TUYMAADA The 26th International Olympiad

ТУЙМААДА XXVI Международная олимпиада

PROBLEMS AND SOLUTIONS

Informatics

Информатика

ЗАДАЧИ И РЕШЕНИЯ

Yakutsk 2019
Якутск 2019

XXVI Международная олимпиада школьников по математике, физике, химии и информатике «Туймаада». ИНФОРМАТИКА. – Якутск, 2019.

Сборник содержит задачи XXVI Международной олимпиады «Туймаада» по информатике, а также возможные варианты их решений. Олимпиада проводилась 10–11 июля 2019 года в г. Якутске. Олимпиада прошла в два дня, в каждый из которых участникам было предложено решить четыре задачи.

This booklet contains the problems and possible solutions of the 26th *Tuymaada* International Olympiad in informatics. The olympiad took place on 10–11 July 2019 in Yakutsk, Russia. The participants were invited to solve four problems on each of the two competition days.

АВТОРЫ		***AUTHORS***
Владимир Васильевич ЭВЕРСТОВ Северо-Восточный федеральный университет им. М.К. Аммосова, Якутск	**A**	*Vladimir EVERSTOV* *Ammosov North-Eastern Federal University, Yakutsk*
Артем Тарасович ВАСИЛЬЕВ Университет ИТМО, Санкт-Петербург	**B**	*Artem VASILYEV* *ITMO University, St Petersburg*
Артем Тарасович ВАСИЛЬЕВ	**C**	*Artem VASILYEV*
Александру ПЕТРЕСКУ Национальный колледж информатики им. Тудора Виану, Бухарест	**D**	*Alexandru PETRESCU* *Tudor Vianu National College of Computer Science, Bucharest*
Айтал Викторович ДЬЯКОНОВ СВФУ	**E**	*Aytal DYAKONOV* *NEFU*
Владимир Семенович ЛЕВЕРЬЕВ СВФУ	**F**	*Vladimir Leveryev* *NEFU*
Александру ПЕТРЕСКУ	**G**	*Alexandru PETRESCU*
Александру ПЕТРЕСКУ	**H**	*Alexandru PETRESCU*

Problems

A. Collection

Young mathematician Talban collects beautiful stones. For now, his collection contains just 10 different stones. Once in Moscow, during one of the olympiads, he saw in a shop a special display stand for stones with 10 cells. Each cell can fit one stone and has a spot light that can produce either blue or red light. Talban liked this stand very much, so he bought it at once.

At home, he set his stones on the stand right away. While enjoying his spotlighted stones, Talban wondered how many ways there are to arrange the stones on the stand? He found that amount very quickly but suddenly noticed that he hadn't taken into account the colour of spotlights. That problem turned out to be difficult. After several days, he finally managed to find a solution to the problem.

He was very tired and proud of himself. But Talban went further and wondered how to solve this problem in the case of N stones and stand with N cells. Please help the tired Talban to solve this difficult problem!

Input

Single integer N ($N \leq 10^4$) — a number of stones in the collection.

Output

Your program should output a single integer — the number of ways to arrange the collection on a stand. We understand that this number can be huge, so print it modulo $10^9 + 7$.

Scoring

This problem has two subtasks. Points for each are awarded only if the solution passes all the tests from this subtask.

Subtask 1 (points: 40)

$N \leq 17$.

Subtask 2 (points: 60)

$N \leq 10^4$.

Examples

standard input	standard output
3	48
5	3840

B. Rice on the chessboard

You might have heard a legend about an ancient mathematician asking the ruler of the country for a reward for the invention of the chess. He asked to place one grain of rice on the first square of the 8×8 chess board, two grains on the second square, four on the third, and so on, each time doubling the number of grains. But this problem is not exactly about that story.

You have an $n \times m$ board. The cell on the intersection of the i-th row and j-th column contains $a_{i,j}$ rice grains. You want to turn this board into *chess-like* one, where each square has the size of $k \times k$, except for squares on the edge of the board which can have width or height less than k. Each square must be colored black or white, and squares that have a common side must have different colors.

Find a coloring of the given board such that the total number of rice grains on black squares is maximum possible.

Input

First line contains three integers n, m, k ($1 \leq n, m, k \leq 2\,000$, $k \leq \min(n, m)$) — number of rows, number of columns and the size of the square. Next n lines contain m integers $a_{i,j}$ each ($0 \leq a_{i,j} \leq 10^9$).

Output

Output a single integer — maximum possible total number of rice grains on black cells.

Scoring

This problem has four subtasks. Points for a subtask are awarded only if solution passes all the tests from this subtask and preceding subtasks.

Subtask 1 (points: 30)

$n, m \leq 50, k = 1$.

Subtask 2 (points: 20)

$n, m \leq 50$.

Subtask 3 (points: 20)

$n, m \leq 500$.

Subtask 4 (points: 30)

No additional limitations.

Examples

standard input	standard output
4 5 2 5 0 1 6 5 0 7 9 0 2 0 5 3 2 1 8 1 1 4 3	55
3 3 1 1 2 3 4 5 6 7 8 9	25

Note

In the first example the optimal coloring is to color $(2, 2) - (3, 3)$ black (after that, colors of all other cells are determined uniquely).

C. Powerful partition

You are given a string consisting of `0` and `1`. You have to insert some `+` signs into this string in such a way, that calculating the resulting expression in binary gives a power of two.

Input

Input contains one line S ($1 \leq |S| \leq 10^6$) composed of characters `0` and `1`.

Output

If it is impossible to place `+` signs so that the result is a power of two, output "`NO`". Otherwise, if a solution exists, output "`YES`" on the first line. Then on the second line print the required expression consisting of `0`, `1` and `+` characters.

After removing the **+** signs, the expression must be equal to input string *S*. The first and the last characters can not be **+**, and there must be no adjacent **+** signs. Leading zeros are allowed. In case there are multiple correct solutions, print any one of them.

Scoring

This problem has three subtasks. Points for a subtask are awarded only if solution passes all the tests from this subtask and preceding subtasks.

Subtask 1 (points: 25)

$|S| \leq 16$.

Subtask 2 (points: 35)

$|S| \leq 500$.

Subtask 3 (points: 40)

No additional limitations.

Examples

standard input	standard output
01001101	YES 01+0+01+1+0+1
11111	YES 1111+1

D. Verkhoyansk

Marcel has planned a trip in the Verkhoyansk Range, a 1200 km mountain range in the Sakha Republic. The landscape can be seen as an array of *N* integers with values between 1 and *N*, representing the heights of the mountain peaks along the range.

Marcel has *Q* friends. Friend *i* will visit all peaks with indices from $L[i]$ to $R[i]$. Marcel wants to know, for each of them, what is the smallest positive integer height of a peak each friend will not visit. He needs to know this in order to optimally plan his next trip.

For example, if one friend visits peaks with heights 3 2 5 1 1 6 3 5, the smallest positive integer height of a peak he didn't visit is 4.

Input

The first line contains numbers $1 \leq N \leq 300.000$ and $1 \leq Q \leq 600.000$. The second line contains N integers with values between 1 and N, representing the heights of mountain peaks. The following Q lines contain 2 numbers, $L[i]$ and $R[i]$, with $0 \leq L[i] \leq R[i] \leq N - 1$.

Output

There will be Q lines. Line i contains the smallest positive integer height of a peak friend number i will not visit.

Scoring

This problem has four subtasks. Points for a subtask are awarded only if solution passes all the tests of a subtask.

Subtask 1 (points: 20)

$N \leq 1.000$ and $Q \leq 10.000$.

Subtask 2 (points: 30)

$N \leq 100.000$ and $Q \leq 200.000$ and all the heights are at most 50.

Subtask 3 (points: 30)

$N \leq 100.000$ and $Q \leq 200.000$.

Subtask 4 (points: 20)

$N \leq 300.000$ and $Q \leq 600.000$.

Note

Note that the array of heights is ”0-indexed”

Examples

standard input	standard output
14 16 3 4 3 2 5 1 6 7 2 1 6 2 4 3 0 4 0 5 9 13 9 12 3 12 2 12 2 11 1 11 3 13 5 13 8 13 0 7 0 8 6 8 6 9 6 10	1 6 5 3 3 8 4 8 8 5 5 8 8 1 3 3

E. Arithmetic progressions

This year, young programmer Grigory has learnt a lot of new mathematics. In particular, he liked arithmetic progressions very much. He doesn't like dry theory, so he decided to explore this exciting topic. Grigory took two arithmetic progressions and wants to know how many common elements they have inside the segment from l to r (endpoints included). Help him solve this problem.

Recall that an arithmetic progression is a sequence of numbers of the form

$$a_1,\ a_1 + b,\ a_1 + 2b,\ \ldots,\ a_1 + (n-1)b,\ \ldots,$$

that is, a sequence where each member starting with the second one is obtained by adding a constant b (the *difference* of the progression) to the previous one:

$$a_n = a_{n-1} + b.$$

The formula for the nth number in the sequence is:

$$a_n = a_1 + (n-1)b.$$

Input

The first line contains two integer numbers a and b ($0 \leq a, b \leq 10^9$), the initial term and the difference of the first arithmetic progression. The second line contains two numbers c and d ($0 \leq c, d \leq 10^9$), the initial term and the difference of the second arithmetic progression. The third line contains two integer numbers l and r ($0 \leq l \leq r \leq 10^9$), the endpoints of the segment.

Output

Output the number of common terms of the two progressions inside the interval $[l, r]$.

Examples

standard input	standard output
3 2 5 6 10 20	2
30 0 15 15 10 1000	1

Note

This task has four subtasks. Points for a subtask are awarded only if solution passes all the tests from this subtask and preceding subtasks.

Subtask 1 (points: 20)

$a, b, c, d, l, r \leq 10^3$.

Subtask 2 (points: 20)

$a, b, c, d, l, r \leq 10^6$.

Subtask 3 (points: 40)

$a, b, c, d \leq 10^6$.

Subtask 4 (points: 20)

No additional limitations.

F. Superheroes

For many years, a war has been going on between a team of superheroes and a gang of supervillains.

Each superhero or supervillain has one of five superpowers:

- (S) super speed — an ability to move blazingly fast;
- (P) super strength — incredible physical strength and health;
- (T) telepathy — an ability to read and control people's minds;
- (K) telekinesis — an ability to move objects by the force of thought;
- (E) energy beam — an ability to shoot at enemies with an energy beam.

The result of a fight of two superpower owners can be predicted by the following rules:

- super speed defeats telepathy and super strength;
- telepathy defeats super strength and telekinesis;
- super strength defeats telekinesis and energy beam;
- telekinesis defeats energy beam and super speed;
- energy beam defeats super speed and telepathy.

On the eve of the final battle, the superheroes have obtained the list of supervillains and their superpowers. It is known that the upcoming battle will consist of a series of one-on-one duels. The number of superheroes in the battle is equal to the number of supervillains, and each participant fights exactly once.

Luckily, the team of superheroes can choose who will fight whom. You must help them to make the right choice to win the maximum number of duels.

Input

The first line of the input data contains an integer N ($1 \le N \le 10^5$), the number of participants in the battle on one side.

The second line of the input data contains a string of N characters denoting the types of superpowers that the supervillains possess.

The third line of the input data contains a string of N characters denoting the types of superpowers possessed by the superheroes.

Output

Print a string of N characters denoting abilities of the superheroes. This string must be a rearrangement of the characters of the third line of input

data. This heroes arrangement must allow them to win the maximal number of duels.

Each *i*-th character in it corresponds to a superhero with this type of superpower who, must fight the supervillain whose superpower is encoded by the *i*-th character on the second line of the input.

Scoring

This problem has two subtasks. Points for each test are assigned individually.

Subtask 1 (points: 30)

There can be heroes and villains with only three abilities: S (super speed), P (super strength) and K (telekinesis).

Subtask 2 (points: 40)

There can be heroes and villains with all five abilities, $N \leq 100$.

Subtask 3 (points: 30)

There can be heroes and villains with all five abilities, $N \leq 10^5$.

Example

standard input	standard output
5 STPKE STPKE	ESTPK

G. Tygyn

One of Marcel's ancestors, called Manchaary, son of Nyurgun and Sahayaana, was one of the servants of Tygyn Darkhan, at the beginning of the 17th century. Tygyn Darkhan, the great leader who unified the Yakutian tribes, seems to have been very passionate about Number Theory.

One day, he assigned some distinct integers from 2 to *M* to his *N* cows. He decided to group the cows, based on their numbers, into as few herds as possible. However, he set the following conditions that should be met for all herds:

- Let x be the smallest number assigned to the cows in the herd. Each other number of a cow in the herd is of the form $x \times k$, with k integer.

- All divisors of k (apart from 1) are greater than or equal to any prime divisor of x, for all cows in the herd.

However, Tygyn is a bit busy with keeping peace in his territory, so he asked Manchaary to take care of the cows. The task was rather difficult for Manchaary, but the reward was magnificent: he could then marry beautiful Sardaana!

We know the end of the story: Manchaary solved the task and Sardaana became an ancestor of Marcel. However, when Marcel learned about the story, he thought the task can go back to the lands it originated. That is why the participants in Tuymaada 2019 have to solve it!

Input

The first line contains integer numbers N, M ($1 \leq N < M \leq 10^6$). The next line contains N numbers with values between 2 and M, representing the distinct numbers assigned to the N cows of Tygyn Darkhan.

Output

The first line contains the minimal number of herds Manchaary could group the cows into.

Scoring

This task has two subtasks. Points for a subtask are awarded only if solution passes all the tests from this subtask.

Subtask 1 (points: 30)

$N \leq 1000$.

Subtask 2 (points: 70)

No additional limitations.

Examples

standard input	standard output
5 100 2 3 11 22 12	3
10 20 8 12 20 6 3 7 11 13 19 15	9
14 20 4 14 3 5 7 11 13 17 19 12 15 16 8 9	9

H. Competition

There are $N + 1$ people competing in some event for N days. Each day, exactly one of them is declared the winner of the day. The score of some participant is equal to the number of days he was winner. After each day, the participants with the highest score receive a coin. After the competition is over, each participant has some happiness value, calculated the following way: for every discretely continuous maximal interval when he receives coin, add to his happiness the square of the length of the interval.

For example, if some contestant won coins on days $3, 4, 10, 11, 12, 18$ and 19, the intervals are $[3-4]$, $[10-12]$ and $[18-19]$, while his happiness is equal to $2^2 + 3^2 + 2^2 = 4 + 9 + 4 = 17$. The outcome of the competition is the sum of happiness for all participants.

Now Marcel comes in, and he is able to insert, somewhere in the array of days, one day that will surely be won by participant number 0.

You are given an array of N integers between 0 and N, representing the winner of each day. Let $f(p)$ = the outcome of the competition if we would insert number 0 in this array after the p'th element in the array. You need to print numbers $f(0), f(1), ..., f(N)$.

For example, if the array of 3 elements is 0 1 1, $f(0)$ = the outcome of the competition **0** 0 1 1. Participant number 0 receives coins in the days 1, 2, 3 and 4. So his happiness is $4^2 = 16$. Participant number 1 receives a coin on day 4. His happiness is $1^2 = 1$. Participants 2 and 3 receive no coins. So $f(0) = 17$. $f(N = 3)$ = the outcome of the competition 0 1 1 **0**. Participant number 0 receives coins in the days $1, 2, 4$ so his happiness is $4 + 1 = 5$. Participant number 1 receives coins in the days $2, 3, 4$ so his happiness is 9. So $f(3) = 14$.

Input

The first line contains a number N ($1 \leq N \leq 10^6$), and on the following line there are N numbers with values between 0 and N, representing the winners of the competition on each day.

Output

There are $N + 1$ lines. Line i contains number $f(i - 1)$.

Scoring

This task has four subtasks. Points for a subtask are awarded only if solution passes all the tests from this subtask and preceding subtasks.

Subtask 1 (points: 11)

$N \leq 100$.

Subtask 2 (points: 13)

$N \leq 3000$.

Subtask 3 (points: 39)

$N \leq 10^5$.

Subtask 4 (points: 37)

$N \leq 10^6$

Examples

standard input	standard output
4 0 4 4 4	20 20 21 21 20
4 1 0 1 1	21 17 17 27 26
4 2 1 1 0	23 23 27 21 21
10 1 2 3 1 2 3 0 1 2 3	140 154 154 154 145 152 157 157 144 149 152

standard input	standard output
10 4 9 10 1 5 3 4 10 3 0	218 226 226 226 226 226 226 190 197 202 202

Tutorials

A. Collection

There are $N!$ ways to arrange N stones on a stand with N cells on it. If we take into account that each cell is lit by one of two colors then there are $2^N \times N!$ ways to set stones on the display stand. This number may be huge, so it is better to use the following formula $a_N = a_{N-1} \times 2 \times N$, with $a_0 = 1$. Then the solution of the initial problem is the Nth member of this sequence. We are to find it modulo $10^9 + 7$. Hence, one should use this formula: $a_N = a_{N-1} \times 2 \times N \bmod (10^9 + 7)$.

B. Rice on the chessboard

Subtask 1 can be solved by calculating sum of all cells (r, c), where $r + c$ is even, $r + c$ is odd, and taking the maximum of these two sums. Since $a_{i,j}$ can be up to 10^9, their sum can be up to $\frac{1}{2} \cdot n \cdot m \cdot 10^9 \approx 10^{12}$, which exceeds the maximum value of a 32-bit **`int`** variable. Don't forget to use 64-bit type in this calculation.

For $k > 1$ every chess-like coloring can be described by the size and color of the left-most top-most rectangle. After the color and the size $h \times w$ of the rectangle are fixed, the colors of each cell on the board can be determined uniquely. Every such rectangle has size $h \times w$, where h and w are less than or equal to k.

If we try all possible combinations of h and w, calculating the sum naively for each combination, we take $O(nmk^2)$ time and pass the first two subtasks.

All that is left to do is to find the sum faster. This can be done using 2D prefix sums. Let's calculate the array $sum(i, j) = \sum_{r \le i,\ c \le j} a_{r,c}$ – sum on a rectangle $(1, 1) - (i, j)$. Using this array and the inclusion-exclusion principle, we can find the sum of any rectangle $(r_1, c_1) - (r_2, c_2)$: $sum(r_2, c_2) - sum(r_1 - 1, c_2) - sum(r_2, c_1 - 1) + sum(r_1 - 1, c_1 - 1)$.

Every chess-like coloring contains at most $(\lfloor \frac{n}{k} \rfloor + 2)(\lfloor \frac{m}{k} \rfloor + 2)$ rectangles of the same color, and we can calculate the sum of each one of them in constant time. This way, calculating the sum for a fixed coloring can be done in $O(\frac{nm}{k^2})$ time. Since there are $2k^2$ different chess-like coloring, the total time complexity of this solution is $O(k^2) \cdot O(\frac{nm}{k^2}) = O(nm)$. Good implementation of the solution above should score 100 points.

C. Powerful partition

Subtask 1 with $|S| \leq 16$ can be solved with brute-forcing all possible placements of **+** signs. String of length n has 2^{n-1} different plus placements (between each pair of adjacent digits we can either put **+**, or not). Then, sum all numbers in binary and check if the result is a power of two. This check can be performed with just one bit operation: x is a power of two iff $x > 0 \wedge x \mathbf{and} (x-1) = 0$.

To solve subtask 2 you have to make a following assumption: let the resulting power of two be small (not exceeding 2^l). Then you can implement a dynamic programming solution: let $dp[i][j]$ be equal to 1, if we can get j using first i digits, and 0 otherwise. Transition is done in $O(l)$: try all possible lengths of the next number (up to l). The total time complexity is $O(nl2^l)$. If we choose l such that $2^l \approx n$, then the time becomes $O(n^2 \log n)$, which is fast enough for the second subtask.

We can implement this solution and check that it finds an answer for all strings with at least one 1. It's possible to prove that the answer always exists, moreover for all strings except **11111** the length of each number is at most 3.

Initially, let's place all possible **+** between all adjacent digits. Current sum is equal to m, where m is the number of ones in the input string. Let $2^{k-1} < m \leq 2^k$. It is always possible to achieve sum 2^k by concatenating adjacent digits into one number. Notice, that concatenating $1 + x \to 1x$ increases the sum by 1, $1 + 0 + x \to 10x$ — by 3, and $1 + 1 + x \to 11x$ — by 4. We will act greedily, concatenating triples of digits while the current sum doesn't exceed 2^k. After that we'll do the same with pairs of digits. We can show that for a large enough number of ones (16 is enough), this algorithm always achieves 2^k. For smaller values of m we can check it by hand; but the algorithm fails for $m = 5$.

Let's solve $m = 5$ separately. If there is a substring $10x$, then we make it a separate number and get 8. Otherwise, the input string is either $0 \ldots 011111$ or $0 \ldots 0111110$. In the first case the answer is $1111 + 1$, in the second one — $11 + 11 + 10$. This greedy and case handling can be done in linear time, which gets full score for this problem.

D. Verkhoyansk

For 20 points. Brute-Force. Process queries online, as you read then from input. Use a bool array *visited* and go from the left end to the right end of the query interval and mark the heights seen as visited. Then go with the supposed *ans* from 1 and increase it while *visited*[*and*] is *true*. Don't forget to reset to 0 the values in *visited* in order to be used correctly for the next queries.

For 80 points. Divide the array of heights into contiguous buckets of size K. There will be $\frac{N}{K}$ such buckets. K must not necessarily divide N. The last bucket can have a smaller size.

We will answer offline to the given mex range queries. For each bucket, we will consider all queries having the left end in the selected bucket and answer all of them. After we have taken into consideration all buckets, all queries will have found their answer and we can print them in order.

Suppose we want to compute the answers for all queries having the left end in some bucket B. Their right ends can be arbitrarily high, but their left ends are all in some interval from $B * K$ to $(B + 1) * K - 1$. We will call the heights found in this interval special. All other heights are not special.

There are at most K special heights and at most $K + 1$ contiguous intervals of non-special heights. For example, if the special heights are 5 and 23, and $N = 100$, the intervals of non-special heights are $[1-4]$, $[6-22]$ and $[24-100]$.

Now, for each query, a non-special height can only be visited outside the bucket, while the special ones can also be visited inside it. The idea behind the solution is to keep a partial mex for each interval of non-special heights, considering only the heights from $(B + 1) * K$ to the right end of the query.

To be more precise, *partialMex*[x] will keep the smallest integer value greater then or equal to x that can't be found between the heights from $(B+1)*K$ to the current right end.

It is also important to notice that it is enough to calculate *partialMex*[x] only for the xs at which an interval of non-special heights start. For the example above, *partialMex*[x] is relevant only for $x = 1$, 6 or 24.

In order to keep these values properly, we will sort the queries (considered for the current bucket) by their right ends. The first queries will have their right ends inside the bucket, so we can iterate the heights in the query interval and calculate their mex in a brute-force manner, as we will make at most K steps.

The next queries will start to increase the right end, and we will mark the new heights as visited. Note that we only mark heights found outside the

bucket in the *visited* bool array. Note, also, that we mark all heights, no matter they are special or not.

For each of the queries, we will calculate the answer the following way. We start with $ans = 1$ and try to increase it. At each step, *ans* represents a height which is either special or not. If it is special, increase *ans* if the height can be found in the bucket, after (to the right of) left end, or it is marked as visited, else stop and return ans. If it is not special, it means it belongs to some interval, and we should ask the corresponding *partialMex* how much can *ans* increase. If the partial mex tells us the hole interval has been seen, we can continue with our expansion with *ans* from $1 + right - end - of - the - interval$. If we are stuck in the middle of the interval, it means the partial mex is the answer to the query. This way, we use exactly all the heights in the query interval.

The official implementation calculates $partialMex[x]$ lazyly (not as we mark heights as visited, but as we need its value when answering a query). So, when we have some *ans* equal to the first non-special height of the current interval, we try to increase $partialMex[ans]$ as long as we remain in a non-special interval of heights and as long as the value has been visited outside the bucket (marked in the *visited* bool array). Then we say $ans = partialMex[ans]$ and move on.

To sum it up, for each bucket, as we move the right end along answering the queries, we keep track of the partial mex of each interval of non-special heights. This makes it possible that for each query we can see at most $2 * K + 1$ values: the special heights and the $K + 1$ intervals, which we jump in constant time because of the partial mex we keep for each of them. Also, the partial mex for each interval can only increase $interval - length$ times, giving a total of at most N steps for each bucket.

Let's analyze the complexity of the algorithm. For each bucket, we visit $O(N)$ positions, because the highest right end of a query is $N - 1$. We also increase *partialMex* $O(N)$ times. For each query, we make $O(K)$ suplimentary steps to find out the answer. Therefore, the complexity is $O(\frac{N*N}{K} + Q * K)$ which reaches its minimum when K equals $\frac{N}{\sqrt{Q}}$, giving a complexity of $O(N * \sqrt{Q} + Q)$.

However, the solution requires a sorting of the queries by their right ends for each bucket, so the final complexity is $O(N * \sqrt{Q} + Q * logQ)$

For 100 points. We will process the queries offline, in increasing order of their right ends. As we move along the heights array and answer the queries, we need to remember, for each height, which was the rightmost position we encountered it in the array. Suppose we know these values. Then, for each

query, we need to find out the highest value *val* such that all values from 1 to $val - 1$ have this position greater than or equal to the left end of the query.

We can use a segment tree over the values. More precisely, suppose we keep this array, $rightMostPosition[x]$ with the meaning, what is the rightmost position where the height is x, encountered so far (up the right end of the current query). $rightMostPosition[x] = -1$ if there was no height equal to x so far. The segment tree will simply store the minimum over these values.

When we change the right end of the current query, we can simply notify the array *rightMostPosition* with the new values encountered. Every change in *rghtMostPosition* means $O(\log N)$ changes in the segment tree. Therefore, we know how to update the segment tree, but how to find the mex with it?

We will binary search the answer, using the segment tree. Each node of the segment tree will tell us whether all values in the node's segment can be seen in the query interval. The condition is that $segmentTree[node]$ to be greater than or equal to the left end of the query. Using this condition, we can binary search the answer while moving along the segment tree in $O(\log N)$.

The final time complexity of the solution is $O((N + Q) * \log N)$.

E. Arithmetic progressions

The degenerate case when any difference is zero is easily treated separately.

Subtask 1

A brute force computation, for example saving all the members of the progressions inside the interval $[l, r]$ into arrays and checking for common values, will solve the problem in $O(r^2)$.

Subtask 2

We check if $k \in [l, r]$ appears in both progressions, for all k. A number k is of the form $k = a + bn$ iff $a \leq k$ and $(k - a)$ is divisible by b. This gives an $O(r)$ solution.

Subtasks 3, 4

For these subtasks, find the first common member:

$$a + bn = c + dm, \qquad n, m \geq 0$$

This is a linear Diophantine equation in n, m. It can be solved, for example in $O(\max(b, d))$ as follows. Rewrite it as

$$n = \frac{c - a + d * m}{b},$$

Swapping the progressions if necessary, we'll assume that $a \leq c$. It is sufficient to consider the values of m from 0 to b. If there is no solution in this interval, then it doesn't exist at all. This search has complexity $O(\max(b, d))$. Solving the Diophantine equation using the extended Euclid algorithm gives an $O(\log(\max(b, d)))$ solution which allows solving Subtask 4.

Denoting by t be the minimum common member of both progressions, we note that the common members form an arithmetic progression with initial value t and the difference of $\mathrm{LCM}(b, d)$, where LCM is the least common divisor function.

Denote by $f(a, b, p)$ the number of the elements of the arithmetic progression with an initial value a and difference b in the interval from 0 to p:

$$f(a, b, p) = \begin{cases} 0 & \text{, if } p < a \\ \lfloor \frac{p-a}{b} \rfloor + 1 & \text{, otherwise} \end{cases}.$$

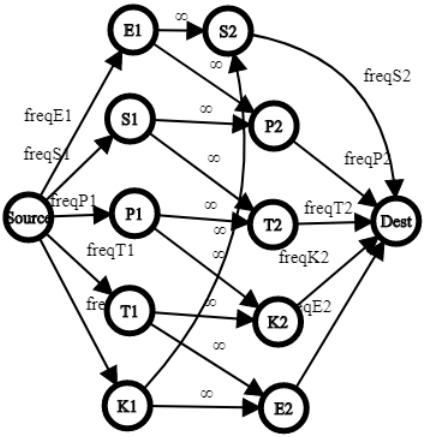
Then the answer is $f(t, \mathrm{LCM}(b, d), r) - f(t, \mathrm{LCM}(b, d), l - 1)$

F. Superheroes

Let's construct a directed bipartite graph 5×5. Five vertices on the left side correspond to superheroes' superpowers, and five vertices on the right side — to those of the supervillains. Assign an integer to each vertex on the left — the number of available superheroes with this superpower. Do the same on the right for the supervillains. Finally, construct two oriented edges going out from each vertex on the left in accordance with the rules determining the winner of a duel from the winner to the loser.

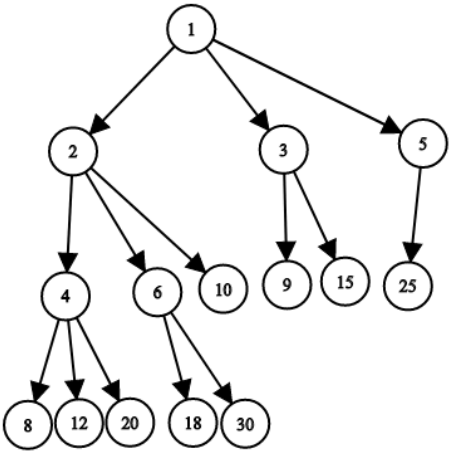
There are several methods allowing us to solve this problem.

For example, we can consider this problem as a flow maximization problem. From this point of view, the number of superheroes with each superpower is the capacity of a flow source, while the the number of supervillians with each superpower is the capacity of the corresponding sink. After finding maximum flow it will show us which superheroes should fight which supervillains. Remaining unallocated superheroes can fight any unallocated supervillains.



G. Tygyn

The solution is based on a special and innovative tree that contains all natural numbers. We will call this special tree the Divisor Tree. The root of the tree is vertex number 1. Each number greater than 1 can be written as a product of prime numbers. The father of any vertex is the number equal to the number divided by the largest prime number dividing it. For example, $3 * 5 * 5 * 7$ has $3 * 5 * 5$ as father, while $2 * 2 * 2$ has $2 * 2$ as father.



In order to understand its structure, you can have a look on the image above. Here you can find a descriptive part of the Divisor Tree. Note that all prime numbers share an edge with root number 1, and the height of each vertex is equal to the number of times you should divide it by a prime number in order to make the number equal to 1 (the sum of exponents of the prime numbers in its factorisation).

If we have a look at the Divisor Tree, at its properties and structure, in relationship with the task statement, we will see that one can make a herd of all the cows whose numbers are in a subtree of an existing cow. Minimizing the number of herds means creating a heard for each cow, except for the ones who have an ancestor in the tree. If the cow has an ancestor in the tree, it means it can go in the herd whose representant (the minimum number in the herd) is the existing ancestor. This way, both the correctness of the herds and their minimality are respected.

There are many ways to implement the algorithm. We must somehow search the tree, so we must keep the highest prime divisor for any number as we visit all vertices. The official implementation uses BFS on the Divisor Tree, marking as it goes the herd each vertex belongs to. The input is easily treated with a bool array marking all given cows.

The solution also requires some knowledge of the prime numbers up to M. We can use the sieve of Eratosthenes to find them, or the linear sieve for finding all primes up to some number. The final complexity is $O(N + M * \log(\log M))$ or $O(N + M)$, depending on how the sieve is implemented.

H. Competition

We say competition $I(p)$ is the configuration of winners if we simply insert a 0 between positions p and $p+1$ in the array. We compute $f(p)$, the total happiness given competition $I(p)$.

For 11 points. We compute happiness of the competition for each of the $N + 1$ instances independently, as follows.

Given the array of winners, just keep a list of the people who receive a coin (using a frequency array, storing each person's frequency up to the current day). This list can be updated each day the following way:

case 1: Someone from the list wins the day. This means he will be the only one to receive a coin on this day. So we can erase all others from the list

and just keep him.

case 2.1: The one who wins has its frequency incremented up the maximum frequency so far, so we just add him into the list.

case 2.2: The one who wins still has a lower frequency than the maximum one, so the people who receive coin the current day are the same. List does not change.

Each day, after updating the list properly, we can go through the people in the list and add the happiness received from them this day. It is enough to know when was the time each person entered the list (if he entered more than once, we are interested only in the last time he entered the list), so we can keep an array $start[x]$ = the last time x entered the list. We can keep this array by $O(1)$ updates on it each day, treating properly the cases.

Because $a^2 = (a-1)^2 + 2a - 1$, additional happiness obtained for a person in the a-th day of consecutive coin receiving is $2a - 1$. So we can just add $2 * (day - start[x] + 1) - 1$ (for each person in the list) and obtain the current total happiness.

This is $O(N^3)$ because we compute the total happiness of some array of winners $O(N)$ times. Each computation requires $O(N)$ days, and on each day we go through all members in the list, and the list may have up to $O(N)$ people.

Note there are easier solutions for 11 points, we showed this one because it is easier to get to the next step with it.

For 24 points. We just optimize the computation for a given situation of the competition. Instead of going through the hole list and add $2 * (day - start[x] + 1) - 1$ for each of them, just keep this sum and the list size as two variables. We don't even store the list as an array as before, it will be stored only conceptually. Let's treat the cases: (suppose person x is the winner of the current day)

case 1: $sum = day - start[x]; size = 1;$

case 2.1: $start[x] = day; size = size + 1;$

case 2.2: do nothing.

Then just add $size$ to sum. Finally, when we want to add the happiness of the day, just add to our answer $2 * sum - size$.

This is clearly $O(N^2)$ because we compute the answer for each of the $O(N)$ configurations of the competition we just go along the $O(N)$ days and make some $O(1)$ computation each day.

For 100 points. In order to solve the task in $O(N)$, we must avoid computing each $f(p)$ independently. The first thing one can notice is that the coin distribution for $I(p)$ is the same as for $I(N)$ up to position p inclusive, and from position $p+1$ to $N+1$ the distribution is the same as for $I(0)$. Here, we can imagine starting with $I(N)$ and, after position p, simply move on to $I(0)$ and go on from there. This is because, for positions $p+1$ to N, having an artificially inserted 0 on position p or on position 0, it does not matter, as the order of numbers in the prefix is no longer relevant, only frequency matters.

Although this way we are able to know the coin distribution only by looking at $I(0)$ and $I(N)$, we are still unable to compute $f(p)$ efficiently. We need a stronger claim.

Suppose you are in $I(0)$ in position p and the days start passing by, and you only go in case 2. Until one first day, when you have a case 1 situation. Let this day be called $t(p)$ = first breaking point strictly after day p in $I(0)$. Now we will use it in the claim:

The distribution of coins in $I(p)$, except for 0, is the same as in $I(N)$ for all days up to $t(p)$, and from $t(p)$ to N it is the same as in $I(0)$. This is true because artificially incrementing *frequency*[0] has no effect on "who receives a coin each day" until a case 1 situation.

The claim makes it possible to compute $f(p)$ in $O(1)$ due to the fact that there is only one person who "survives" day $t(p)$. But what do we really need to compute the answer now?

At first, we will compute some partial sums for the happiness increase each day in $I(0)$ and $I(N)$. Also, for $I(0)$, we need to keep some information on case 1 days: what day it is, who survives it, the left end of survivor's segment of coin receiving, the right end of survivor's segment of coin receiving. We can also compute all $t(p)$ in $O(N)$. Note that the claim excludes 0 from the coin distributions, so we will compute all we need to know about 0 separately. He will not enter the computations for the partial sums.

Now we can compute $f(p)$ in $O(1)$. At first, we will add the partial sum of happiness up to day $t(p)$ in $I(N)$, and the partial sum of happiness from day $t(p)$ to N in $I(0)$. Now, we must repair the computations for the segment who survived, because, if the survivor is not 0, the partial sums got him false

happiness, which we can compute correctly using the data stored for each breaking point.

We must still treat the happiness of 0, but it can be computed efficiently in a similar manner, using its segments of coin receiving and some partial sums.

The final complexity is $O(N)$ because we can obtain in $O(N)$ all the precomputations about $I(0)$ and $I(N)$ and we can print each of the $N + 1$ numbers using $O(1)$ time.

Задачи

A. Коллекция

Юный математик Талбан коллекционирует камни. Коллекция у него небольшая и пока состоит из 10 разных камней. Однажды во время поездки на одну из олимпиад в Москву он заметил в магазине подставку для камней из 10 отсеков, и при этом каждый отсек вмещал один камень и мог быть подсвечен либо красным, либо синим цветом. Она ему так понравилась, что он ее сразу купил.

Приехав домой, он сразу расставил свои камни на новой подставке. Любуясь красиво подсвеченными камнями, пытливый Талбан подумал, а сколькими способами он сможет расставить свои камни на этой подставке. Эта задача была быстро решена, но потом, он заметил, что не учел цвет подсветки камней. И тут задача для него оказалась сложной, на поиск решения Талбан потратил несколько дней, но все-таки решил ее.

После марафона с этой задачей, наш юный математик подумал: «А что, если бы у меня было не 10, а N камней, которые нужно расставить в аналогичной подставке с N отсеками?» Помогите Талбану решить эту сложную для него задачу!

Формат входных данных

На вход программе дается единственное натуральное число N ($N \leq 10^4$).

Формат выходных данных

Программа должна вывести единственное число — количество способов расстановки камней. Мы понимаем, что это число может быть очень большим, поэтому выведите его по модулю $10^9 + 7$.

Система оценки

Данная задача содержит две подзадачи. Баллы за подзадачу начисляются только, если все тесты этой подзадачи пройдены.

Подзадача 1 (баллы: 40)

$N \leq 17$.

Подзадача 2 (баллы: 60)

$N \leq 10^4$.

Примеры

стандартный ввод	стандартный вывод
3	48
5	3840

B. Рис на шахматной доске

Возможно, вы слышали о легенде, по которой один древний математик попросил у правителя страны награду в качестве рисовых зерен. Математик достал обычную шахматную доску 8×8, попросил положить одно зерно на первую клетку, два зерна на вторую клетку, четыре — на третью, и так далее, каждый раз удваивая. Но эта задача не совсем об этой истории.

У вас есть доска размера $n \times m$. В клетке, находящейся на пересечении строки с номером i и столбца с номером j находится $a_{i,j}$ зерен. Вы хотите придать этой доске *почти шахматную* раскраску, где каждый квадрат имеет размер $k \times k$ клеток, но квадраты на границе доски могут быть обрезаны и иметь ширину или высоту меньше, чем k. Каждый квадрат раскрашен либо в черный, либо в белый цвет, и любые два соседних по границе (но не по диагонали) квадрата должны иметь разные цвета.

Найдите такую почти шахматную раскраску заданной доски, что суммарное количество зерен на черных клетках максимально.

Формат входных данных

В первой строке содержатся три целых числа n, m, k ($1 \leq n, m, k \leq 2\,000$, $k \leq \min(n, m)$) — количество строк и столбцов доски, а также размер квадрата в почти шахматной раскраске. Следующие n строк содержат по m чисел $a_{i,j}$ ($0 \leq a_{i,j} \leq 10^9$).

Формат выходных данных

Выведите одно число — максимальное количество зерен на черных клетках, которое можно получить.

Система оценки

Данная задача содержит четыре подзадачи. Баллы за подзадачу начисляются, только если все тесты этой и всех предыдущих подзадач пройдены.

Подзадача 1 (баллы: 30)

$n, m \leq 50, k = 1$.

Подзадача 2 (баллы: 20)

$n, m \leq 50$.

Подзадача 3 (баллы: 20)

$n, m \leq 500$.

Подзадача 4 (баллы: 30)

Без дополнительных ограничений.

Примеры

стандартный ввод	стандартный вывод
4 5 2 5 0 1 6 5 0 7 9 0 2 0 5 3 2 1 8 1 1 4 3	55
3 3 1 1 2 3 4 5 6 7 8 9	25

Замечание

В первом примере оптимальной раскраской будет покрасить квадрат $(2, 2) - (3, 3)$ в черный цвет (цвета всех остальных клеток определяются однозначно).

C. Двоичное разбиение

Вам дана строка из символов **0** и **1**. Вам необходимо добавить в нее символы **+** таким образом, что если вычислить получившееся выражение,

рассматривая строки из `0` и `1` как числа в двоичной системе счисления, то получится степень двойки.

Формат входных данных

Входной файл содержит единственную строку S ($1 \leq |S| \leq 10^6$), состоящую из символов `0` и `1`.

Формат выходных данных

Если расставить знаки `+` нужным образом нельзя, то выведите «`NO`». Если решение есть, выведите «`YES`», а во второй строке выведите искомое выражение, состоящее только из символов `0`, `1` и `+`. После удаления символов `+` выражение должно быть равно исходной строке S. Первый и последний символы не должны быть равны `+`, а также не должно быть двух `+` подряд. В числах разрешены ведущие нули. Если решений несколько, выведите любое.

Система оценки

Данная задача содержит три подзадачи. Баллы за подзадачу начисляются, только если все тесты этой и всех предыдущих подзадач пройдены.

Подзадача 1 (баллы: 25)

$|S| \leq 16$.

Подзадача 2 (баллы: 35)

$|S| \leq 500$.

Подзадача 3 (баллы: 40)

Без дополнительных ограничений.

Примеры

стандартный ввод	стандартный вывод
01001101	YES 01+0+01+1+0+1
11111	YES 1111+1

D. Верхоянский хребет

Марсель планирует посетить Верхоянский хребет в Республике Саха (Якутия) протяженностью 1200 км. Ландшафт этого хребта можно представить в виде массива N натуральных чисел из интервала от 1 до N, представляющих высоты пиков вдоль хребта.

У Марселя Q друзей. Его друг с номером i будет посещать все вершины с индексами от $L[i]$ до $R[i]$. Для каждого из своих друзей Марсель хочет узнать, какова минимальная высота горы, которую он не посетит в этот раз. Эта информация ему необходима, чтобы наиболее оптимально спланировать следующую поездку.

Например, если один из друзей Марселя посещает вершины с высотами 3 2 5 1 1 6 3 5, то наименьшая высота вершины, которую он не смог посетить, равна 4.

Формат входных данных

Первая строка содержит натуральные числа N и M ($1 \leq N \leq 300000$, $1 \leq M \leq 600000$). Во второй строке даны N целых чисел со значениями из интервала от 1 до N включительно, представляющие высоты вершин Верхоянского хребта. Далее следуют Q строк, содержащих по два числа $L[i]$ и $R[i]$, такие что $0 \leq L[i] \leq R[i] \leq N - 1$.

Формат выходных данных

Выведите Q строк, каждая из которых содержит наименьшее натуральное число — высоту вершины, которую i-й друг не посетит в эту поездку.

Система оценки

Данная задача содержит четыре подзадачи. Баллы за подзадачу начисляются, только если все тесты этой подзадачи пройдены.

Подзадача 1 (баллы: 20)

$N \leq 1000$ и $Q \leq 10000$.

Подзадача 2 (баллы: 30)

$N \leq 100000$ и $Q \leq 200000$, высоты вершин не превышают 50.

Подзадача 3 (баллы: 30)

$N \leq 100000$ и $Q \leq 200000$.

Подзадача 4 (баллы: 20)

$N \leq 300000$ и $Q \leq 600000$.

Примеры

стандартный ввод	стандартный вывод
14 16 3 4 3 2 5 1 6 7 2 1 6 2 4 3 0 4 0 5 9 13 9 12 3 12 2 12 2 11 1 11 3 13 5 13 8 13 0 7 0 8 6 8 6 9 6 10	1 6 5 3 3 8 4 8 8 5 5 8 8 1 3 3

Замечание

Следует отметить, что нумерация в массиве высот вершин начинается с 0.

E. Арифметические прогрессии

В этом учебном году юный программист Григорий узнал много нового по математике, в частности ему очень понравились арифметические прогрессии. Он не любит терять время впустую и поэтому решил исследовать понравившуюся ему тему. Григорий взял две арифметические прогрессии и хочет узнать, сколько совпадающих элементов лежат на промежутке от *l* до *r* включительно. Помогите ему решить поставленную задачу.

Напомним, что арифметическая прогрессия — это числовая последовательность вида

$$a_1,\ a_1 + b,\ a_1 + 2b,\ \ldots,\ a_1 + (n-1)b,\ \ldots,$$

то есть последовательность чисел, в которой каждое число, начиная со второго, получается из предыдущего добавлением к нему постоянного числа b (*шага*, или *разности* прогрессии):

$$a_n = a_{n-1} + b.$$

Любой (n-й) член прогрессии может быть вычислен по формуле общего члена:

$$a_n = a_1 + (n-1)b.$$

Формат входных данных

В первой строке входных данных находятся два целых числа a и b ($0 \le a, b \le 10^9$) — начальное значение и шаг первой арифметической прогрессии. Во второй строке находятся два целых числа c, d ($0 \le c, d \le 10^9$) — начальное значение и шаг второй арифметической прогрессии. В третьей строке находятся два целых числа l, r ($0 \le l \le r \le 10^9$) — начало и конец промежутка.

Формат выходных данных

Выведите одно число — количество совпадающих элементов двух арифметических прогрессий на отрезке $[l, r]$.

Примеры

стандартный ввод	стандартный вывод
3 2 5 6 10 20	2
30 0 15 15 10 1000	1

Замечание

Данная задача содержит четыре подзадачи. Баллы за подзадачу начисляются, только если все тесты этой и всех предыдущих подзадач пройдены.

Подзадача 1 (баллы: 20)

$a, b, c, d, l, r \leq 10^3$.

Подзадача 2 (баллы: 20)

$a, b, c, d, l, r \leq 10^6$.

Подзадача 3 (баллы: 40)

$a, b, c, d \leq 10^6$.

Подзадача 4 (баллы: 20)

Без дополнительных ограничений.

F. Супергерои

Уже много лет, скрытно от простых людей, идет война между командой супергероев и бандой суперзлодеев.

Каждый супергерой или суперзлодей обладает одной из пяти суперспособностей:

- (S) суперскорость — способность двигаться во много раз быстрее обычных людей;
- (P) суперсила — невероятная физическая сила и здоровье;
- (T) телепатия — способность читать и внушать другим людям мысли;
- (K) телекинез — способность перемещать предметы силой мысли;
- (E) энергетические лучи — способность стрелять в противников энергетическими лучами.

При столкновении двух обладателей суперспособностей результат определяется по следующим правилам:

- суперскорость побеждает телепатию и суперсилу;
- телепатия побеждает суперсилу и телекинез;
- суперсила побеждает телекинез и энергетические лучи;
- телекинез побеждает энергетические лучи и суперскорость;
- энергетические лучи побеждают суперскорость и телепатию.

Накануне финального сражения, супергероям стал известен состав суперзлодеев, которые примут в ней участие, с указанием типа их суперспособностей. Известно, что битвы такого рода состоят из ряда дуэльных поединков, причем каждый участник сражается ровно один раз. Количество супергероев в предстоящей битве равно количеству суперзлодеев.

Благодаря счастливому стечению обстоятельств, супергерои могут выбрать кто с кем будет сражаться. Помогите им сделать правильный выбор, так чтобы одержать победу в максимальном количестве дуэлей.

Формат входных данных

В первой строке входных данных задано целое число N ($1 \leq N \leq 10^5$) — количество участников битвы с одной стороны.

Во второй строке входных данных дана строка из N символов — типы суперсил, которыми обладают суперзлодеи.

Во третьей строке входных данных дана строка из N символов — типы суперсил, которыми обладают супергерои.

Формат выходных данных

Выведите строку из N символов — перестановку типов суперсил супергероев, обеспечивающую им победу в максимальном количестве дуэлей.

Каждый i-й символ в ней соответствует супергерою с таким типом суперспособностей, который должен вступить в схватку с суперзлодеем, суперспособность которого закодирована i-м символом второй строки входных данных.

Система оценки

Задача содержит две подзадачи. Баллы за каждый тест начисляются независимо друг от друга.

Подзадача 1 (баллы: 30)

В битве принимают участие герои и злодеи только с тремя разными способностями: S (суперскорость), P (суперсила) и K (телекинез).

Подзадача 2 (баллы: 40)

В битве принимают участие герои и злодеи со всеми пятью видами способностей, $N \leq 100$.

Подзадача 3 (баллы: 30)

В битве принимают участие герои и злодеи со всеми пятью видами способностей, $N \leq 10^5$.

Пример

стандартный ввод	стандартный вывод
5 STPKE STPKE	ESTPK

G. Задание Тыгына

Один из предков Марселя — Манчаары, сын Ньургуна и Сахайааны который в начале 17 века, был слугой Тыгын Дархана. Тыгын Дархан — великий вождь, который объединил якутские племена, кажется, очень увлекался теорией чисел.

Однажды он назначил N своим коровам попарно различные номера из интервала от 2 до M. Затем, используя эти номера, он решил сгруппировать их в по возможности минимальное количество стад. Однако, он установил определенные правила, которые в обязательном порядке должны соблюдаться для каждого стада:

1. Пусть x – наименьший номер коровы в стаде. Номера остальных коров стада должны иметь вид $x \times k$, где k – целое число.

2. Все делители числа k (кроме 1) должны быть больше или равны любому простому делителю числа x для каждой коровы в стаде.

Так как Тыгын был очень занят обеспечением мира на своей территории, он приказал своему слуге Манчаары разделить коров на стада согласно описанным правилам. Задача оказалась сложной для Манчаары, но вознаграждение было очень желанным: ему была обещана в жены прекрасная Сардаана!

Нам известен конец этой истории: Манчаары удалось сгруппировать коровы по правилам Тыгына, и прекрасная Сардаана стала его женой. Узнав эту историю, Марсель решил, что задача должна вернуться в места, где она была сформулирована. Вот поэтому участникам Туймаады–2019 предлагается решить ее!

Формат входных данных

Первая строка содержит целые числа N и M ($1 \leq N < M \leq 10^6$). Следующая строка содержит N попарно различных целых чисел — номера коров, значения которых лежат в интервале от 2 до M.

Формат выходных данных

Первая строка должна содержать минимальное количество стад, на которые Манчаары сможет разбить коров Тыгына.

Система оценки

Данная задача содержит две подзадачи. Баллы за подзадачу начисляются, только если все тесты этой подзадачи пройдены.

Подзадача 1 (баллы: 30)

$N \leq 1000$.

Подзадача 2 (баллы: 70)

Без дополнительных ограничений.

Примеры

стандартный ввод	стандартный вывод
5 100 2 3 11 22 12	3
10 20 8 12 20 6 3 7 11 13 19 15	9
14 20 4 14 3 5 7 11 13 17 19 12 15 16 8 9	9

H. Итоги спартакиады

$N + 1$ участников соревнуются в некоторой спартакиаде, которая длится N дней. Каждый день один из спортсменов признается победителем дня. Участникам присуждаются очки, равные количеству дней, в которых они становились победителями. В конце каждого дня участники с наибольшим количеством очков получают по монете. После спартакиады для каждого участника вычисляется показатель счастья по следующей схеме: каждый непрерывный интервал номеров дней, когда он по-

лучал монеты, добавляет к его показателю счастья квадрат длины этого интервала.

Предположим, некоторый участник получал монеты в следующие дни: 3, 4, 10, 11, 12, 18 и 19. Непрерывные интервалы для него — [3–4], [10–12], [18–19]. Тогда показатель счастья для этого участника будет равен $2^2 + 3^2 + 2^2 = 4 + 9 + 4 = 17$. Итог спартакиады вычисляется как сумма показателей счастья всех участников.

Марсель в результаты спартакиады может добавить один день, в котором победителем назначается участник под номером 0.

Дан массив N целых чисел, значения которых лежат между 0 и N — номера победителей каждого дня. Пусть $f(p)$ — итог спартакиады в случае, если Марсель вставит 0 в этот массив после p-го элемента. Распечатайте числа $f(0), f(1), ..., f(N)$.

К примеру, если массив содержит числа 0 1 1, то $f(0)$ — это итог спартакиады с результатами **0** 0 1 1. Участник под номером 0 в этом случае получает монеты в 1, 2, 3 и 4 дни, следовательно, его показатель счастья $4^2 = 16$, участник под номером 1 получает монету только в 4 день, поэтому его показатель счастья $1^2 = 1$. Участники 2 и 3 не получают монет. Итог этой спартакиады $f(0) = 17$. Вычислим $f(3)$, результаты этой спартакиады — 0 1 1 **0**. Участник под номером 0 получает монеты в дни 1, 2 и 4, тогда его показатель счастья равен $4 + 1^2 = 5$. Участник под номером 1 получает монеты во 2, 3, 4 дни и его показатель счастья равен 9. Итог спартакиады $f(3) = 14$.

Формат входных данных

Первая строка содержит число $N (1 \leq N \leq 10^6)$, а в следующей даны N целых чисел от 0 до N: номера победителей каждого дня.

Формат выходных данных

Программа должна вывести $N + 1$ строк, каждая i-я строка содержит значение $f(i - 1)$.

Система оценки

Данная задача содержит четыре подзадачи. Баллы за подзадачу начисляются, только если все тесты этой и всех предыдущих подзадач пройдены.

Подзадача 1 (баллы: 11)

$N \leq 100$.

Подзадача 2 (баллы: 13)

$N \leq 3000$.

Подзадача 3 (баллы: 39)

$N \leq 10^5$.

Подзадача 4 (баллы: 37)

$N \leq 10^6$.

Примеры

стандартный ввод	стандартный вывод
4 0 4 4 4	20 20 21 21 20
4 1 0 1 1	21 17 17 27 26
4 2 1 1 0	23 23 27 21 21
10 1 2 3 1 2 3 0 1 2 3	140 154 154 154 145 152 157 157 144 149 152

стандартный ввод	стандартный вывод
10 4 9 10 1 5 3 4 10 3 0	218 226 226 226 226 226 226 190 197 202 202

Решения

A. Коллекция

Расставить N камней по N отсекам подставки можно $N!$ способами. Если учитывать, что каждый отсек подставки подсвечивается, то количество разных способов расставить камни с разной подсветкой будет равна $2^N \times N!$. Вычислять напрямую это число затруднительно, поэтому воспользуемся рекуррентной формулой $a_N = a_{N-1} \times 2 \times N$, при этом $a_0 = 1$. Тогда задача сводится к нахождению N-го элемента этой последовательности. Так как по заданию необходимо найти остаток от деления на $10^9 + 7$, то следует применить следующую формулу для нахождения N-го члена: $a_N = a_{N-1} \times 2 \times N \bmod (10^9 + 7)$.

B. Рис на шахматной доске

Подзадача 1 может быть решена путем подсчета суммы чисел на клетках (r, c), где $r+c$ четно, $r+c$ нечетно, и выбора максимальной из этих двух сумм. Поскольку числа $a_{i,j}$ могут быть порядка 10^9, то их сумма может достигать значения $\frac{1}{2} \cdot n \cdot m \cdot 10^9 \approx 10^{12}$, что превосходит максимальное значение 32-битного типа `int`. Нужно не забывать использовать 64-битный тип для подсчета.

Для $k > 1$, каждая почти шахматная раскраска может быть охарактеризована размером и цветом самого левого верхнего прямоугольника в раскраске. Любой такой прямоугольник имеет размер $h \times w$, где h и w не превосходят k. После того, как цвет этого прямоугольника, а также h и w зафиксированы, цвет всех остальных клеток вычисляется однозначно. Если перебрать размер, и затем просуммировать все клетки наивно, то решение работает за $O(nmk^2)$ и проходит первые две подгруппы.

Осталось научиться быстро считать сумму для фиксированной раскраски. Это можно сделать с помощью двумерных частичных сумм. Посчитаем значение $sum(i,j) = \sum_{r \le i, c \le j} a_{r,c}$ — сумма на подпрямоугольнике $(1,1)$ – (i,j). С помощью такого массива, используя формулу включений-исключений, можно найти сумму произвольного прямоугольника (r_1, c_1) – (r_2, c_2): $sum(r_2, c_2) - sum(r_1 - 1, c_2) - sum(r_2, c_1 - 1) + sum(r_1 - 1, c_1 - 1)$.

Теперь, поскольку в любой почти шахматной раскраске не больше

$(\lfloor \frac{n}{k} \rfloor + 2)(\lfloor \frac{m}{k} \rfloor + 2)$ одноцветных прямоугольников, а сумму в каждом из них можно найти за $O(1)$, то сумму для фиксированной раскраски мы считаем за $O(\frac{nm}{k^2})$. Поскольку всего разных раскрасок $2k^2$, суммарная сложность решения равна $O(k^2) \cdot O(\frac{nm}{k^2}) = O(nm)$. Аккуратная реализация такого решения получает 100 баллов.

C. Двоичное разбиение

Подзадачу 1 с $|S| \leq 16$ можно было решить полным перебором расстановок знаков **+**. Для строки длины n существует 2^{n-1} расстановок плюсов (между каждой парой соседних цифр можно либо поставить плюс, либо нет). Далее нужно просуммировать числа в двоичной системе, и проверить, является ли результат степенью двойки. Эту проверку можно сделать одной битовой операцией: x является степенью двойки тогда и только тогда, когда $x > 0 \wedge x \ \ \mathbf{and} \ \ (x - 1) = 0$.

Для решения подзадачи 2 можно сделать следующее предположение: пусть степень двойки, которая получится в итоге, будет небольшой (до 2^l). Тогда можно реализовать решение с помощью динамического программирования: пусть $dp[i][j]$ равно 1, если используя первые i символов можно набрать сумму j, и 0 иначе. Переход производится за $O(l)$: перебираем длину нового числа (не больше, чем l), и суммарно решение работает за $O(nl2^l)$. Если выбрать l так, что $2^l \approx n$, то получится время работы $O(n^2 \log n)$, что проходит по времени вторую подзадачу.

Такое решение можно реализовать, и проверить, что оно находит ответ на всех строках, где есть хотя бы одна 1. Действительно, можно доказать, что на всех таких строках ответ существует, более того, на всех, кроме **11111**, длина строки в разбиении не превосходит 3.

Расставим изначально знаки **+** между всеми парами соседних цифр. Текущая сумма равна m, где m — это количество единиц в изначальной строке. Пусть $2^{k-1} < m \leq 2^k$. Теперь путем склейки соседних чисел в одно увеличим ответ до 2^k. Заметим, что склеивание $1 + x \to 1x$ всегда увеличивает сумму на 1, $1 + 0 + x \to 10x$ — на 3, а $1 + 1 + x \to 11x$ — на 4. Будем действовать жадно, проводить склеивание троек цифр, пока сумма не превосходит 2^k. После этого сделаем то же самое, только с парами соседних цифр. Можно показать, что при достаточно большом количестве единиц (16 единиц достаточно), такой алгоритм всегда полу-

чит сумму 2^k. Для меньших значений m можно проверить, что алгоритм не работает только при $m = 5$.

Разберем случай $m = 5$ отдельно. Если есть подстрока $10x$, то нужно выделить ее в отдельное число, и получится сумма 8. Иначе, наша строка выглядит как $0 \ldots 011111$ или $0 \ldots 0111110$. В первом случае ответ $1111 + 1$, а во втором — $11 + 11 + 10$. Такую жадную стратегию и разбор случаев можно реализовать за линейное время, что дает полный балл за задачу.

D. Верхоянский хребет

Первая подзадача. Для решения первой подзадачи достаточно реализовать переборный алгоритм. Будем обрабатывать запросы по мере считывания. Заведем булевый массив *visited* и будем двигаться слева направо интервала запроса и отмечать все вершины из заданного интервала как посещенные. Инициализируем переменную $ans = 1$ и будем увеличивать его пока $visited[ans]$ истинно. Не стоит забывать сбрасывать значения массива *visited*, для того чтобы корректно обрабатывать следующий запрос.

Вторая и третья подзадачи. Разделим массив высот вершин на смежные подмассивы длины K. Всего будет $\frac{N}{K}$ подмассивов. N не обязательно должно делиться на K, при этом последний подмассив может иметь меньшую длину.

Запросы считаем полностью в память. Для каждого подмассива рассмотрим все запросы, левая граница которых лежит внутри рассматриваемого подмассива и вычислим ответы на все запросы. Как только мы рассмотрим все подмассивы мы найдем ответы на запросы и сможем их распечатать в нужном порядке.

Предположим, что мы хотим вычислить ответы на запросы, левая граница которых принадлежит некоторому подмассиву B. Их правые границы могут быть достаточно большими, но левые границы принадлежат интервалу от $B * K$ до $(B + 1) * K - 1$. Назовем вершины из этого интервала специальными.

Существует не более K специальных вершин и $K+1$ смежных интервалов не специальных вершин. К примеру, если специальными вершинами являются вершины 5 и 23, и $N = 100$, интервалы не специальных вершин:

$[1-4]$, $[6-22]$ и $[24-100]$.

Теперь, для каждого запроса не специальная вершина может быть посещена только вне подмассива, когда как специальная вершина может быть посещена внутри нее. Идея решения заключается в вычислении и хранении частичных mex для каждого интервала не специальных высот вершин, рассматривая вершины только от $(B+1)*K$ до правой границы запроса.

Более точно, в $partialMex[x]$ будем хранить минимальное значение большее или равное x, которое не может быть найдено в интервале от $(B+1)*K$ до правой границы текущего запроса.

Важно отметить, что достаточно вычислить $partialMex[x]$ только для значений x, на которых интервалы не специальных вершин начинается. Для вышеприведенного примера $partialMex[x]$ будет вычисляться только для $x = 1$, 6 или 24.

Для правильного хранения этих данных отсортируем запросы (для рассматриваемого подмассива) по правой границе. Правые границы первых запросов будут принадлежать подмассиву, так что мы можем просто пройтись по интервалу запроса и вычислять их mex перебором, что в худшем случае займет K шагов.

Последующие запросы будут увеличивать правую границу, и мы будем отмечать новые вершины как посещенные. Заметим, что мы можем отмечать только высоты вершин вне рассматриваемого подмассива, а также мы отмечаем все вершины не зависимо от того является специальным или нет.

Для каждого запроса ответ вычисляем следующим образом: инициализируем $ans = 1$ и попробуем ее увеличить. На каждом шаге ans представляет высоту, которая является либо специальной, либо нет. Если она специальная, то увеличиваем ans, если она находится в подмассиве (после левой границы и до правой) или отмечена как посещенная. Иначе выходим из цикла и возвращаем ans. Если частичный mex говорит о том, что весь интервал был рассмотрен мы можем продолжить рассматривать ans с $1 + right - end - of - the - interval$. Если мы застряли по середине интервала, тогда частичный mex является ответом запроса. Таким образом, мы точно используем все высоты в интервале запроса.

Эталонное решение вычисляет $partialMex[x]$ лениво (не отмечаем как посещенные, получаем его значение, когда нам необходим ответ на за-

прос). Когда мы получаем какой-то *ans*, который равен первой не специальной вершине, мы пытаемся увеличивать *partialMex*[*ans*] пока она остается в интервале не специальных вершин, и она посещенная вне подмассива (отмечена в массиве *visited*). Тогда мы можем сказать, что $ans = partialMex[ans]$ и продолжать.

Для того, чтобы подвести итог, для каждого подмассива, по мере движения вправо, и отвечая на запросы, мы анализируем частичные mex каждого интервала не специальных вершин. Это дает нам возможность для каждого запроса мы можем рассмотреть максимум $2 * K + 1$ значений: специальные вершины и $K + 1$ интервалов, которые мы рассматриваем за константное время, т.к. частичные mex, которые мы храним для каждого из них. Также частичные mex будут увеличиваться только $interval - lenght$ раз, в результате в итоге получаем максимум N шагов для каждого подмассива.

Проанализируем сложность алгоритма. Для каждого подмножества мы посещаем $O(N)$ позиций, т.к. наибольшее значение правая граница равна $N - 1$. Мы так же увеличиваем *partialMex* $O(N)$ раз. Для каждого запроса мы совершаем $O(K)$ дополнительных шага. Следовательно, общая сложность алгоритма $O(\frac{N*N}{K} + Q * K)$, которая достигает минимума когда K равно $\frac{N}{\sqrt{Q}}$, которая в итоге дает сложность $O(N * \sqrt{Q} + Q)$.

Однако, решение требует сортировки запросов по правой границе для каждого подмассива, что в итоге нам дает сложность $O(N * \sqrt{Q} + Q * logQ)$.

Полное решение. Будем обрабатывать запросы по мере увеличения их правой границы. когда мы движемся по массиву высот и отвечаем на запросы, нам необходимо запоминать для каждой вершины, какая из них наиболее правая, которую мы рассматривали в массиве. Предположим мы знаем эти значения. Тогда для каждого запроса нам нужно выяснить наибольшее значение *val*, такое что все значения от 1 до $val - 1$ больше или равно левой границы запроса.

Мы можем использовать дерево отрезков по этим значениям. Предположим, мы храним этот массив, *rightMostPosition*[x] со значениями наиболее правой позиции, где высота равна x, рассмотренная к данному моменту (до правой границы текущего запроса). *rightMostPosition*$[x] = -1$ если мы не нашли вершины равной x пока. Дерево отрезков будет хранить минимальное значение этого массива.

Когда мы поменяем правую границу текущего запроса, мы просто можем заменить значения в массиве. Каждая замена в массиве означает $O(logN)$ изменений в дереве отрезков. Мы знаем как изменить дерево отрезков, но как найти mex используя его?

Используем алгоритм бинарного поиска по дереву отрезков для поиска ответа. Каждый узел дерева показывает, все ли значения узла принадлежат интервалу запроса. Условие таково: $segmentTree[node]$ должно быть больше или равно левой границе запроса. Используя данное условие, мы можем искать ответ в дереве отрезков за $O(\log N)$.

Итоговая сложность данного решения $O((N + Q) * \log N)$.

E. Арифметические прогрессии

Случай, когда хотя бы в одной из прогрессий шаг равен 0, рассматривается отдельно. Данные случаи легко разбираются.

Решение подзадачи 1

Для решения данной задачи достаточно полного перебора. Для первой прогрессии пройдемся по всем элементам, которые меньше r, сохраняя значения, которые лежат на промежутке $[l, r]$. И для второй прогрессии сделаем те же самые действия. Простым подсчетом совпадающих элементов полученных множеств, находим ответ на задачу. Данный способ работает за $O(r^2)$.

Решение подзадачи 2

Проходясь по всем числам от l до r проверяем, является ли текущее число элементом обеих прогрессий. Рассмотрим, как проверить число на принадлежность к прогрессии. Пусть число будет k, а начальное значение и шаг прогрессии — a и b соответственно. Если выполняется условие

$$k = a + bn$$

для какого-нибудь неотрицательного целого n, то число k является элементом данной прогрессии. Такое число n существует тогда и только тогда, когда $a \leq k$ и $(k - a)$ кратно b.

Данное решение работает за $O(r)$ времени.

Решение подзадачи 3, 4

Для решения данной подзадачи найдем наименьшее число, которое является общим для обеих прогрессий. Надо решить следующее уравне-

ние

$$a + bn = c + dm.$$

Данное выражение является линейным диофантовым уравнением. Есть два способа решить это уравнение. Первый способ потребует $O(\max(b, d))$ времени. Преобразуем уравнение и приведем к следующему виду

$$n = \frac{c - a + dm}{b}.$$

Предположим, что $a \leq c$ (если не так, то можем поменять прогрессии местами). Достаточно перебрать значения для m от 0 до b. Если решения есть, то есть решение и в этом интервале. Если в этом интервале нет решения, то не существует такого n, который удовлетворяет уравнению. Такое нахождение решения займет $O(\max(b, d))$. Если решить данное уравнение с помощью расширенного алгоритма Евклида, то нахождение решения займет $O(1)$, что дает возможность решить подзадачу 4.

Пусть t будет наименьшим числом, которое будет общим для обеих прогрессий. Тогда можно заметить, что общие числа будут представлять из себя арифметическую прогрессию с начальным значением t и с шагом (b, d), где НОК — это функция нахождения наименьшего общего кратного.

Определим функцию $f(a, b, p)$ поиска количества элементов арифметической прогрессии с начальным значением a и с шагом b на отрезке от 0 до p:

$$f(a, b, p) = \begin{cases} 0 & \text{, если } p < a \\ \lfloor \frac{p-a}{b} \rfloor + 1 & \text{, иначе} \end{cases}$$

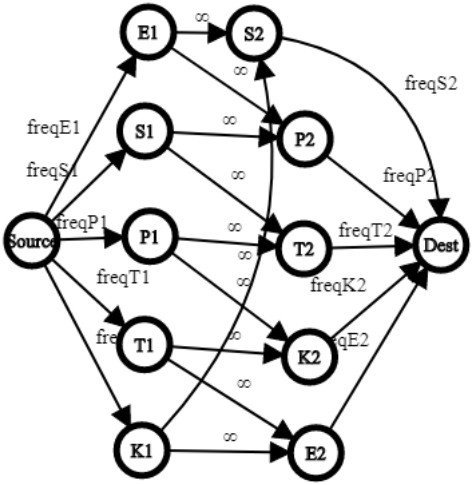
Тогда ответом будет $f(t, (b, d), r) - f(t, (b, d), l - 1)$

F. Супергерои

Построим ориентированный двудольный граф 5×5. Пусть пять вершин с левой стороны будут соответствовать суперспособностям супергероев, а пять вершин с правой стороны — суперспособностям суперзлодеев. Каждой вершине слева назначим число — количество супергероев с этой суперспособностью. Сделаем то же самое справа для суперзлодеев. Наконец, создадим направленные ребра из каждой вершины слева согласно правилам определения победителя дуэлей.

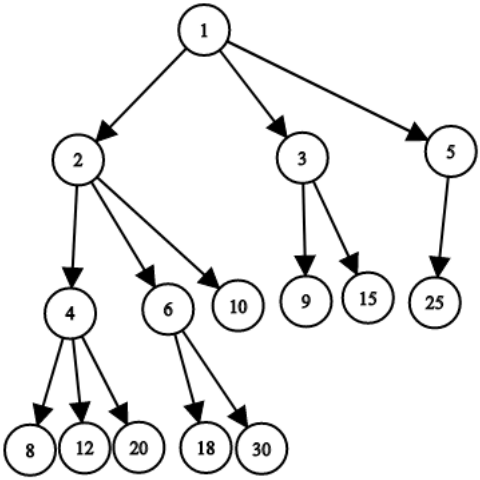
Существует несколько методов решения этой задачи.

Например, можно рассмотреть задачу максимизации потока. С этой точки зрения источниками будут служить вершины с супергероями, а стоками — вершины с суперзлодеями (с тем же успехом можно рассмотреть зеркальную задачу). После нахождения максимального потока он покажет, какие супергерои с какими злодеями должны сражаться. Оставшиеся нераспределенными супергерои могут сражаться с любыми нераспределенными суперзлодеями.



G. Задание Тыгына

Рассмотрим особое дерево, которое назовем *деревом делителей*. Его узлы помечены целыми числами, в корне его находится единица, а родителем каждого узла $n > 1$ является n/p, где p — максимальный простой делитель n. Например, родителем числа $525 = 3 \cdot 5 \cdot 5 \cdot 7$ является число $75 = 3 \cdot 5 \cdot 5$, а родителем $8 = 2 \cdot 2 \cdot 2$ — число $2 \cdot 2$.



На рисунке приведена часть дерева делителей. Заметим, что все простые числа являются детьми узла 1, а высота каждого узла равна числу множителей в разложении соответствующего числа в произведение простых (или сумме показателей простых делителей в этом разложении).

Стадо, собранное из всех коров, находящихся в поддереве узла, соответствующего некоторой корове, удовлетворяет требованиям Тыгына. Создав такое стадо для каждой коровы, не имеющей предка в дереве, и отнеся каждую из всех остальных коров к стаду своего самого отдаленного предка, мы получим решение задачи.

Возможны разные способы получения решения. Для поиска по дереву нам нужно будет знать максимальный простой делитель для каждого узла. Решение жюри использует поиск в ширину, сохраняя для каждого посещаемого узла стадо, которому он принадлежит. При чтении входных данных отмечаем коров в булевском массиве.

Для решения также требуется знание всех простых чисел M. Можно использовать решето Эратосфена или линейный отсев. Сложность такого решения равна $O(N + M\log(\log M))$ или $O(N + M)$ в зависимости от реализации решета.

H. Итоги спартакиады

Обозначим через $I(p)$ соревнование, которое получается в результате вставки победы участника номер 0 в список между днями p и $(p+1)$. Обозначим через $f(p)$ общий суммарный показатель счастья для $I(p)$.

За 11 баллов. Ищем независимо все $N+1$ значений $f(p)$. Будем вести список L людей, получивших монету в текущий день, то есть имеющих к этому дню максимальное число побед, которое обозначим M. Этот список обновляется так. Возможны случаи:

1) Победителем дня стал некоторый $x \in L$. Тогда его число побед — новый максимум, $M+1$, и мы удаляем из L всех остальных.

2а) Победителем дня стал некоторый $x \notin L$, новое число побед которого сравнялось с M. Тогда мы просто добавляем x к L.

2б) Победителем дня стал некоторый $x \notin L$, который все еще имеет число побед, меньшее максимального. L не меняется.

Рассмотрим добавленные за этот день каждым участником L показатели счастья. Поскольку $a^2 = (a-1)^2 + 2a - 1$, то в a-й день к показателю вчерашнего дня добавляется $2a-1$. Будем хранить в массиве $start[x]$ для каждого $x \in L$ день, когда x появился в L (в последний раз снова вошел в список, если x появлялся в L несколько раз). Рассматривая случаи, видим, что обновление этого массива в любой день выполняется за константное время $O(1)$. Суммарный текущий показатель счастья возрастает при переходе ко дню day на $2 * (day - start[x] + 1) - 1$ для каждого $x \in L$.

Это решение имеет сложность $O(N^3)$, так как мы вычисляем суммарный показатели счастья для $I(0), \ldots I(N)$. Для каждого вычисления требуется пройтись по $O(N)$ дням, и в каждый день надо обработать всех людей в списке L, который может содержать до $O(N)$ человек.

Заметим, что существуют более простые решения за 11 баллов, но от такого легче перейти к следующему пункту.

За 24 балла. Для заданного соревнования $I(p)$ и дня, вместо поочередного добавления к показателю счастья слагаемых вида $2 * (day - start[x] + 1) - 1$ для каждого $x \in L$ будем запоминать значение этой суммы sum и длину списка $size$. При этом нам даже нет необходимости хранить сам список. Если x победитель очередного дня, то в

соответствии с разобранными выше случаями
1) $sum = day - start[x]; size = 1$;
2а) $start[x] = day; size = size + 1$;
2б) ничего не меняется.

Затем прибавляем *size* к *sum*. Наконец, для подсчета суммарного показателя за день, прибавляем к ответу $2 * sum - size$.

Очевидно, сложность такого решения равна $O(N^2)$, поскольку мы попрежнему рассматриваем все соревнования $I(0)$ ...$I(p)$ по отдельности и обрабатываем $O(N)$ дней до каждого, выполняя какие-то $O(1)$ операции.

За 100 баллов. Получению решения за $O(N)$ препятствует то, что мы вычисляем каждое $f(p)$ независимо. Можно заметить, что соревнование $I(p)$ до дня p включительно совпадает с $I(N)$, а начиная с дня $p + 1$ и до $N + 1$ включительно — с $I(0)$. Таким образом в наших вычислениях мы можем переключиться с $I(N)$ на $I(0)$ начиная с дня $p + 1$, так как на вручение монет влияет только количество побед к настоящему дню, а не момент вставки фальшивой победы. Хотя это позволяет свести дело к рассмотрению только соревнований $I(0)$ и $I(N)$, этого недостаточно для эффективного вычисления $f(p)$. Рассмотрим день p в соревновании $I(0)$. Пусть после него происходит только случай 2, вплоть до некоторого дня, в котором происходит случай 1. Этот день назовем переломным и обозначим через $t(p)$ — первый день строго после p в $I(0)$, где появляется новый максимум. Распределение монет в $I(p)$, для всех кроме участника 0, совпадает с распределением в $I(N)$ для всех дней до $t(p)$, а с $t(p)$ и до N оно совпадает с распределением в $I(0)$. Число выигрышей нулевого игрока не влияет на получение монет другими участниками до появления случая 1. Это наконец позволяет вычислять $f(p)$ за $O(1)$ так как список L после дня $t(p)$ состоит только из одного участника. Для такого вычисления нам нужны инкременты частичных сумм для каждого дня в $I(0)$ и $I(N)$. Также нам необходимы некоторые данные о днях типа 1 в $I(0)$: номер дня, «выживающий» в L участник, левый и правый концы отрезка, где он получает монеты. Все $t(p)$ можно найти за $O(N)$. Наше утверждение исключает из рассмотрения участника номер 0, так что для него нужно будет проводить вычисления отдельно. Он не входит в расчет частичных сумм. Итак, мы можем находить $f(p)$ за $O(1)$. Сначала мы сложим все частичные суммы показателей счастья до переломного дня $t(p)$ в $I(N)$, и частичные суммы показателей счастья от дня $t(p)$ до N в $I(0)$. Остается скоррек-

тировать вычисления для отрезка выжившего, поскольку, если это не 0, частичные суммы дают неверное значение показателя счастья, которое мы исправляем при помощи предвычисленных для каждой переломной точки данных. Все еще остается рассмотреть показатель счастья для 0, но это может быть сделано эффективно аналогично, с использованием частичных сумм и отрезков получения монет. Окончательно сложность решения $O(N)$, поскольку все предвычисления об $I(0)$ и $I(N)$ могут быть проведены за $O(N)$, а вычисление каждого из $N + 1$ чисел — за $O(1)$.

ИТОГИ / FINAL STANDINGS

	Name	Country	Первый день / First day					Второй день / Second day					ИТОГО TOTAL	НАГРАДА AWARD
			A	B	C	D	Σ	E	F	G	H	Σ		
1	Tinca Matei	Romania	100	100	60	100	*360*	100	100	100	24	*324*	**684**	1DG
2	Mușat Tiberiu-Ioan	Romania	100	100	25	100	*325*	100	100	100	24	*324*	**649**	1DG
3	Wong Swee Chong, Dave	Singapore	100	100	25	100	*325*	100	100	100	24	*324*	**649**	1DG
4	Popovici Robert-Adrian	Romania	100	100	25	80	*305*	100	95	100	24	*319*	**624**	2DG
5	Seritan Luca	Romania	100	100	60	100	*360*	100	100	30	24	*254*	**614**	2DG
6	Donciu Mircea	Romania	100	70	0	80	*250*	100	100	100	24	*324*	**574**	2DG
7	Tursynbay Dinmukhamed	Kazakhstan	100	50	25	80	*255*	100	93	100	11	*304*	**559**	2DG
8	Bulatov Vasilii	Russia	100	100	25	50	*275*	100	100	.	11	*211*	**486**	3DG
9	Semenov Nikita	Russia	100	100	25	50	*275*	100	30	30	11	*171*	**446**	3DG
10	Stepanov Dmitrii	Russia	100	70	0	20	*190*	40	100	100	.	*240*	**430**	3DG
11	Zykov Timur	Russia	100	50	.	20	*170*	40	97	.	.	*137*	**307**	3DG
12	Petrov Petar Georgiev	Bulgaria	100	70	25	20	*215*	0	30	30	.	*60*	**275**	HM
13	Tsypandin Nikolai	Russia	100	30	0	0	*130*	40	100	0	.	*140*	**270**	HM
14	Daneshara Alireza Hossein	Iran	100	30	0	20	*150*	100	.	.	11	*111*	**261**	HM

15	Mukhametkarim Kaiyrly	Kazakhstan	100	30	25	20	*175*	40	10	30		*80*	**255**	
16	Petrova Margarita	Russia	100	50	25	0	*175*	40	0	30	.	*70*	**245**	SA
17	Krylykov Maksim	Russia	100	30	0	20	*150*	20	70	0	0	*90*	**240**	
18	Murgoci Vlad	Romania	100	30	.	20	*150*	40	14	30	.	*84*	**234**	
19	Badanov Chingis	Russia	100	30	.	20	*150*	0	44	30	.	*74*	**224**	
20	Zakharov Aital	Russia	100	50	.	0	*150*	40	.	30	0	*70*	**220**	SA
21	Knyazhev Aleksey	Russia	100	0	0	0	*100*	0	30	30	.	*60*	**160**	
22	Mohammadi Mohammadi Fatemeh Hadi	Iran	100	30	.	0	*130*	0	30	0	.	*30*	**160**	SA
23	Pavlovskii Vladislav	Russia	100	0	.	0	*100*	0	5	.	.	*5*	**105**	
24	Falsafi Arian Alireza	Iran	40	.	.	0	*40*	0	51	.	.	*51*	**91**	

13.07.2019

НАГРАДЫ / AWARDS: 1DG — диплом 1 степени / 1st degree diploma,
2DG — диплом 2 степени / 2nd degree diploma,
3DG — диплом 3 степени / 3rd degree diploma,
HM — почетная грамота / Honorable mention,
SA — спецприз / special award

Содержание | Contents