# Problem A. Apple Diameter

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 1 second |
| Memory limit: | 256 megabytes |

An apple is a polygon where all corners but one are convex. In other words, there is one special corner whose inner angle is strictly above 180 degrees. All other corners have an inner angle strictly below 180 degrees.

The distance between two points on the border of the apple is the length of the shortest path completely inside the apple.

The diameter is the maximum distance between two border points.

Given the apple, calculate its diameter.

## Input

The first line contains $n$ ($4 \leq n \leq 1000$), the number of points in the apple. The following $n$ lines contain two integers $x$ and $y$ ($0 \leq x, y \leq 10^6$), the coordinates of the points. The points are given in counterclockwise order.

## Output

The diameter of the apple as a floating point number. Your answer is accepted if its relative or absolute error is below $10^{-9}$.

## Examples

| standard input | standard output |
|---|---|
| 5<br>0 0<br>3 4<br>6 0<br>6 6<br>0 6 | 10 |
| 5<br>0 0<br>3 2<br>6 0<br>6 6<br>0 6 | 8.4852813742385695406 |
| 5<br>10 0<br>13 4<br>16 0<br>26 6<br>0 6 | 26 |

# Problem B. Biased Benchmarking

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 1 second |
| Memory limit: | 256 megabytes |

There are two companies, A and B. Companies A and B are developing competing software tools PSI and PHI, respectively. PSI and PHI are tools for general artificial intelligence and problem solving. You have a friend who is a sales representative for company A.

There is a set of $N$ benchmark problems, numbered from 1 to $N$. For each of them, your friend knows the time it takes PSI and PHI to solve the benchmark.

Your friend has already figured out that there are two basic kinds of potential customers, those who like large absolute gains (group X), and those who like large relative gains (group Y). When presented with a set of benchmark results, members of group X will compute the total benchmark time for PHI and subtract from it the total benchmark time for PSI. In contrast, group Y will divide the total benchmark time for PHI by the total benchmark time for PSI. Both groups prefer large numbers.

Your friend has already figured out (using PSI), which potential customers belong to which group. Your friend intends to use targeted advertising to show each group a (potentially) different ad, where each ad contains the benchmark results for some subset of the benchmarks. The only remaining question is which sets of benchmarks to display on the two ads. Your friend has already asked PSI this crucial final question, but it refused to cooperate due to ethical concerns. Can you help?

For each of the two ads, one shown to group X and one shown to group Y, determine an optimal non-empty set of benchmarks whose results to print on the ads. In order to avoid leaving the impression that the benchmarks have been cherry-picked, maximize the number of benchmarks on each ad under the given constraints.

## Input

The first line of the input contains a single integer $N$ ($1 \leq N \leq 10^5$), the number of benchmarks. The second line of the input contains $N$ non-negative integers between 1 and $10^9$. The $i$-th number is the time it took PSI to solve benchmark $i$, in milliseconds. The third line of the input contains $N$ non-negative integers between 1 and $10^9$. The $i$-th number is the time it took PHI to solve benchmark $i$, in milliseconds.

## Output

Print two lines, describing the set of benchmarks to print on each ad. The first line describes the set of benchmarks shown to group X and the second line describes the set of benchmarks shown to group Y. Each line begins with an integer $K > 0$, describing the number of benchmarks on the respective ad, followed by $K$ integers between 1 and $N$ in ascending order, describing the benchmarks to be shown on the ad. Numbers on the same line should be separated with single spaces.

## Examples

| standard input | standard output |
|---|---|
| 3<br>1 1 1<br>2 1 2 | 3 1 2 3<br>2 1 3 |
| 4<br>1 1 2 2<br>2 2 1 1 | 2 1 2<br>2 1 2 |

## Note

In the first example test case, there are three benchmarks. On the first and third benchmark, PHI

takes twice as long as PSI. The ad for group X shows all three benchmarks. Note that showing only benchmarks 1 and 3 would also maximize the difference between the total benchmark times of PHI and PSI ($(2+2)-(1+1)=(2+1+2)-(1+1+1)=2$), but this would be fewer benchmarks. The ad for group Y shows only benchmarks 1 and 3, resulting in a quotient of benchmark times $(2+2)/(1+1)=2$. This is the maximal number of benchmarks that can be shown, as $(2+1+2)/(1+1+1)=5/3<2$.

In the second example test case, PSI is twice as fast as PHI on the first two benchmarks, but on the remaining two benchmarks, PHI is twice as fast as PSI. The resulting ads hide the benchmarks where PHI is faster than PSI. This results in a total difference of $(2+2)-(1+1)=2$ and a quotient of $(2+2)/(1+1)=2$.

# Problem C. Commutative Concatenation

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 1 second |
| Memory limit: | 256 megabytes |

As you may know, $+$ is commutative: we have $a + b = b + a$. However, in C++, `+` denotes concatenation of `std::string`s, and we obtain confusing counterexamples to commutativity, like for example `std::string("abc") + std::string("123") == std::string("abc123") != std::string("123abc") == std::string("123") + string("abc")`!

Clearly this situation is not tenable. You are preparing a proposal to make `std::string` concatenation in C++ commutative: it should be undefined behavior to concatenate `std::string`s `S` and `T` with `S + T != T + S`. In particular, this will allow the optimizer to assume that `S + T == T + S` for any pair of strings `S` and `T`, which will bring a huge performance boost to most existing C++ programs that manipulate `std::string`s.

While this is a great proposal, you fear that the ISO C++ committee will reject it based on concerns about backwards compatibility. Therefore, you will need to argue that any code that relies on the non-commutativity of `std::string` concatenation is already broken anyway.

As a minor part of your convincing treatise about the merits of commutative concatenation, you in particular need an example, i.e., a pair of strings `S` and `T` with `S + T == T + S`. You have already decided that your strings should only contain the first $K$ small letters of the English alphabet, and that the length of `S` should be $N$ and the length of `T` should be $M$. You are using the following simple model to evaluate how convincing a given example is: each position in both of the two `std::string`s is assigned a score for each of the possible letters. The score of an example is the sum over all positions of the score of the particular letter in that position. Examples with higher score are more convincing. If two examples have the same score, the one with the lexicographically smaller concatenation is more convincing.[1]

Find the most convincing example.

## Input

The first line of the input contains three positive integers $K$, $N$ and $M$ ($1 \le K \le 26$, $K \cdot (N + M) \le 10^5$). Each of the following $N$ lines contains $K$ numbers, where the $j$-th number in the $i$-th line specifies the score of the $j$-th small English letter at the $i$-th position of the string `S`. Each of the next $M$ lines contains $K$ numbers, where the $j$-th number in the $i$-th line specifies the score of the $j$-th small English letter at the $i$-th position of the string `T`. All scores are between 0 and $10^4$.

## Output

Print the two strings `S` and `T` in order, each on its own line.

## Examples

| standard input | standard output |
|---|---|
| 2 2 1<br>1 2<br>1 2<br>1 2 | bb<br>b |
| 3 2 2<br>1 1 2<br>1 0 3<br>2 0 1<br>2 1 1 | ac<br>ac |

[1] Note how we did not need to specify in which order the two strings are to be concatenated!

# Problem D. Decreasing Debts

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 1 second |
| Memory limit: | 256 megabytes |

There are $N$ people in this world, numbered from 1 to $N$. They are using swiss francs to buy goods and services. Occasionally, a person might not have enough currency to buy what he wants or needs, so he borrows money from someone else, with the idea that he will repay the loan later with interest. Let $d(A, B)$ denote the debt of $A$ towards $B$, or 0 if there is no such debt.

Sometimes, this becomes very complex, as the person lending money can run into financial troubles before his debtor is able to repay his debt, and finds himself in the need borrowing money.

When this process runs for a long enough time, it might happen that there are so many debts that they can be consolidated. There are three ways this can be done:

1. Let $d(A, B) > 0$ and $d(B, C) > 0$. We can increase the $d(A, C)$ by $z$ and decrease $d(A, B)$ and $d(B, C)$ by $z$, where $0 < z \leq \min(d(A, B), d(B, C))$.
2. Let $d(A, B) > 0$ and $d(B, A) > 0$. We can decrease the $d(A, B)$ and $d(B, A)$ by $z$, where $0 < z \leq \min(d(A, B), d(B, A))$.
3. Let $d(A, B) > 0$ and $d(C, D) > 0$. We can decrease the $d(A, B)$ and $d(C, D)$ by $z$ and increase $d(C, B)$ and $(A, D)$ by $z$, where $0 < z \leq \min(d(A, B), d(C, D))$.

The total debt is defined as the sum of all debts:

$$\Sigma_d = \sum_{A,B} d(A, B)$$

Your goal is to use the above rules in any order any number of times, to make the total debt as small as possible.

**Helpful note:** The input is very large. If you are using C++ and standard streams (`std::cin` and `std::cout`), remember to call `std::ios_base::sync_with_stdio(false);` before reading the input.

## Input

The first line contains two space separated integers $N$ ($1 \leq N \leq 10^5$) and $M$ ($0 \leq M \leq 3 \cdot 10^5$), representing the number of people and the number of debts, respectively.

$M$ lines follow, each of which contains three space separated integers $u_i$, $v_i$ ($1 \leq u_i, v_i \leq N, u_i \neq v_i$), $d_i$ ($1 \leq d_i \leq 10^9$), meaning that the person $u_i$ borrowed $d_i$ francs from person $v_i$.

## Output

On the first line print an integer $D$ ($0 \leq D \leq 3 \cdot 10^5$), representing the number of debts after the consolidation. It can be shown that an answer always exists with this additional constraint.

After that print $D$ lines, $i$-th of which should contain three space separated integers $u_i, v_i, d_i$, meaning that the person $u_i$ owes the person $v_i$ exactly $d_i$ francs. The output must satisfy $1 \leq u_i, v_i \leq N$, $u_i \neq v_i$ and $d_i > 0$.

For each pair $i \neq j$, it should hold that either $u_i \neq u_j$ or $v_i \neq v_j$. In other words, each pair of people can be included at most once in the output.

## Examples

| standard input | standard output |
|---|---|
| 3 2<br>1 2 10<br>2 3 5 | 2<br>1 2 5<br>1 3 5 |
| 3 3<br>1 2 10<br>2 3 15<br>3 1 10 | 1<br>2 3 5 |
| 4 2<br>1 2 12<br>3 4 8 | 4<br>1 2 7<br>3 4 3<br>1 4 5<br>3 2 5 |
| 3 4<br>2 3 1<br>2 3 2<br>2 3 4<br>2 3 8 | 1<br>2 3 15 |

## Note

In the first sample, we can apply the first transformation with $A = 1$, $B = 2$, $C = 3$ and $z = 5$. After that, no further transformation reduces the total debt.

In the second sample, we can apply the first transformation with with $A = 1$, $B = 2$, $C = 3$ and $z = 10$. After that, the debts will be $d(1, 2) = d(2, 1) = d(3, 2) = 0$, $d(1, 3) = d(3, 1) = 10$, $d(2, 3) = 5$. Applying the second transformation on $A = 1$, $B = 3$ and $z = 10$, we cancel out the debts of persons 1 and 3. It can be shown that the debt cannot be reduced any further.

In the third case, there is no way to reduce the total debt. We can, however, use the third rule with $A = 1$, $B = 2$, $C = 3$, $D = 4$ and any $z$ up to 8. The sample output shows the situation when $z = 5$ is selected. Note that you don't have to minimise the **number** of debts, only the **total debt**.

# Problem E. Sampling equal subsequences

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 1 second |
| Memory limit: | 256 megabytes |

You are given two strings $s$ and $t$ of lengths $n$ and $m$. You sample a subsequence of $s$ and a subsequence of $t$ independently and uniformly at random by removing each character of $s$ and of $t$ with probability $\frac{1}{2}$. What is the probably that the two subsequences are the same?

## Input

The first line contains two integers $n$ and $m$ ($1 \leq n, m \leq 4000$). The second line contains two strings $s$ and $t$.

## Output

It can be proven that the answer can be written as a fraction $\frac{P}{Q}$ where $P$ and $Q$ are coprime. Print $P \cdot Q^{-1} \bmod (10^9 + 7)$ where $Q^{-1}$ is the modular multiplicative inverse of $Q$ modulo $10^9 + 7$.

## Examples

| standard input | standard output |
|---|---|
| 1 1<br>x x | 500000004 |
| 3 5<br>bxa acbba | 996093757 |
| 8 4<br>zbbzbzzb bzzb | 317871096 |

## Note

In the first sample, if we keep the character `'x'` in both $s$ and $t$, the resulting subsequences are both `"x"`. If we drop the character from both $s$ and $t$, the resulting subsequences are both empty and thus equal. Both of these cases have probabilty $\frac{1}{4}$, so the answer is $\frac{1}{2} \bmod (10^9 + 7) = 500000004$.

In the second sample, the possible resulting equal subsequences are: empty string with probabilty $\frac{1}{256}$, `"a"` with probabilty $\frac{1}{128}$, `"b"` with probabilty $\frac{1}{128}$ and `"ba"` with probabilty $\frac{1}{128}$. Hence the answer is $\frac{7}{256}$.

# Problem F. Candies

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 1 second |
| Memory limit: | 16 megabytes |

Alice has $n$ candies and $n^2$ friends. She wants to invite some of her friends, possibly zero, and give everyone, including herself, the same number of candies. However she wants to put aside $x$ candies for her mother who comes to visit on Sunday. Moreover, Alice wants every of the $x$ candies for her mother to carry a unique, personal dedication of one of her friends — therefore the number of guests invited must be at least $x$. Finally, no candies should go to waste, i.e., each candy needs to be given either to Alice's mother, one of Alice's friends, or Alice herself.

What is the number of possible numbers of guests that Alice can invite?

## Input

Two non-negative integers in decimal format on a single line, separated with a single space. The first one is the number of candies $n$. The second one is the number of candies to be put aside for Alice's mother $x$. You can assume $0 \leq x < n \leq 10^9$.

## Output

Single integer in decimal format — the number of possible numbers of guests.

## Examples

| standard input | standard output |
|---|---|
| 7 1 | 3 |
| 7 0 | 2 |

## Note

Consider the first sample. Given $n = 7$ candies including $x = 1$ candies to be put aside for Alice's mother, Alice can invite either 1, 2 or 5 guests. Thus the result is 3.

# Problem G. Oriental Graph

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 1 second |
| Memory limit: | 256 megabytes |

An oriental graph is a directed graph on $n$ vertices, for even $n$, such that there is a path from vertex $2i$ to vertex $2i + 1$ for all $i \in \{0, \ldots, n/2 - 1\}$.

You are given an undirected graph on $n$ vertices and have to determine whether it is possible to convert it into an oriental graph by choosing the direction of the given edges in a clever way.

## Input

The first line contains two numbers $n$ and $m$, where $1 \leq n \leq 1000$ is the number of vertices and $0 \leq m \leq 2500$ is the number of undirected edges in the original graph.

The following $m$ lines contain two numbers $a$ and $b$ each ($0 \leq a < n$, $0 \leq b < n$) – meaning that there is an edge between vertex $a$ and vertex $b$. It is guaranteed that no edge occurs twice.

## Output

If it is possible to make the input graph oriental, print ‘‘`oriental`’’, otherwise print ‘‘`non-oriental`’’.

## Examples

| standard input | standard output |
|---|---|
| 4 4<br>0 2<br>2 3<br>3 1<br>1 0 | oriental |
| 4 3<br>1 2<br>2 3<br>3 0 | non-oriental |
| 8 7<br>0 6<br>1 6<br>2 6<br>3 7<br>4 6<br>5 7<br>6 7 | oriental |

# Problem H. Highways

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 2 seconds |
| Memory limit: | 256 megabytes |

The government of the Shech Republic plans to connect the $n$ largest cities by a highway network so that each city can be reached from any other city, possibly passing through some intermediate cities. The highways are bidirectional and all crossings outside a city are multilevel. A survey has proposed $m$ possible routes (i.e., pairs of cities) which might constitute the network. It is guaranteed that each city can be reached from any other city by following these routes, possibly passing through some intermediate cities. In order to use the taxpayers' money efficiently, a cheapest network and companies building its routes will be selected by an independent consulting company. In particular, if multiple companies bid equally on some route, the consulting company may choose any of the companies to build the route at its own discretion. Moreover, each company may only submit a bid for a single route in the whole network.

There have already been made bids for each route. You own a construction company and know the lowest bid made so far for each route, due to your good connections with the government. Being an eager, but also risk averse entrepreneur, you would like to determine the highest possible bid for each route so that you are guaranteed to make a deal if you choose to bid on this route, regardless of which (cheapest) network and companies building its routes are selected by the independent consulting company. Recall that also your company may submit a bid for only a single route in the whole network.

## Input

The first line of the input contains two integers $n$ and $m$ ($2 \leq n \leq 2 \cdot 10^5, n-1 \leq m \leq 2 \cdot 10^5$), where $n$ and $m$ are the number of cities and the number of possible routes between them, respectively.

Each of the next $m$ lines contains three integers $u, v, b$ ($1 \leq u, v \leq n, u \neq v, 2 \leq b \leq 10^9$) meaning that one of the possible routes connects the cities $u$ and $v$ with the lowest bid $b$ made so far.

It is guaranteed that each city can be reached from any other city by following these routes, possibly passing through some intermediate cities.

## Output

Print the highest possible bid for each route on a separate line in the order the routes are given in the input.

## Examples

| standard input | standard output |
|---|---|
| 4 3<br>1 2 10<br>2 3 42<br>3 4 20 | 9<br>41<br>19 |
| 4 4<br>1 2 10<br>2 3 10<br>3 4 10<br>1 4 20 | 9<br>9<br>9<br>9 |

## Note

Note that all bids have to be positive integers.

# Problem I. Isolated Peaks

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 4 seconds |
| Memory limit: | 256 megabytes |

In topography, the isolation of a summit is the minimum distance to a point of equal elevation, representing a radius of dominance in which the peak is the highest point.

For example, the isolation of Uetliberg, the prominent mountain with look-out tower overseeing Zürich, is 6.9 km, as the nearest point of the same elevation is on a hill above Langnau. The isolation of Dufourspitze, the highest mountain of Switzerland, is 78.3 km towards the closest point on the eastern slope of Mont Blanc. The isolation of K2 – the second highest mountain of the world 8611 meters tall – is 1316 kilometers, because that is the distance to the closest point of the same elevation on the north-west ridge of Mount Everest. The isolation of Mount Everest is *infinite* as on Earth there are no points of the same elevation as its peak.

You are given a one-dimensional model of a mountain range. Your goal is to report the isolation of all peaks in this range. More precisely, you are given a list of peaks, where each peak has an $x$-coordinate $x_i$ and elevation $e_i$. On both sides of each peak, the hill goes down with a slope of 1, until it hits the sea level (point where the elevation is 0), or it meets the slope of an adjacent peak, forming a V-shaped valley.

For instance, if there are two peaks, $x_1 = 3$, $e_1 = 10$ and $x_2 = 8$, $e_2 = 12$, their slopes meet at point $(4.5, 8.5)$.

It is guaranteed that each peak is above the slope of all other peaks – in the example above, one cannot have $x_3 = 11$ and $e_3 = 9$, as the point $(11, 9)$ lies on or below the east slope of the mountain with peak $(8, 12)$.

## Input

The first line contains a single integer $N$ ($1 \leq N \leq 10^6$).

The second line contains $N$ space separated integers $x_1, x_2, \ldots, x_N$ ($-10^{18} \leq x_i \leq 10^{18}$), where $x_i$ is the horizontal coordinate of the $i$-th peak. It is guaranteed that the coordinates are pairwise distinct and increasingly sorted, i.e., $x_i < x_{i+1}$ for all $1 \leq i < N$.

The third line contains $N$ space separated integers $e_1, e_2, \ldots, e_N$ ($0 < e_i \leq 10^{18}$), where $e_i$ is the elevation of the $i$-th peak.

It is guaranteed that this is a valid mountain ridge.

**Helpful note:** The input is very large. If you are using C++ and standard streams (`std::cin` and `std::cout`), remember to call `std::ios_base::sync_with_stdio(false);` before reading the input.

## Output

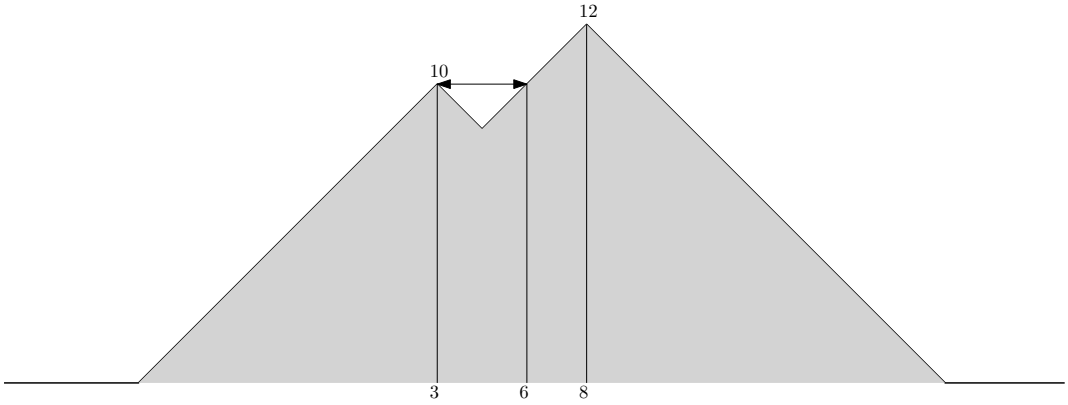
Output $N$ space separated integers $d_1, d_2, \ldots, d_N$ – the $d_i$ being the isolation of the $i$-th peak. If the isolation of the $i$-th peak is infinite, put $d_i = 0$ instead.

## Examples

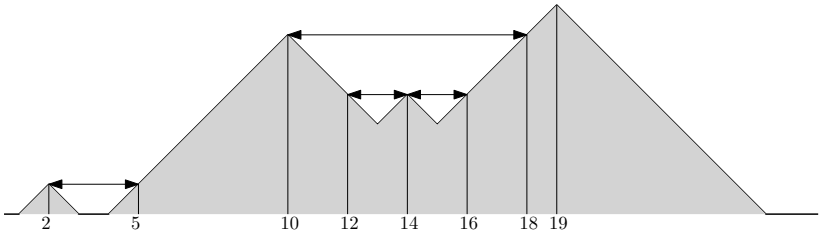
| standard input | standard output |
|---|---|
| 2<br>3 8<br>10 12 | 3 0 |
| 4<br>2 10 14 19<br>1 6 4 7 | 3 8 2 0 |
| 2<br>0 10<br>12 12 | 10 10 |

## Note

The first sample is the example from the problem statement. The mountain ridge contains two points of the elevation 10 other than the peak $(3, 10)$ itself – they are both on the slopes of the second peak, namely $(6, 10)$ and $(10, 10)$. The former is closer to $(3, 10)$, hence its isolation is $|6 - 3| = 3$. The second peak is the single highest point, so its isolation is infinite.




In the second sample, the closest point to the first peak is $(5, 1)$, the closest point to the second peak is $(18, 6)$, and the third peak has two equidistant closest points of the same elevation – $(12, 4)$ and $(16, 4)$.




In the third sample, the only two peaks are the closest points for each other.

# Problem J. Guybrush the Mighty Pirate

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 1 second |
| Memory limit: | 256 megabytes |

Guybrush is a mighty pirate. Recently he obtained a sequence of commands that should bring him to the location of a buried treasure. Each of the commands is an instruction to take a step north, south, east, or west.

Today, Guybrush decided that he will try following those instructions. Who knows, maybe he'll really be able to find a treasure.

You probably noticed that one important thing is missing from the instructions: A place where the person looking for the treasure should start their walk. But have no fear, Guybrush has that sorted out already. You see, recently he noticed a large X in the middle of a huge meadow. In the absence of clear instructions he may as well start from there. And that's precisely what he did. He spent the entire afternoon following the instructions step by step. When he got to the end of them, he realized he forgot his shovel, so he just gave up and headed home.

You are probably a much better pirate than Guybrush will ever be. If that is indeed the case, you already suspect that the instructions are fake, and you have a **pretty good** idea where the actual treasure lies.

You are given a string of the letters `N`, `S`, `E`, `W`, representing an instruction to move north, south, east and west, respectively. Compute the number of times Guybrush stood exactly on top of the buried treasure. (Yes, we **do** mean the large X where he started.)

## Input

The input consists of a single line. This line contains a non-empty string $s$ consisting of letters `N`, `S`, `E`, `W`. The length of $s$ won't exceed 10 000 characters.

## Output

Output a single integer $k$ – the number of times Guybrush stood exactly on top of the buried treasure including his initial position.

## Examples

| standard input | standard output |
|---|---|
| EEE | 1 |
| NWES | 2 |
| WEWE | 3 |

## Note

The starting and ending locations also count.

# Problem K. Timogehrov Complexity

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 1 second |
| Memory limit: | 64 megabytes |

Every computer science student should know the concept of Timogehrov complexity. For reference, we show the textbook definitions:

**Timogehrov's Definition:** A sequence of bytes (numbers between 0..255) is called Timogehrov if and only if:

1. it consists of a single byte $b$ ($0 \leq b \leq 255$), or
2. it is a repetition of $k$ consecutive Timogehrov sequences, or
3. it is a concatenation of two Timogehrov sequences.
4. it is the reverse of a Timogehrov sequence

**Lemma:** Every file is Timogehrov.

**Timogehrov Complexity:** The Timogehrov complexity of a file is the minimum number of steps (in Timogehrov's Definition) to reproduce it.

**Timogehrov's Theorem:** Let $c$ be the Timogehrov complexity of a file. Then the file can be compressed to $k \cdot c \log(c)$ bytes (for some constant $k$).

The theorem is typically proven by explicitly defining a compression algorithm, the so-called "Timogehrov's zipper". In Linux such files have the ending .tgz. See "Input"section for an informal specification of the Timogehrov format. Note that there are limits for $k$ in step 2, which are responsible for the factor of $\log(c)$ in the theorem.

The **Timogehrov checksum** of a sequence of bytes $b_1, b_2, \ldots, b_n$ is defined as $(\sum_{i=1}^{n}(b_i \cdot 1009^{i-1}))$ mod 1000000007.

Given a file as Timogehrov's zipper, calculate its Timogehrov checksum.

## Input

The first line contains a single integer $N \leq 10000$ an upper bound on the Timogehrov complexity of the given file. The following N lines (numbered from 1 to $N$) each contain one of the following zipper file descriptions:

- B <$b$>: where the file consists of a single byte $0 \leq b \leq 255$.
- R <$k$> <$i$>: where the file consists of $1 \leq k \leq 10^9$ repetitions of the Timogehrov file specified in line $i$.
- C <$i$> <$j$>: where the Timogehrov file is the concatenation of the Timogehrov files specified in lines $i$ and $j$.
- F <$i$>: flip file $i$

It is guaranteed that the zipper definition contains no cycles.

## Output

Output the Timogehrov checksum for the file specified on line 1 of the input.

## Examples

| standard input | standard output |
|---|---|
| 5<br>F 2<br>C 3 4<br>B 1<br>R 2 5<br>B 0 | 1018081 |
| 6<br>C 2 4<br>R 2 3<br>B 1<br>F 6<br>B 0<br>R 20 5 | 1010 |

## Note

The first sample contains 5 lines describing Files:

- File 1: (0,0,1), the flip of File 2
- File 2: (1,0,0), the concatenation of File 3 and File 4
- File 3: (1), a byte
- File 4: (0,0), 2 repetitions of File 5
- File 5: (0), a byte

The checksum of File 1 is $0 + (0 \cdot 1009) + (1 \cdot 1009^2) = 1018081$.

In the second sample File 1 is (1,1,0,0,0,...,0) which results in a checksum of $1 + (1 \cdot 1009) + (0 \cdot 1009^2) + \cdots + (0 \cdot 1009^{21}) = 1010$.