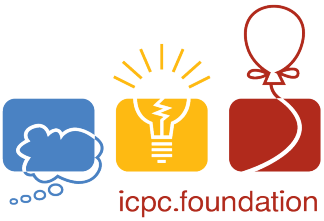


regionals | latin america region

icpc international collegiate programming contest

International Collegiate Programming Contest

2020

Latin American Regional Contests

July 10, 2021

Contest Session

This problem set contains 14 problems; pages are numbered from 1 to 21.

This problem set is used in simultaneous contests hosted in the following countries:

Argentina, Bolivia, Brasil, Chile, Colombia, Costa Rica, Cuba, Ecuador, El Salvador, Guatemala, Jamaica, México, Nicaragua, Perú, Puerto Rico, República Dominicana, Trinidad y Tobago and Venezuela

General information

Unless otherwise stated, the following conditions hold for all problems.

Program name

1. Your solution must be called *codename*`.c`, *codename*`.cpp`, *codename*`.java`, *codename*`.kt`, *codename*`.py3`, where *codename* is the capital letter which identifies the problem.

Input

1. The input must be read from standard input.
2. The input consists of a single test case, which is described using a number of lines that depends on the problem. No extra data appear in the input.
3. When a line of data contains several values, they are separated by *single* spaces. No other spaces appear in the input. There are no empty lines.
4. The English alphabet is used. There are no letters with tildes, accents, diaereses or other diacritical marks (ñ, Ã, é, Ì, ô, Ü, ç, etcetera).
5. Every line, including the last one, has the usual end-of-line mark.

Output

1. The output must be written to standard output.
2. The result of the test case must appear in the output using a number of lines that depends on the problem. No extra data should appear in the output.
3. When a line of results contains several values, they must be separated by *single* spaces. No other spaces should appear in the output. There should be no empty lines.
4. The English alphabet must be used. There should be no letters with tildes, accents, diaereses or other diacritical marks (ñ, Ã, é, Ì, ô, Ü, ç, etcetera).
5. Every line, including the last one, must have the usual end-of-line mark.
6. To output real numbers, round them to the closest rational with the required number of digits after the decimal point. Test case is such that there are no ties when rounding as specified.

Problem A – Almost Origami

You have a rectangular sheet of paper of height 1 and you want to locate any point at height H measured from the bottom border of the sheet. Since you do not know Haga's theorems, you plan to repeat the following step. Assume you already located a point P_L at height L on the left border of the sheet, and a point P_R at height R on the right border of the sheet. Then you draw a line from the lower left corner of the sheet to P_R, and another line from the lower right corner of the sheet to P_L. If the crossing point is at height H, then you are done. Otherwise you draw a horizontal line that passes through the crossing point and go for another step.

As an example, consider the case $H = 1/3$. During the first step, the only possibility is choosing the upper corners of the sheet (that is, $L = R = 1$). So you draw the two diagonals of the sheet, and the crossing point is at height $1/2$. Since $H \neq 1/2$, you draw a horizontal line that passes through the crossing point. This line provides two new points with known height $1/2$ on the borders of the sheet, one on the left border and the other one on the right border. For the second step you can choose between using the original known points at height 1, or the points you have just located at height $1/2$. That is, you can choose either $L = 1$ or $L = 1/2$ and of course $R = 1$ or $R = 1/2$. It is easy to see that if you choose $L = R = 1/2$, then the crossing point would be at height $1/4$. However, if you choose $L = 1/2$ and $R = 1$, then the crossing point would be at the desired height $H = 1/3$. By symmetry, the same occurs if you choose $L = 1$ and $R = 1/2$.

Given a rational height H, you must determine a shortest sequence of heights on the borders of the sheet that allows locating a point at height H.

As the above example shows, only a point at height $1/2$ can be located in a single step, and so a possible shortest sequence for $H = 1/3$ is $S = (1, 1, 1/2, 1)$. The first two heights must be chosen during the first step, and the remaining two heights must be chosen during the second step.

Input

The input consists of a single line that contains two integers M and N ($1 \leq M < N \leq 100$) such that $H = M/N$ is an irreducible fraction.

Output

Output a single line with the character "*" (asterisk) if a point at height H cannot be located by means of the described procedure. Otherwise, output a shortest sequence of heights $S_1, S_2, \ldots, S_K$ that allows locating a point at height H, if they are chosen in the order they appear in the sequence. Height S_i must be written in the i-th line using two integers A_i and B_i such that $S_i = A_i/B_i$ is an irreducible fraction ($i = 1, 2, \ldots, K$). It is guaranteed that when a point at height H can be located, it can be optimally located choosing only rational heights.

Sample input 1	**Sample output 1**
1 3	1 1 1 1 1 2 1 1

Sample input 2	**Sample output 2**
1 3	1 1 1 1 1 1 1 2

Sample input 3	**Sample output 3**
3 4	*

Sample input 4	**Sample output 4**
1 4	1 1 1 1 1 2 1 2

Problem B – Beautiful Mountains

A subarray of an array is a contiguous portion of the array. A partition of an array into subarrays is a collection of subarrays that cover all the array without overlaps (each element in the array belongs to exactly one subarray). For instance, if $A = [3, 1, 4, 1, 5]$ is an array, $[3, 1, 4]$ and $[1, 5]$ form a partition of A into subarrays, while $[3, 4, 5]$ is *not* a subarray of A.

Those are standard definitions that you may have read elsewhere. So, what's new here? Well, a few more definitions follow.

Given an array A of integers, a subarray $[A_i, A_{i+1}, \ldots, A_j]$ of A is called a mountain if there exists an index k such that $i < k < j$, the subarray from A_i to A_k is non-decreasing, and the subarray from A_k to A_j is non-increasing. In simple words, the values in the subarray "go up" until index k and then "go down", resembling a mountain. Note that a subarray with fewer than three elements cannot be a mountain.

An array of integers is called a beautiful mountain chain if it can be partitioned into mountains, each of them having the same number of elements, except for the last mountain which may have fewer elements.

As an example, $[5, 10, 4, 1, 3, 2]$ is a beautiful mountain chain because it can be partitioned into $[5, 10, 4]$ and $[1, 3, 2]$, being both mountains that have the same number of elements. Another example is the array $[5, 10, 4, 4, 10, 20, 30, 20, 2, 3, 1]$, which is also a beautiful mountain chain because it can be partitioned into $[5, 10, 4, 4]$, $[10, 20, 30, 20]$ and $[2, 3, 1]$.

Given an array of positive integers, where some values can be missing, determine if it is possible to complete the array with positive integers such that it becomes a beautiful mountain chain.

Input

The first line contains an integer N ($3 \leq N \leq 10^5$) indicating the number of elements in the array. The second line contains N integers $A_1, A_2, \ldots, A_N$ ($A_i = -1$ or $1 \leq A_i \leq 10^9$ for $i = 1, 2, \ldots, N$), where $A_i = -1$ indicates that the i-th element of the array needs to be determined, and a positive value is the actual i-th element of the array.

Output

Output a single line with the uppercase letter "`Y`" if it is possible to complete the array with positive integers such that it becomes a beautiful mountain chain, and the uppercase letter "`N`" otherwise.

Sample input 1	**Sample output 1**
6 5 10 4 1 3 2	Y

Sample input 2	**Sample output 2**
11 5 10 4 -1 10 20 30 20 2 3 -1	Y

Sample input 3	**Sample output 3**
12 1 3 2 5 -1 8 9 -1 7 -1 4 5	N

Problem C – Crisis at the Wedding

A famous football player just got married and is holding a party for his wedding guests. The guests are seated at tables around a circular pond in the garden of the player's villa. Each table accommodates exactly the same number of guests, and consecutive tables around the pond are at a unit distance.

At the moment of the traditional Best Man toast a crisis erupted: although the total number of champagne glasses in the guests' tables is exactly the number of guests, the glasses could have been distributed unevenly over the tables, with some tables having more glasses than guests and some other tables having fewer glasses than guests.

A single waiter is available to fix the glasses distribution, collecting surplus glasses from tables and delivering them to tables needing glasses. The cost of each glass fix is the distance the waiter carries the glass until he delivers it to a table. The total cost for the operation is the sum of the costs for all glasses. The waiter can start at any table, but the player is superstitious and will only allow the waiter to walk in a strict clockwise or counterclockwise direction when fixing the glasses distribution. That is, once the waiter starts in one direction (clockwise or counterclockwise) he cannot change the direction.

Earn an autographed jersey from the football player by helping him to calculate the smallest possible total cost for fixing the glasses distribution.

Input

The first line contains an integer N ($1 \leq N \leq 10^5$) indicating the number of tables around the circular pond. The second line contains N integers $G_1, G_2, \ldots, G_N$ ($0 \leq G_i \leq 1000$ for $i = 1, 2, \ldots, N$), representing the number of glasses in the different tables. These numbers are given in clockwise order. It is guaranteed that N divides $\sum_{i=1}^{N} G_i$.

Output

Output a single line with an integer indicating the smallest possible total cost for fixing the glasses distribution.

Sample input 1	**Sample output 1**
4 14 10 6 10	8

Sample input 2	**Sample output 2**
6 24 122 0 37 49 242	454

Sample input 3	**Sample output 3**
6 0 0 0 0 60 0	150

Sample input 4	**Sample output 4**
1 0	0

Problem D – Dividing Candy

Bob and Charlie are two brothers that like powers of 2 a lot. Their mum decided to give them N boxes of candy, each of them containing a number of candy bars that is a power of 2.

They want to split the boxes between them, that is, for each box, they will decide who gets it. Each box must be given to exactly one brother.

Now they wonder: is it possible that, for each of the two brothers, the total amount of candy bars he receives is also a power of 2?

For example, if $N = 4$ and the boxes contain 4, 4, 32, and 8 candy bars, the answer would be *yes*, as one possible solution is giving the third box to Bob (32 candy bars), and the remaining boxes to Charlie ($4 + 4 + 8 = 16$ candy bars in total).

Input

The first line contains an integer N ($1 \leq N \leq 10^5$), the number of boxes the brothers want to split. The second line contains N integers $A_1, A_2, \ldots, A_N$ ($0 \leq A_i \leq 10^5$ for $i = 1, 2, \ldots, N$), indicating that the i-th box has 2^{A_i} candy bars.

Output

Output a single line with the uppercase letter "`Y`" if it is possible to split the boxes so that the total amount of candy received by each brother is a power of 2, and the uppercase letter "`N`" otherwise.

Sample input 1	**Sample output 1**
`4` `2 2 5 3`	`Y`

Sample input 2	**Sample output 2**
`1` `42`	`N`

Sample input 3	**Sample output 3**
`5` `3 1 4 1 5`	`N`

Sample input 4	**Sample output 4**
`7` `0 0 1 2 3 4 5`	`Y`

Problem E – Excellent Views

Shiny City is a beautiful city, famous for three things: the fact that it only has one street, the fact that all buildings have different heights, and the breathtaking views from the top of said buildings.

Since the pandemic began, the amount of tourists that visit Shiny City has gone down significantly. You are determined to write an amazing blog to attract more tourists and impede financial doom to your lovely, but terribly inefficient city. Unfortunately, there is still some information missing from the blog.

In Shiny City there are N buildings, and the i-th building is identified by its position i. Going from building i to building j takes $|i - j|$ minutes. Each building has a different height H_i, and the taller the building, the better the view from its top.

If you are at a certain building, it might be worth going to a different building that has a better view. Because of transportation costs, it's never worth it to go to a building if there is a taller one that you can reach without using more time.

Formally, we can say that going from building i to another building j is worth it if there is no k such that $|i - k| \leq |i - j|$ and $H_j < H_k$. Note that k may be equal to i.

You want to write on your blog, for each building, how many other buildings are worth going to from it. Please gather this information, otherwise Shiny City will be forever doomed.

Input

The first line contains an integer N ($1 \leq N \leq 10^5$), the number of buildings in Shiny City. The second line contains N different integers $H_1, H_2, \ldots, H_N$ ($1 \leq H_i \leq 10^9$ for $i = 1, 2, \ldots, N$), where H_i is the height of building i.

Output

Output a single line with N integers, such that the i-th of them represents the number of buildings worth going to from building i.

Sample input 1	**Sample output 1**
10 23 20 7 30 43 70 5 42 67 10	3 4 3 2 1 0 1 2 1 2

Problem F – Fascinating Partitions

A subarray of an array is a contiguous portion of the array. A partition of an array into subarrays is a collection of subarrays that cover all the array without overlaps (each element in the array belongs to exactly one subarray). For instance, if $A = [3, 1, 4, 1, 5]$ is an array, $[3, 1, 4]$ and $[1, 5]$ form a partition of A into subarrays, while $[3, 4, 5]$ is *not* a subarray of A.

Given an integer array and a partition of the array into non-empty subarrays, we define the cost of each subarray as its maximum element, and the cost of the whole partition as the sum of the costs of the subarrays.

As an example, consider the array $[3, 5, 7, 1, 2, 4]$. The partition formed by the subarrays $[3, 5]$, $[7]$ and $[1, 2, 4]$ has cost $5 + 7 + 4 = 16$, while the partition formed by the subarrays $[3]$, $[5, 7, 1]$ and $[2, 4]$ has cost $3 + 7 + 4 = 14$. Both partitions are formed by $k = 3$ subarrays, but they have different costs. Other partitions may have different costs.

Given an array A of N integers, and an integer k such that $1 \leq k \leq N$, consider the set $P(A, k)$ containing all the partitions of A into k non-empty subarrays. Can you compute the minimum cost over $P(A, k)$? Can you also compute the maximum cost? For each possible k? Okay, go ahead then.

Input

The first line contains an integer N ($1 \leq N \leq 8000$), the number of elements in the array A. The second line contains N integers $A_1, A_2, \ldots, A_N$ ($1 \leq A_i \leq 10^9$ for $i = 1, 2, \ldots, N$) representing the array.

Output

Output N lines, such that the k-th line contains two integers indicating respectively the minimum and maximum costs over $P(A, k)$.

Sample input 1	**Sample output 1**
6 3 5 7 1 2 4	7 7 10 12 12 16 14 19 17 21 22 22

Sample input 2	**Sample output 2**
4 1 1 1 1	1 1 2 2 3 3 4 4

Problem G – Game of Slots

Alice and Bob are playing the following game. Initially, each of them receives N cards, each card with an integer randomly and uniformly chosen from the interval $[1, 10^{18}]$ written on it. There are also N slots, numbered from 1 to N.

Alice starts by placing her cards on the slots, in such a way that exactly one card is placed on each slot. She may choose any of the $N!$ options of placement she wants. After that, Bob looks at the numbers placed by Alice on each slot and does the same with his cards. The game then ends, and for each slot i the person who placed the largest number on it earns i points ($i = 1, 2, \ldots, N$). If both of the placed numbers happen to be equal, the players share the points for this slot, that is, each player gets $i/2$ points.

As an example of the game, suppose $N = 3$, Alice is given cards with numbers 10, 20 and 40, and Bob is given cards with numbers 10, 30 and 50. Alice may choose to place numbers 40, 20 and 10 on slots 1, 2 and 3 respectively. After seeing her moves Bob may choose to place numbers 30, 50 and 10 on slots 1, 2 and 3 respectively. In this case, Alice earns 1 point for slot 1, 0 points for slot 2, and $3/2 = 1.5$ points for slot 3. Her total score is 2.5 points.

The purpose of the above example is to illustrate the rules of the game. Alice and Bob may not place their cards as described, because both of them play optimally. You may wonder what does it mean.

In his turn, Bob has full information on Alice's cards and decisions, and of course, he knows his cards. He plays in such a way to maximize his total score.

On the other hand, Alice does not know Bob's cards when it's her turn. She knows Bob's strategy, and that his cards were randomly and uniformly chosen. Thus, she plays to maximize the expected value of her total score.

Given N, you must calculate the expected value of Alice's total score at the end of the game, if the players play optimally as described.

Input

The input consists of a single line that contains an integer N ($1 \leq N \leq 100$), the number of slots in the game.

Output

Output a single line with a number indicating the expected value of Alice's total score at the end of the game, if the players play optimally as described. The result must be output as a rational number with exactly six digits after the decimal point, rounded if necessary.

Sample input 1	**Sample output 1**
1	0.500000

Sample input 2	**Sample output 2**
2	1.333333

Sample input 3	**Sample output 3**
99	589.631287

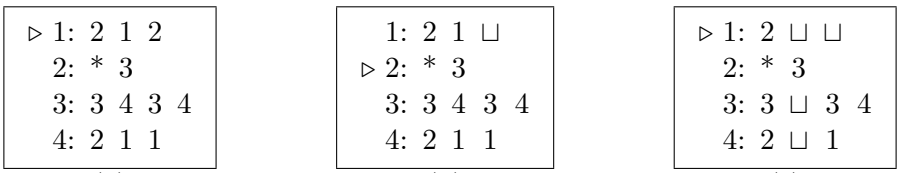
Problem H – Halting Wolf

Senoof loves programming languages, and the only thing he loves more than using them is creating new ones. His latest invention is the Wolf Programming Language, a very simple language consisting of only two types of instructions. They are numbered consecutively and written one under the other to make a program. Execution starts at instruction 1 and continues until the program gets stuck.

The two types of instructions are:

- "$K\ L_1\ L_2\ \cdots\ L_K$" is a finite jump. Each value L_i is an instruction number in the program, while K indicates how many of them are specified. When a finite jump is executed, one of the values L_i is chosen, and the execution continues with instruction L_i. But that's not all! The program changes the finite jump instruction so as to consume the chosen value. If a program executes a finite jump without available values, it gets stuck and halts.
- "$*\ L$" is an infinite jump. When it's executed, the program continues with instruction L, leaving the infinite jump instruction unmodified.

I know, Senoof is crazy, but it's not that difficult. The picture below shows an example, where current instruction is indicated with a ▷ sign, and a consumed value is denoted with a ⊔ sign. The program in (a) starts execution at instruction 1, which is a finite jump. Suppose that the second value is chosen, that is, execution continues with instruction 2 and this value is consumed in instruction 1, which yields the situation shown in (b). Since instruction 2 is an infinite jump to instruction 3, execution continues with this instruction, without consuming any value from instruction 2. Now imagine that from instruction 3 execution jumps to instruction 4, then to instruction 1, and then again to instruction 1, consuming the corresponding values. The situation at this point is shown in (c). As you can see the program gets stuck and halts, because there are no available values for jumping.



(a) (b) (c)

After some playing around, Senoof noticed that programs written in Wolf may run forever, which does not imply that a given instruction can be executed infinitely many times. He kindly just sent us the following example of a program that may run forever, although instruction 1 can be executed at most twice.

```
1: 2 1 2
2: * 4
3: 3 4 3 4
4: * 2
```

Given a program written in Wolf, you must determine the maximum number of times that instruction 1 can be executed.

Input

The first line contains an integer N ($1 \leq N \leq 100$), the number of instructions the program has. Each of the next N lines describes an instruction. A finite jump is represented with a non-negative integer K followed by K integers $L_1, L_2, \ldots, L_K$ ($1 \leq L_i \leq N$ for $i = 1, 2, \ldots, K$). On the other hand, an infinite jump is described with the character "$*$" (asterisk) followed by an integer L ($1 \leq L \leq N$). It is guaranteed that the total amount of instructions mentioned in the finite jumps is at most 10^4.

Output

Output a single line with an integer indicating the maximum number of times instruction 1 can be executed, or the character "*" (asterisk) if instruction 1 can be executed infinitely many times.

Sample input 1	**Sample output 1**
4 2 1 2 * 3 3 4 3 4 2 1 1	3

Sample input 2	**Sample output 2**
4 2 1 2 * 4 3 4 3 4 * 2	2

Sample input 3	**Sample output 3**
4 2 2 3 2 3 4 1 1 1 1	3

Sample input 4	**Sample output 4**
3 * 3 * 1 * 2	*

Sample input 5	**Sample output 5**
1 0	1

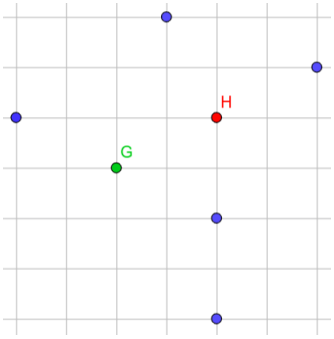
Problem I – Impenetrable Wall

The president of Happyland needs to be reelected. She will do so by focusing her last efforts on one of the things the population of Happyland loves the most: children. She knows it is a great problem that children keep running away from the national orphanage, so she decided to rebuild the wall around it so the orphanage will be more secure and the population will be happier.

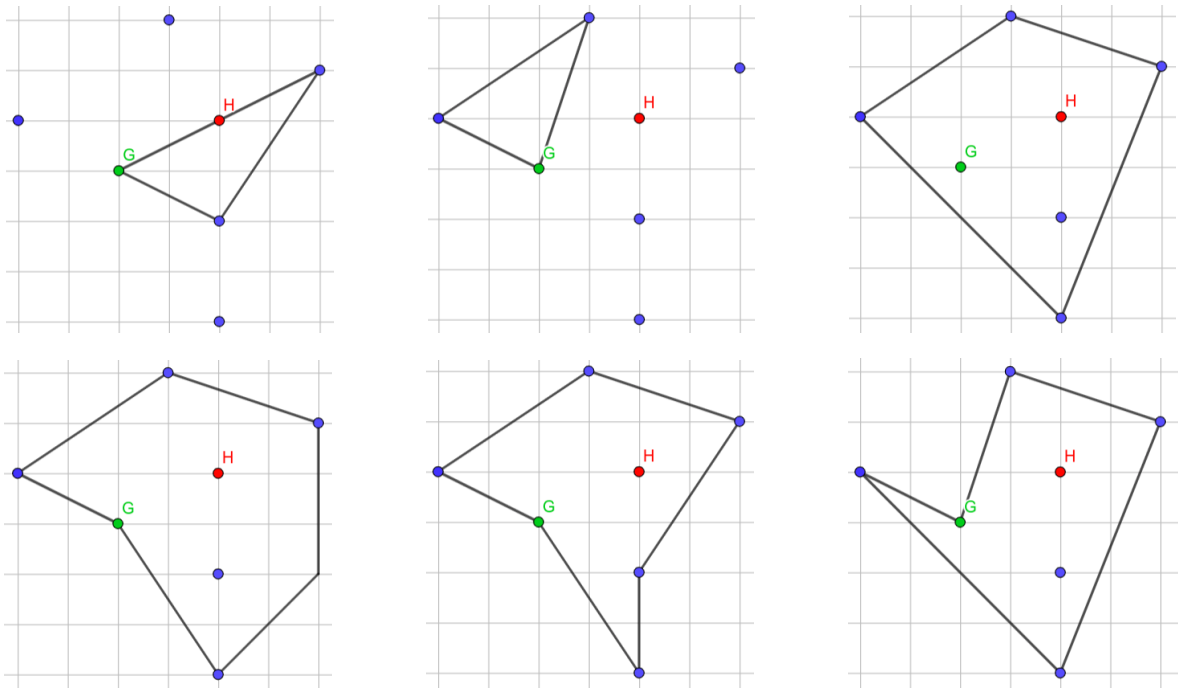
The area around which the wall must be built consists of the orphanage house, the orphanage gate, and a set of observation towers, left from Happyland's glory days.

The president of Happyland determined, in a meeting with orphanage specialists, that a safe wall must be a polygon such that:

1. The house is strictly enclosed by the wall.
2. The gate is a vertex of the wall and all other vertices are observation towers.
3. All internal angles of vertices of the wall that are observation towers are strictly less than 180 degrees (note that this does not apply to the gate vertex).
4. The whole wall is visible from the house. This means that for every point in the wall, the segment between the house and that point does not pass through the wall.

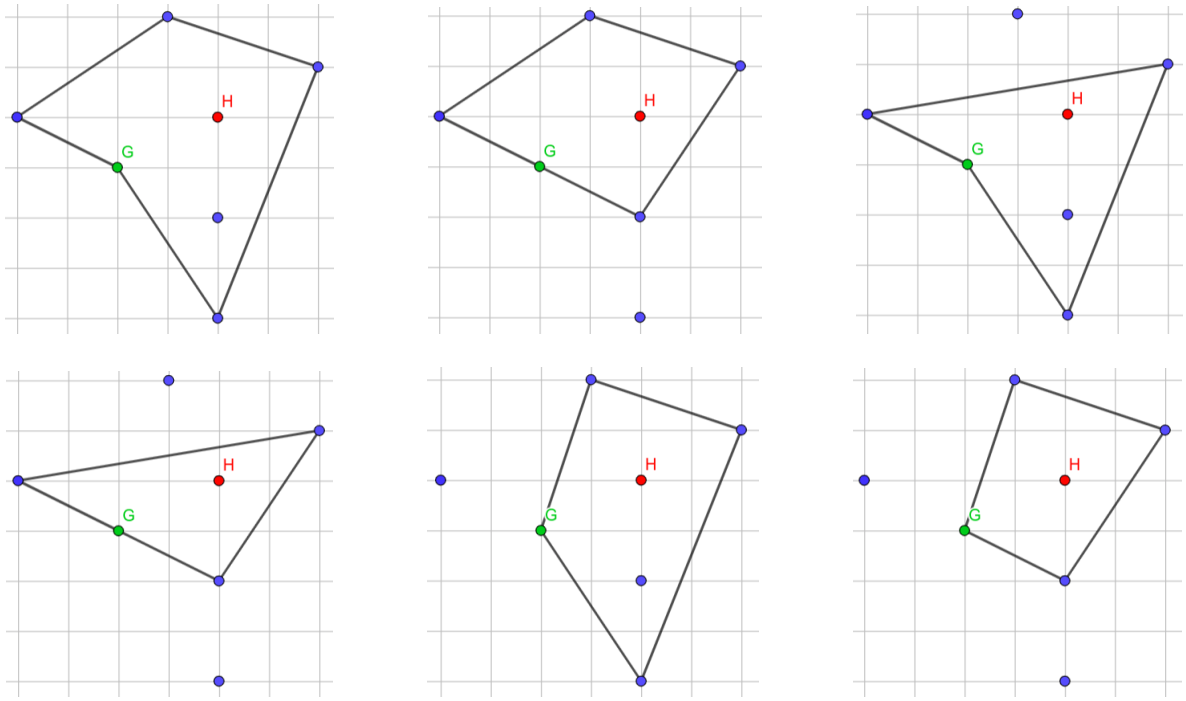


A possible configuration of object as in Sample input 1. The point H represents the house, the point G is the gate, and all other points are observation towers.



The walls represented above violate one or more rules, so they are not valid walls.

Now the president wants your help to know how many distinct safe walls can be built. Two walls are distinct if and only if there is an observation tower that is a vertex of one wall but not the other.



The walls represented above are all the distinct valid walls for Sample input 1.

Input

The first line contains two integers X_h and Y_h, the coordinates of the house. The next line contains two integers X_g and Y_g, the coordinates of the gate. The next line contains an integer N ($0 \leq N \leq 300$), the number of observation towers. Each of the next N lines contains two integers X_t and Y_t, the coordinates of a tower. All mentioned coordinates (X, Y) are distinct and such that $-10^9 \leq X, Y \leq 10^9$.

Output

Output a single line with an integer indicating the number of distinct walls that follow the restrictions. Because this number can be very large, output the remainder of dividing it by $10^9 + 7$.

Sample input 1	**Sample output 1**
0 0 -2 -1 5 2 1 0 -4 -1 2 -4 0 0 -2	6

Sample input 2	**Sample output 2**
2 2 2 0 3 -1 2 4 7 2 -2	1

Sample input 3	**Sample output 3**
100 100 -200 -120 0	0

Problem J – Job Allocator

The Infrastructure Consortium for Public Compute (ICPC) is a network of computers ran by volunteers from all around the world that share compute resources with one another. Contributors are able to plug and unplug their machines on the network, and also to run compute jobs on the network machines. With the ICPC, important projects that would otherwise have prohibitive infrastructure costs (like running online judges for programming competitions) become viable endeavors.

As great as it sounds on paper, for now, the ICPC is just a dream. To make it work, a key piece of software is missing: the job allocator. This is where you come in: the community is counting on you to make this important (but voluntary, of course) contribution.

The network is extremely dynamic: machines connect and disconnect all the time. The job allocator needs to keep track of the machines that are currently connected and which resources they share. There are several types of resources, such as CPU cores, GPUs, and SSD disks. One machine can share one or more resources, possibly more than one of the same type. Moreover, at any point in time, users can request machines to run compute jobs. For that, they specify a list of resources that a machine needs to run their job, and the job allocator has to determine how many of the currently connected machines have all the required resources to run the job. For example, for a job that needs one CPU core and two GPUs, the allocator would need to count how many machines have at least one CPU core, and two or more GPUs.

Your task is to simply count how many connected machines satisfy each job's resource requirements, since another volunteer took on the task of implementing the actual assignment of jobs to machines. The whole ICPC community depends on you. Are you able to help?

Input

The first line contains two integers N ($1 \leq N \leq 10^5$) and K ($1 \leq K \leq 8$), indicating respectively the number of network events that must be processed and the number of types of resources that are available on the ICPC. Events are described in chronological order in the next N lines, one event per line. There are three types of events.

If the event represents that a new machine is being connected to the network, the line contains the uppercase letter "`C`", followed by an integer R ($1 \leq R \leq 8$) indicating the number of resources the machine is sharing, followed by R integers $T_1, T_2, \ldots, T_R$ ($1 \leq T_i \leq K$ for $i = 1, 2, \ldots, R$), describing the type of each of the shared resources. New machines are implicitly assigned unique sequential integer identifiers by the ICPC, starting at 1.

When the event represents that a machine is being disconnected from the network, the line contains the uppercase letter "`D`", followed by an integer indicating the identifier of the machine. It is guaranteed that this identifier corresponds to a valid connected machine.

Finally, if the event represents that a user wants to run a job, the line contains the uppercase letter "`J`", followed by an integer R ($1 \leq R \leq 8$) indicating the number of resources the job needs, followed by R integers $T_1, T_2, \ldots, T_R$ ($1 \leq T_i \leq K$ for $i = 1, 2, \ldots, R$), describing the type of each of the required resources. It is guaranteed that the input contains at least one event of this type.

Output

Output a line for each event of type "`J`". The line must contain an integer indicating the number of machines that at the moment of the event are connected to the network and provide all the requested resources. Write the results in chronological order, that is, using the same order as the input.

Sample input 1	**Sample output 1**
3 2 C 3 1 1 2 J 2 1 2 J 2 2 2	1 0

Sample input 2	**Sample output 2**
11 3 C 2 1 2 J 1 3 C 3 1 2 3 J 1 3 J 1 1 D 2 J 1 3 J 2 1 2 J 2 1 1 D 1 J 1 2	0 1 2 0 1 0 0

Problem K – Keylogger

Lately you have been very curious about your typing speed, and you have been wondering how long it takes for you to press each key on your keyboard, which has K keys.

To figure that out, you installed a keylogger on your own computer. It has been registering the delta time between each pair of key presses. After collecting data for a couple of weeks, you now have access to a 2-dimensional matrix T with K rows and K columns. The element at the i-th row and j-th column is $T_{i,j}$, and it represents how long it takes, on average, for you to press the key j right after having pressed the key i. For example, the element $T_{3,5}$ represents how long it takes, on average, for you to press the key 5 right after having pressed the key 3. Coincidentally, each row on T is ordered non-decreasingly.

Given that your typing speed varies according to the time of the day and your mood, your keylogger has also given you a latency margin error L. That means that, for every pair of keys i and j on your keyboard, it actually takes between $T_{i,j} - L$ and $T_{i,j} + L$, inclusive, for you to press the key j right after having pressed the key i.

You classified for the South American ICPC regional competition, and you have been asked to update some of your contact information on the ICPC website. The problem is that you have been studying so hard that you forgot your password. All you remember is that your password has length N.

Luckily, your keylogger also has data about the last time you typed your password on that website. So now you have an array P with $N - 1$ elements. Each element P_i represents the delta time between each consecutive key presses from your password. In other words, P_1 represents the delta time between you pressing the keys of the first and the second characters of your password, P_2 is the delta time between you pressing the keys of the second and third characters of your password, and so on. Notice that the latency L does not apply to P, because each P_i is not an average but a single delta time, measured precisely.

You need to recover your password as soon as possible. Your plan now is to try every sequence of keys that is compatible with the information you have. A sequence S of length N is compatible with L, T, and P if each pair of consecutive keys S_i and S_{i+1} satisfy that $T_{S_i,S_{i+1}} - L \leq P_i \leq T_{S_i,S_{i+1}} + L$. How many such sequences are there?

Input

The first line contains two integers K ($1 \leq K \leq 750$) and L ($0 \leq L \leq 10^9$), indicating respectively how many keys are there on your keyboard and the latency margin error given by your keylogger. The next K lines contain K integers each, representing the matrix T. The j-th integer on the i-th line is $T_{i,j}$ ($1 \leq T_{i,j} \leq 10^9$ for $i = 1, 2, \ldots, K$ and $j = 1, 2, \ldots, K$). Recall that $T_{i,j}$ indicates how long it takes, on average, for you to press the key j right after having pressed the key i, and that each row on T is ordered non-decreasingly ($T_{i,j} \leq T_{i,j+1}$ for $i = 1, 2, \ldots, K$ and $j = 1, 2, \ldots, K - 1$). The next line contains an integer N ($2 \leq N \leq 10^4$), representing the length of your password. The final line contains $N-1$ integers $P_1, P_2, \ldots, P_{N-1}$ ($1 \leq P_i \leq 10^9$ for $i = 1, 2, \ldots, N - 1$), denoting the delta time between each consecutive key presses from your password.

Output

Output a single line with an integer indicating how many different sequences of keys are compatible with the information you have. Because this number can be very large, output the remainder of dividing it by $10^9 + 7$.

Sample input 1	**Sample output 1**
4 0 1 1 3 5 2 4 4 10 1 1 1 8 5 6 7 8 5 2 3 8 5	1

Sample input 2	**Sample output 2**
3 3 9 10 15 9 13 16 3 5 6 3 10 5	0

Sample input 3	**Sample output 3**
5 1 1 5 6 8 10 1 2 4 5 5 5 5 5 6 8 3 3 3 4 5 1 1 3 4 5 4 1 3 7	4

Problem L – Lola's Schedule

Lola is an energetic girl with many interests, making each day an ocean of possibilities for her, full of exciting activities she is more than willing to participate in. Unfortunately, many of the activities Lola participates in happen in closed spaces, and because of that her vitamin D levels are slightly below ideal. To help, a doctor prescribed a vitamin supplement that she must take daily every X minutes.

Lola wrote an app to keep track of her activities. The main feature of the app is the scheduling of activities. Each activity consists of a title and its start and end times. The app also allows creating one-off reminders, consisting simply of a title and the moment she should receive a single notification, and recurring reminders, consisting of a title, the time for the first notification, and how often the reminder should repeat. After buying the supplement, Lola wants to add a recurring reminder to the app, that repeats every X minutes, to make sure she will get notified whenever it is time to take the supplement. Ideally, she prefers not to have to take the supplement while participating in an activity. With such a busy schedule, she is having a hard time coming up with the ideal moment to start taking it.

Your task is to help Lola to choose the ideal time T after buying the supplement to take it for the first time. A time T is said to be ideal if it is at most `8` hours past the moment when Lola bought the supplement, the number of times when she needs to take the supplement which conflicts with her activities is minimum, and there's no other time earlier than T that would result in the same number of conflicts.

For clarity purposes, suppose that Lola needs to take the supplement every `30` minutes, bought it a certain day at `10:00`, and has a single activity scheduled from `18:00` to `18:30` during that day. If she chooses to take the supplement for the first time immediately, she would take it at `10:00`, `10:30`, ..., `17:30`, `18:00`, `18:30`, ... Thus, she'd have two conflicts with her activity (one at `18:00` and the other at `18:30`).

Although Lola cannot wait until her activity ends to take the supplement for the first time (because of the `8` hours limit), she can reduce the number of conflicts by waiting `7` hours and `59` minutes. In that way she would take the supplement at `17:59`, `18:29`, `18:59`, ..., with a single conflict at `18:29`. Much better, isn't it? However, this is not ideal, because there is an earlier time that also produces a single conflict, which is waiting `1` minute and taking the supplement for the first time at `10:01`, with a single conflict at `18:01`. As it's not possible to completely avoid conflicts, the ideal time, in this case, is taking the supplement for the first time `1` minute after buying it.

You'll be given a list of activities, exported from the app Lola wrote, containing information about all her activities starting after the time when she bought the supplement. Lola already did some pre-processing of the data and each activity is described by the number of minutes from the time she bought the supplement and its duration in minutes.

Input

The first line contains two integers N ($1 \leq N \leq 10^4$) and X ($1 \leq X \leq 720$), indicating that Lola has N upcoming activities exported from her app, and that she must take the supplement every X minutes. Each of the following N lines describes an activity with two integers S and D ($1 \leq S, D \leq 10^5$), representing that the activity starts S minutes after buying the supplement, and that its duration is D minutes. Activities do not overlap, that is, given any pair of different activities, one of them ends strictly before the other starts, or vice versa.

Output

Output a single line with two integers T and C, indicating respectively the ideal time for taking the supplement for the first time, expressed in minutes since buying the supplement, and the number of conflicts that taking the supplement at this time will lead to.

Sample input 1	**Sample output 1**
1 30 480 30	1 1

Sample input 2	**Sample output 2**
5 30 195 30 120 45 240 30 30 60 300 180	451 1

Sample input 3	**Sample output 3**
4 720 60 30 150 75 750 60 1500 60	0 0

Sample input 4	**Sample output 4**
2 720 1 479 482 298	0 1

Problem M – May I Add a Letter?

You have a string S of length N and you are asked to perform a sequence of Q updates of two types to S:

- Append a given character to the end of S.
- Delete the last character from S.

Initially and after each update, you must calculate the number of distinct strings that occur at least twice as substrings of S.

For example, if S is initially "`ABABC`", the answer is 3, as "`A`", "`B`" and "`AB`" occur twice as substrings of S. If you are asked to append the character "`C`", S will become "`ABABCC`" and the answer will be 4, as now "`C`" occurs twice too. If you are asked to append "`C`" again, S will be "`ABABCCC`" and the answer will be 5, as "`CC`" occurs twice now. If you are given a delete operation now, S will become "`ABABCC`" and the answer will be 4 again.

Input

The first line contains a string S of length N ($1 \leq N \leq 10^5$), indicating the initial value of the string. Each character of S is an uppercase letter. The second line contains a string U of length Q ($1 \leq Q \leq 10^5$), representing the updates to perform. Each character of U is either an uppercase letter indicating that such a letter must be appended, or the character "`-`" (hyphen) denoting a delete operation. The updates must be applied in the order they appear in U. It is guaranteed that delete operations are not applied to empty strings.

Output

Output $Q + 1$ lines, each line with an integer indicating the number of distinct strings that occur at least twice as substrings of S. Line 1 refers to the initial value of S, while line $i + 1$ refers to the value of S after applying the first i updates ($i = 1, 2, \ldots, Q$).

Sample input 1	**Sample output 1**

`ABABC`
`CC-`
	`3`
	`4`
	`5`
	`4`

Sample input 2	**Sample output 2**

`ABAB`
`A--CC`
	`3`
	`5`
	`3`
	`1`
	`1`
	`2`

Sample input 3	**Sample output 3**
`HAVE` `FUN`	`0` `0` `0` `0`

Problem N – Non-Integer Donuts

Neil is a very important lawyer with a very important bank account. Since Neil is such a successful lawyer with many clients, he deposits money to his account every single morning.

After going to the bank and depositing money, Neil goes to work. And there lies Neil's great weakness: a donut shop. You see, Neil is a recovering donut addict, and although he hasn't eaten a donut in years, he can't help but wonder how many \$1.00 donuts he could buy with the money in his account if he were to relapse.

Having \$5.00 in his account means 5 donuts Neil could have, but what about \$4.50? Well, that is more than 4 donuts for sure, but definitely less than 5. How would one even buy a non-integer amount of donuts? That concept confuses Neil, so every time his account balance is not an integer, he stops to ponder the nature of non-integer donuts and ends up being late to work.

Now Neil has been late too many times and is starting to worry he will lose his job. He wants to know how many times he will be late to work during the next N days, given his initial account balance and the amount of money he will deposit each day. Please answer this for him, or else Neil will start pondering again.

Input

The first line contains an integer N ($1 \leq N \leq 1000$), the number of days Neil is interested in. Each of the next $N + 1$ lines contains a string representing an amount of money. The first string is Neil's account initial balance, while the following N strings are the amounts Neil will deposit to his account in the different days. Each string has the form $\$X.Y$ where X is a substring of length 1 or 2 indicating the whole money in the amount $\$X.Y$, while Y is a substring of length exactly 2 denoting the cents in the amount $\$X.Y$. Both X and Y are made of digits, at least one of them contains a non-zero digit, and X does not have leading zeros.

Output

Output a single line with an integer indicating how many times Neil will be late to work during the following N days.

Sample input 1	**Sample output 1**
1 \$1.57 \$3.14	1

Sample input 2	**Sample output 2**
4 \$1.00 \$0.01 \$0.99 \$10.00 \$98.76	2