution if and only if the minimum distance along the y-axis between the upper and lower envelopes does not exceed $2 \cdot \mathcal{E}$, since the inequalities in the system arise from parallel translations along the y-axis of inequalities (T1) and (T2)—the lower and upper envelopes by \mathcal{E} upwards and \mathcal{E} downwards, respectively.

Thus, to solve our problem, it is necessary to maintain the current set of lines defined by the pair of coefficients $(-x_i, y_i)$, as well as to maintain the upper and lower envelopes and the minimum distance along the y-axis between them, adding the next i -th line until the minimum distance between the envelopes exceeds $2 \cdot \mathcal{E}$.

Problem J. Strange Sum

Idea: Ildar Gainullin

Development: Ildar Gainullin

Consider the quantity $W_{l,r}$ — the sum of $w_{i,j}$ for all $l \leq i \leq j \leq r$.

Similarly, consider $c_k = \max W_{l,l+k-1}$.

We will break x into summands t_1, t_2, \dots, t_m from 1 to n , in order to maximize $\sum_{i=1}^m c_{t_i}$. In other words, we will solve the knapsack problem, where we need to achieve a total weight equal to x , maximizing the value, with the weight of the k -th item equal to k , and the value equal to c_k .

It is claimed that the answer to the problem is this maximum value.

Indeed, for any answer a_1, a_2, \dots, a_n , we can consider all maximal inclusive segments of numbers that are not less than y for all values of y , and notice that $\sum_y \sum_k c_k$ is not less than the value $f(A)$ from the problem.

On the other hand, starting from an array of zeros, for each item of weight k taken in the knapsack, we can add 1 on the segment where the maximum is achieved, increasing the answer by an amount not less than c_k .

Thus, we have proven that the answer is neither less than nor greater than the result of the knapsack problem, thereby guaranteeing their equality.

It remains to solve the knapsack problem for $x = 10^9$. It is claimed that as long as $x > n^2$, it is beneficial to take the item with the highest $\frac{c_k}{k}$. This is because if the total number of items not equal to k exceeds n (which follows if their sum is $> n^2$), then if we take any n of them in any order, there will be a subsegment among them with a sum of weights divisible by k (since by the pigeonhole principle, two prefix sums will have the same remainder modulo k), meaning all items in this segment can be replaced with some number of items of weight k , without decreasing the value. Using division with a remainder, we can calculate how many items of weight k we will find in this way.

For the remaining $x \leq n^2$, we will solve the knapsack problem in $O(n^2x)$.

Problem K. Two Skew Roads

Idea: Mikhail Ivanov
Development: Mikhail Ivanov

If you need to cross only one of the two streets, you can do so away from the intersection — meaning four of these six numbers will be $\ell_H, \ell_W, \ell_H, \ell_W$. The remaining two distances between the opposite parts need to be found. Two of the parts are acute (or right) angles α , and two are obtuse (or right) angles $180^\circ - \alpha$. The intersection of the streets forms a parallelogram $ABCD$ with angle $\angle DAB = \alpha$ at the base, and this parallelogram has known heights: the distance between AB and CD is ℓ_H , and between BC and DA is ℓ_W . The parts of the city are the angles vertical to the angles of the parallelogram.

Let's start with the pair of angles vertical to $\angle DAB$ and $\angle BCD$. Clearly, for them, nothing better can be found than AC . How to find AC ? We can explicitly construct this parallelogram, but we will do it this way: we will drop a height CC' onto the extension of segment AB beyond point B . Then $AC = \sqrt{(AB + BC')^2 + C'C^2}$.

- $C'C^2 = \ell_H^2$, this is straightforward;
- BC' can be found knowing that in triangle $\triangle BC'C$, we have a right angle $\angle BC'C$, angle $\angle CBC' = \alpha$, and leg $CC' = \ell_H$. Then the other leg is $BC' = CC' \operatorname{ctg} \angle CBC' = \ell_H \operatorname{ctg} \alpha$;
- AB can be found by dropping a height BB' from B onto AD . Then in triangle $\triangle BB'A$, we know the right angle $\angle BB'A$, angle $\angle B'AB = \alpha$, leg $BB' = \ell_W$, so the hypotenuse is $AB = \frac{BB'}{\sin \angle B'AB} = \frac{\ell_W}{\sin \alpha}$.

From this, we have

$$\begin{aligned}
 AC &= \sqrt{\left(\frac{\ell_W}{\sin \alpha} + \ell_H \operatorname{ctg} \alpha\right)^2 + \ell_H^2} = \\
 &= \sqrt{\frac{\ell_W^2}{\sin^2 \alpha} + \frac{2\ell_H \ell_W \cos \alpha}{\sin^2 \alpha} + \frac{\ell_H^2 \cos^2 \alpha}{\sin^2 \alpha} + \ell_H^2} = \\
 &= \frac{\sqrt{\ell_H^2 + 2\ell_H \ell_W \cos \alpha + \ell_W^2}}{\sin \alpha}.
 \end{aligned}$$

Similarly, we can find $BD = \frac{\sqrt{\ell_H^2 - 2\ell_H \ell_W \cos \alpha + \ell_W^2}}{\sin \alpha}$. However, there is a nuance: the distance between the angles vertical to $\angle ABC$ and to $\angle CDA$ may equal BD , but sometimes it is less — specifically, if angle $\angle ABD$ is obtuse or if angle $\angle CBD$ is obtuse. Let's learn to work in one of these cases; the other is analogous. If, say, angle $\angle ABD$ is obtuse, it is faster to move not along segment BD , but along segment DD' , where D' is the projection of D onto AB . So the answer will simply be ℓ_H . But how to determine if angle $\angle ABD$ is obtuse? It is obtuse if and only if $BC' > AB$. So let's compare — both segments we have already calculated: $BC' > AB$ is equivalent to $\ell_H \operatorname{ctg} \alpha > \frac{\ell_W}{\sin \alpha}$ which is equivalent to $\ell_H \cos \alpha > \ell_W$. So if this inequality holds, instead of calculating BD , we simply output ℓ_H . Similarly, if $\ell_W \cos \alpha > \ell_H$, then the answer is ℓ_W .

Problem L. System of Equations with XOR

Idea: Demid Kucherenko
 Development: Evgenii Karpovich

The problem can be solved in different ways; we will discuss one of them (this method is one of the fastest for solving the problem if the numbers x and y are not chosen randomly). We will solve the problem using the branch and bound method — recursive enumeration with pruning. To do this, we will construct the numbers x and y from the most significant bits to the least significant bits.

We can implement a function $get(i, x, y)$ that fixes the bits with indices $i + 1, i + 2, \dots, 61$ in the numbers x and y , while the least significant i bits are not considered yet (they remain zero). We will fix these most significant bits in such a way that the XOR of these most significant bits corresponds to the number b .

To do this, we just need to do the following:

- If the i -th bit of the number b is one, then we have two options for transitions — $get(i - 1, x + 2^i, y)$ and $get(i - 1, x, y + 2^i)$.
- If the i -th bit of the number b is zero, then we have two other options for transitions — $get(i - 1, x + 2^i, y + 2^i)$ and $get(i - 1, x, y)$.

If during the execution of our recursive function we find ourselves in the state $i = -1$, where $x \cdot y = a$, then we have found the answer and can exit the recursion.

Next, we will apply a series of optimizations that will help speed up the solution and avoid considering cases that will not lead us to an answer.

- We can assume that $x \geq y$ (the order of the variables does not matter to us, so we can work under this assumption). To do this, we just need to exclude the transition $get(i - 1, x, y + 2^i)$ if $x = y$.
- Let's learn to estimate the minimum product if we somehow fill in the least significant i bits in the numbers x and y so that their XOR equals b . Let $c = (b \& (2^{i+1} - 1))$ — the least significant i bits of the number b .

Claim: The product of the numbers x and y will be at least $(x + c) \cdot y$ (recall that we assume $x \geq y$).

Then if $(x + c) \cdot y > a$, we can exit this case in the recursion, as we will not find an answer.

- Similarly, we can make an estimate for the maximum product. But here we will need to consider two cases:
 - If $x = y$, the estimate is somewhat more complicated. However, we find ourselves in this case very rarely, so we can make this estimate weaker (greater than it actually is). In this case, let's estimate the maximum product as $(x + 2^{i+1} - 1) \cdot (y + 2^{i+1} - 1)$.
 - If $x > y$. Similarly, let $c = (b \& (2^{i+1} - 1))$ — the least significant i bits of the number b . Then we can estimate the maximum product as $(x + ((2^{i+1} - 1) \oplus c)) \cdot (y + 2^{i+1} - 1)$.

If we find that the maximum product (our upper estimate for it) is less than a , then we can exit consideration of this case in our recursive enumeration.

This solution works correctly and is much faster than the naive recursive enumeration, passing all tests.

Problem M. Mathematical Calendar

Idea: Rita Sablina
Development: Rita Sablina

In February of a non-leap year, there are 28 days, which is divisible by 7, so the number of each day of the week is the same—4. We also note that if the year starts on a day numbered x , then the first of December will be on day $(x - 2) \bmod 7$.

In December and January, there are 31 days, so for example, if the year starts on a Monday, there will be five Mondays, five Tuesdays, and five Thursdays in January, while all other days of the week will have four. December will then start on a Saturday, and there will be five Saturdays, Sundays, and Mondays in December. In this case, the number of Mondays will be two more than the number of Thursdays and Fridays. We note that a situation where the number of days x is two more than another can only occur if the year started on day x . Thus, in our problem, the year starts on a Thursday. Therefore, to solve the problem, we need to count the number of days that have passed since the beginning of the year to the given date P , and the answer will be $(3 + P) \bmod 7 + 1$.