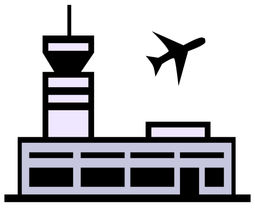
PROBLEM A

Airports



An airline company offers flights out of n airports, conveniently labeled from 1 to n. The flight time t_{ij} from airport i to airport j is known for every i and j. It may be the case that $t_{ij} \neq t_{ji}$, due to things like wind or geography. Upon landing at a given airport, a plane must be inspected before it can be flown again. This inspection time p_i is dependent only on the airport at which the inspection is taking place and not where the previous flight may have originated.

Given a set of m flights that the airline company must provide, determine the minimum number of planes that the company needs to purchase. The airline may add unscheduled flights to move the airplanes around if that would reduce the total number of planes needed.

Input

The first line of input contains two space-separated integers n and m ($1 \leq n, m \leq 500$). The next line contains n space-separated integers $p_1, \ldots, p_n$ ($0 \leq p_i \leq 10^6$).

Each of the next n lines contains n space-separated integers. The jth integer in line $i+2$ is t_{ij} ($0 \leq t_{ij} \leq 10^6$). It is guaranteed that $t_{ii} = 0$ for all i. However, it may be the case that $t_{ij} \neq t_{ji}$ when $i \neq j$.

Each of the next m lines contains three space-separated integers, s_i, f_i, and t_i ($1 \leq s_i, f_i \leq n$, $s_i \neq f_i$, $1 \leq t_i \leq 10^6$), indicating that the airline company must provide a flight that flies out from airport s_i at exactly time t_i, heading directly to airport f_i.

Output

Print, on a single line, a single integer indicating the minimum number of planes the airline company must purchase in order to provide the m requested flights.

Sample Input	**Sample Output**

```
2 2
1 1
0 1
1 0
1 2 1
2 1 1
```
	`2`

Sample Input	Sample Output

```
2 2
1 1
0 1
1 0
1 2 1
2 1 3
```
	`1`

Sample Input	Sample Output

```
5 5
72 54 71 94 23
0 443 912 226 714
18 0 776 347 810
707 60 0 48 923
933 373 881 0 329
39 511 151 364 0
4 2 174
2 1 583
4 3 151
1 4 841
4 3 993
```
	`3`

Problem B
Mastering Mastermind

Mastermind is a two-person code breaking game which works as follows. The first person (the *code maker*) creates a sequence of n colored pegs (with duplicate colors allowed) and hides it from view. This sequence of pegs is the *code*.

The second person (the *code breaker*) has the job of trying to determine the code maker's code and she does so by making a series of guesses. Each guess also consists of n colored pegs. After each guess, the code maker gives the code breaker feedback about how close she is. This feedback consists of two number r and s, where

- $r =$ the number of pegs that are identical in both color and position in the code and the guess, and
- $s =$ the number of remaining pegs that are identical in color but not in the same position.

For example, if the code is BACC (where we use different letters to represent colors) and the guess is CABB, then $r = 1$ (the A in the second position of both the code and the guess) and $s = 2$ (a B and C in the remaining three characters). Note that only one of the B's in the guess will "match" with the single B in the code: once pegs in the code and the guess have been "matched" with each other, they can't be matched with any other pegs.

Your job in this problem is to determine r and s given a code and a guess.

Input

The input is a single line containing a positive integer $n \leq 50$ (the length of the code) followed by two strings of length n — the first of these is the code and the second is the guess. Both code and guess are made up of upper-case alphabetic characters.

Output

Output the values of r and s for the given input.

Sample Input 1	Sample Output 1
`4 BACC CABB`	`1 2`

Sample Input 2	Sample Output 2
`13 ABCDEFGHIJKLM NOPQRSTUVWXYZ`	`0 0`

Problem C

Classy

In his memoir *So, Anyway...*, comedian John Cleese writes of the class difference between his father (who was "middle-middle-middle-lower-middle class") and his mother (who was "upper-upper-lower-middle class"). These fine distinctions between classes tend to confuse American readers, so you are to write a program to sort a group of people by their classes to show the true distinctions.

There are three main classes: upper, middle, and lower. Obviously, upper class is the highest, and lower class is the lowest. But there can be distinctions within a class, so upper-upper is a higher class than middle-upper, which is higher than lower-upper. However, all of the upper classes (upper-upper, middle-upper, and lower-upper) are higher than any of the middle classes.

Within a class like middle-upper, there can be further distinctions as well, leading to classes like lower-middle-upper-middle-upper. When comparing classes, once you've reached the lowest level of detail, you should assume that all further classes are the equivalent to the middle level of the previous level of detail. So upper class and middle-upper class are equivalent, as are middle-middle-lower-middle and lower-middle.

Input

The first line of input contains n ($1 \leq n \leq 1{,}000$), the number of names to follow. Each of the following n lines contains the name of a person (a sequence of 1 or more lowercase letters '`z`'–'`z`'), a colon, a space, and then the class of the person. The class of the person will include one or more modifiers and then the word `class`. The colon, modifiers, and the word `class` will be separated from each other by single spaces. All modifiers are one of `upper`, `middle`, or `lower`. It is guaranteed that the input is well-formed. Additionally, no two people have the same name. Input lines are no longer than 256 characters.

Output

Print the n names, each on a single line, from highest to lowest class. If two people have equivalent classes, they should be listed in alphabetical order by name.

Sample Input

```
5
mom: upper upper lower middle class
dad: middle middle lower middle class
queenelizabeth: upper upper class
chair: lower lower class
unclebob: middle lower middle class
```

Sample Output

```
queenelizabeth
mom
dad
unclebob
chair
```

Sample Input

```
10
rich: lower upper class
mona: upper upper class
dave: middle lower class
charles: middle class
tom: middle class
william: lower middle class
carl: lower class
violet: middle class
frank: lower class
mary: upper class
```

Sample Output

```
mona
mary
rich
charles
tom
violet
william
carl
dave
frank
```

Problem D

Triangle

Determine if it is possible to produce two triangles of given side lengths, by cutting some rectangle with a single line segment, and freely rotating and flipping the resulting pieces.

Input

The input consists of two lines. The first line contains three space-separated positive integers, indicating the desired side lengths of the first triangle. Similarly, the second line contains three space-separated positive integers, denoting the desired side lengths of the second triangle. It is guaranteed that the side lengths produce valid triangles. All side lengths are less than or equal to 100.

Output

Print, on a single line, whether there exists a rectangle which could have been cut to form triangles of the given side lengths. If such a rectangle exists, print `YES`. Otherwise, print `NO`.

Sample Input

```
3 4 5
4 3 5
```

Sample Output

```
YES
```

Sample Input

```
3 4 6
4 6 3
```

Sample Output

```
NO
```

Sample Input

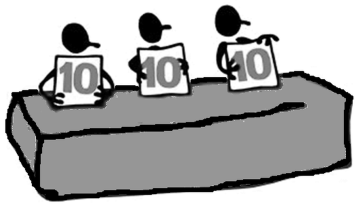
```
39 52 65
25 60 65
```

Sample Output

```
NO
```

Problem E

Excellence



The World Coding Federation is setting up a huge online programming tournament of teams comprised of pairs of programmers. Judge David is in charge of putting teams together from the Southeastern delegation. Every student must be placed on exactly one team of two students. Luckily, he has an even number of students who want to compete, so that he can make sure that each student does compete. However, he'd like to maintain his pristine reputation amongst other judges by making sure that each of the teams he fields for the competition meet some minimum total rating. We define the total rating of a team to be the sum of the ratings of both individuals on the team.

Help David determine the maximum value, X, such that he can form teams, each of which have a total rating greater than or equal to X.

Input

The first line of input contains a single positive integer n ($1 \leq n \leq 10^5$, n is even), the number of students who want to enter the online programming tournament. Each of the following n lines contains one single integer s_i ($1 \leq s_i \leq 10^6$), the rating of student i.

Output

Print, on a single line, the maximum value, X, such that David can form teams where every team has a total rating greater than or equal to X.

Sample Input	Sample Output

```
4
1
2
3
5
```
	`5`

Sample Input	Sample Output

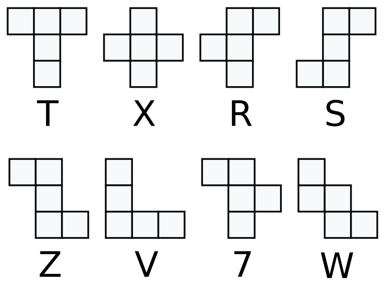
```
2
18
16
```
	`34`

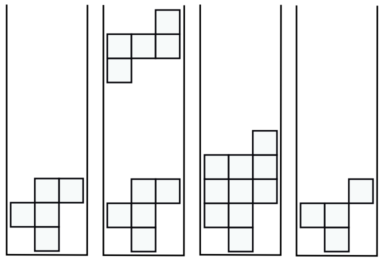
Sample Input	Sample Output

```
4
13
12
19
14
```
	`27`

Problem F

Falling Blocks





FallingBlocks is a Tetris-like arcade game, played on a board with 3 columns and 10 rows according to the following rules. A known, indefinitely repeating sequence of pentominoes (simply called *pieces*) fall down from the top of the board, one at a time. The 8 pieces and their labels are shown above.

The pieces can be rotated freely (by 0, 90, 180 or 270 degrees), but they cannot flipped.

The rules are similar to that of Tetris. The newly introduced piece falls from the top of the board as far as possible until it hits the bottom of the board or an existing block in the board. Then, any rows that are completely full are removed and rows above are moved down, with no further change in the rows themselves.

To illustrate this, consider an empty board, on which an `R` piece, followed by a `Z` piece, falls. If we drop the `R` piece without rotating, we end up with first board shown above. Dropping a rotated `Z` piece on top of it causes two rows to fill. These rows are removed, and the rows above are pushed downward. The final situation is as shown in the rightmost picture. The final position has a "hanging block;" this block does not fall any further at this point.

Unlike Tetris, the top three rows of the board must be completely empty in order to place a piece, i.e., if any of the top three rows is not empty after removing all rows that are full, the game is over.

The score is solely based on the number of pieces played on the board before the game is over. Given the sequence of pieces that repeats indefinitely, determine the maximum number of pieces that can be played.

Below are some example sequences of pieces, followed by explanation.

- `X`: Every drop of an `X` piece leaves two rows filled that cannot be removed by additional `X` pieces. After placing four pieces, we have eight non-empty rows left, so the next `X` piece cannot be placed. So the result is 4.
- `XXXXR`: The `R` piece could be rotated to not overlap the square left in the highest non-empty row, but our rule is that the top three rows must be completely empty to place any piece. So the result is 4.
- `VZV`: Two `V` pieces and a `Z` piece can be placed to clear the board, so this game can go on forever.

Input

The input consists of a single line that contains a single string, representing the sequence of pentominoes. The input sequence contains between 1 and 20 characters.

Output

Print, on a single line, the maximum number of pieces that can be played until no more piece can be placed on the board. If the game can continue indefinitely, print `forever`.

Sample Input	Sample Output
`T`	`forever`

Sample Input	Sample Output
`W`	`8`

Sample Input	Sample Output
`VZ`	`25`

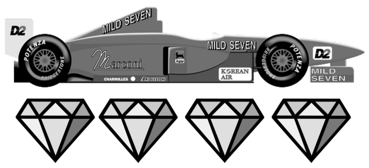
Sample Input	Sample Output
`VR`	`forever`

Sample Input	Sample Output
`VVTRX`	`130`

Sample Input	Sample Output
`XSVTVT`	`forever`

Problem G

Racing Gems



You are playing a racing game. Your character starts at the x axis ($y = 0$) and proceeds up the race track, which has a boundary at the line $x = 0$ and another at $x = w$. You may start the race at any horizontal position you want, as long as it is within the track boundary. The finish line is at $y = h$, and the game ends when you reach that line. You proceed at a fixed vertical velocity v, but you can control your horizontal velocity to be any value between $-v/r$ and v/r, and change it at any time.

There are n gems at specific points on the race track. Your job is to collect as many gems as possible. How many gems can you collect?

Input

The first line of input contains four space-separated integers n, r, w, and h ($1 \le n \le 10^5$, $1 \le r \le 10$, $1 \le w, h \le 10^9$). Each of the following n lines contains two space-separated integers x_i and y_i, denoting the coordinate of the ith gem ($0 \le x_i \le w$, $0 < y_i \le h$). There will be at most one gem per location.

The input does not include a value for v.

Output

Print, on a single line, the maximum number of gems that can be collected during the race.

Sample Input	Sample Output

```
5 1 10 10
8 8
5 1
4 6
4 7
7 9
```
	`3`

Sample Input

```
5 1 100 100
27 75
79 77
40 93
62 41
52 45
```

Sample Output

```
3
```

Sample Input

```
10 3 30 30
14 9
2 20
3 23
15 19
13 5
17 24
6 16
21 5
14 10
3 6
```

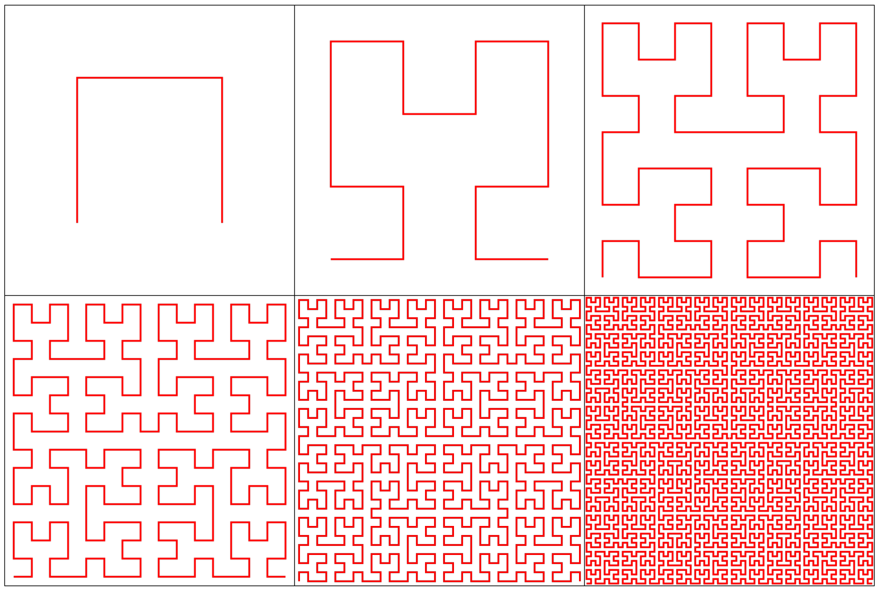
Sample Output

```
4
```

Problem H

Hilbert Sort

In database storage, arranging data items according to a numeric key not only makes it easier to search for a particular item, but also makes better use of a CPU's cache: any segment of data that's contiguous in memory will describe items with similar keys. This is useful if, for instance, we want to access all items whose keys are in some range. Things get more complicated if the keys represent points on a 2D grid, as might happen in a GPS guidance system. If the points (x, y) are sorted primarily by x, breaking ties by y, then points that are adjacent in memory will have similar x coordinates but not necessarily similar y, potentially placing them far apart on the grid. To better preserve distances, we may sort the data along a continuous space-filling curve.



We consider one such space-filling curve called the *Hilbert curve.* The Hilbert curve starts at the origin $(0, 0)$ and finishes at $(S, 0)$, in the process traversing the entire axis-aligned square with corners at $(0, 0)$ and (S, S). It has the following recursive construction: split the square into four quadrants meeting at $(S/2, S/2)$, and recursively fill each of them with a suitably rotated and scaled copy of the full Hilbert curve. First, the lower-left quadrant is filled with a curve going from $(0, 0)$ to $(0, S/2)$. Second, the upper-left quadrant is filled from $(0, S/2)$ to $(S/2, S/2)$. Third, the upper-right quadrant is filled from $(S/2, S/2)$ to $(S, S/2)$. And finally, the lower-right quadrant is filled from $(S, S/2)$ to $(S, 0)$. The Hilbert curve can alternatively be constructed as the mathematical limit of a sequence of curves, the first six of which are shown in the figure.

Given some locations of interest, you are asked to sort them according to when the Hilbert curve visits them. Note that while the curve intersects itself at infinitely many places, e.g., at $(S/2, S/2)$; making S odd guarantees that all integer points are visited just once.

Input

The first line of input contains two space-separated integers n and S ($1 \leq n \leq 200{,}000, 1 \leq S < 10^9$, S is odd). This is followed by n lines. Line $i+1$ describes the ith location of interest by space-separated integers x_i and y_i ($0 \leq x_i, y_i \leq S$) and an identifier string consisting of at most 46 alphanumeric characters ('A'–'Z', 'a'–'z', '0'–'9'). No two locations will share the same position or the same identifier.

Output

Print the n identifier strings, one on each line, Hilbert-sorted according to their positions.

Sample Input

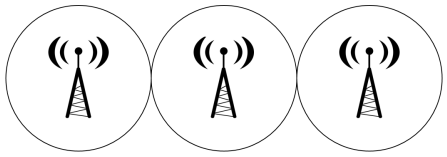
```
14 25
5 5 Honolulu
5 10 PugetSound
5 20 Victoria
10 5 Berkeley
10 10 Portland
10 15 Seattle
10 20 Vancouver
15 5 LasVegas
15 10 Sacramento
15 15 Kelowna
15 20 PrinceGeorge
20 5 Phoenix
20 10 SaltLakeCity
20 20 Calgary
```

Sample Output

```
Honolulu
Berkeley
Portland
PugetSound
Victoria
Vancouver
Seattle
Kelowna
PrinceGeorge
Calgary
SaltLakeCity
Sacramento
LasVegas
Phoenix
```

Problem I

Coverage



A cellular provider has installed n towers to support their network. Each tower provides coverage in a 1 km radius, and no two towers are closer than 1 km to one another. The coverage region of this network is therefore the set of all points that are no more than 1 km away from at least one tower. The provider wants as much of this region as possible to be connected, in the sense that a user at any point within a connected subregion can travel to any other point within the connected subregion without having to exit the subregion. Their current installation of towers may or may not already form a single connected region, but they have the resources to build one more tower wherever they want, including within 1 km of an existing tower.

Given that the provider is able to build one more tower, what is the maximum number of towers (including the one just built) that can be included within a single connected subregion of coverage?

Input

The first line consists of a single integer n ($1 \leq n \leq 5{,}000$), denoting the number of existing towers. Next follow n lines each with 2 space-separated real numbers x_i, y_i ($0 \leq x_i, y_i \leq 10^5$), denoting the location of tower i in km. It is guaranteed that the optimal number of towers will not change even if the coverage radius of all the towers is increased or decreased by one millimeter.

Output

Print, on a single line, a single integer denoting the maximum number of towers that can be within a single connected subregion of the network after installing one additional tower.

Sample Input	Sample Output

```
5
1.0 1.0
3.1 1.0
1.0 3.1
3.1 3.1
4.2 3.1
```
	`6`

Sample Input	Sample Output

```
5
1.0 1.0
3.1 1.0
1.0 3.1
3.1 3.1
10.0 10.0
```
	`5`

Problem J: Cow Confinement

A nearby pasture can be represented as a rectangular grid consisting of 10^6 rows and 10^6 columns. The rows are numbered with integers 1 through 10^6 top to bottom, the columns with integers 1 through 10^6 left to right.

A herd of n cows is scattered through the grid, each cow occupying a unit square. The pasture also contains m dandelion flowers (which cows like), again each occupying a unit square. Finally, the pasture contains p fences, each a rectangle running along the edges of unit squares. Fences *do not intersect or touch*. However, a fence may contain other fences inside the enclosed area.

Due to unfavorable wind conditions, cows can only move in two directions – down or right. Cows can go through squares occupied by other cows or flowers, but cannot cross fences.

For each cow, find the total number of flowers reachable from its present location.

Input

Input contains three blocks – the first block describes fences, the second one flowers and the third one cows.

The first line of the first block contains an integer f ($0 \leq f \leq 200\,000$) – the number of fences. Each of the following f lines contains four integers r_1, c_1, r_2, c_2 ($1 \leq r_1, c_1, r_2, c_2 \leq 10^6$) describing a single fence – r_1 and c_1 are the coordinates (row and column) of the upper-left corner square inside the fence, while r_2 and c_2 are the coordinates of the lower-right corner square inside the fence. No two fences will intersect or touch.

The first line of the second block contains an integer m ($0 \leq m \leq 200\,000$) – the number of flowers. The k-th of the following m lines contains two integers r and c ($1 \leq r, c \leq 10^6$) – the location of the k-th flower. No two flowers will occupy the same location.

The first line of the third block contains an integer n ($1 \leq n \leq 200\,000$) – the number of cows. The k-th of the following n lines contains two integers r and c ($1 \leq r, c \leq 10^6$) – the location of the k-th cow. No two cows will occupy the same location, and no flower and cow will occupy the same location.

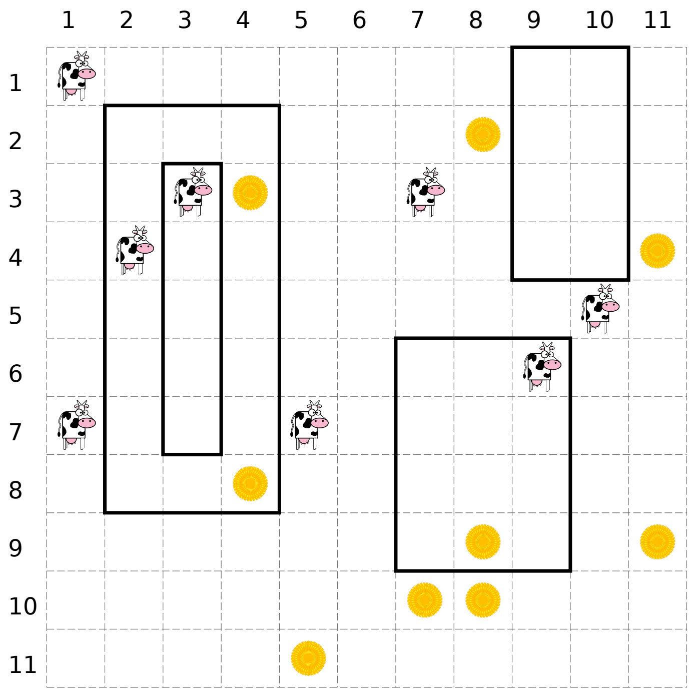
Output

Output should consist of n lines. The k-th line should contain a single integer – the total number of flowers reachable from the location of the k-th cow.

Example

input

```
4
2 2 8 4
1 9 4 10
6 7 9 9
3 3 7 3
9
3 4
8 4
11 5
10 7
10 8
9 8
2 8
4 11
9 11
8
1 1
5 10
6 9
3 7
7 1
4 2
7 5
3 3
```



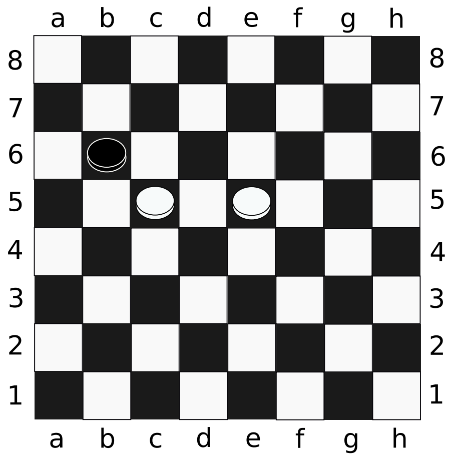
output

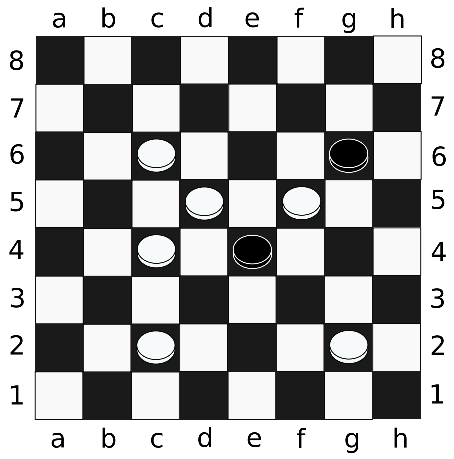
```
5
1
0
1
3
1
3
0
```

Problem K

Checkers

Checkers is played on a $n \times n$ checkerboard (typically n equals 8, 10, or 12, but for this problem, n will range from 2 to 26). The board has squares colored red and black, and all pieces move only on the black squares. The two sides are called "Black" and "White," and their pieces are so colored. The columns of the checkerboard are lettered starting with `a` on the left and increasing alphabetically. The rows are numbered $1, \ldots, n$, starting from the bottom. We refer to each square on the board by its label: the column letter followed by the row number, e.g., `c6`, `z10`, or `b26`. Two sample boards are given below (with additional labels to illustrate the column numbering).





A piece may jump diagonally over a piece of the other color to capture the piece (removing it from the board). In order to perform a jump, the piece that is jumped over must be diagonally adjacent to the piece performing a jump, and the square on the other side of the piece jumped over must be vacant. If such a capture is possible, the jumping piece may continue jumping and capturing pieces of the other color until no more jumps are possible.

For example, in the left sample board, the Black piece at position `b6` can capture both White pieces in a single move by first jumping over the White piece at `c5` (which moves the Black piece to `d4`), and then jumping over the White piece at `e5`, landing at `f6`. In the right sample board, no Black piece can jump any White pieces.

It is Black's turn to move. Given a board of checkers, determine if it is possible for Black to jump all of White's pieces in a single move.

Input

The first line of input contains n ($2 \leq n \leq 26$), the size of the board. The following n lines of n characters describe the board. Red squares (to which no piece can ever move) are labeled with '`.`'. Black squares with no pieces are labeled with '`_`'. Black pieces are labeled with '`B`', and White pieces are labeled with '`W`'.

It is guaranteed that the given board has at least one Black piece and one White piece. Additionally, the board is guaranteed to be well-formed; that is, no piece is on a red square, and the board is correctly colored.

Output

Print, on a single line, the location of the Black piece that can capture all of White's pieces in a single move. If there are multiple such Black pieces, print `Multiple`. If there is no such Black piece, print `None`.

Sample Input

```
8
._._._._
_._._._.
.W._.B._
_.W.W._.
.W.B._._
_._._._.
.W._.W._
_._._._.
```

Sample Output

```
None
```

Sample Input

```
10
._._._._._
_.W.W._._.
._._._._._
_.W.W._._.
._._._._._
_.W.W.W.W.
._._._._._
_.W.W.W.W.
.B.B.B._._
_._._._._.
```

Sample Output

```
d2
```

Problem L

Shuffling Along

Most of you have played card games (and if you haven't, why not???) in which the deck of cards is randomized by shuffling it one or more times.

A *perfect shuffle* is a type of shuffle where the initial deck is divided exactly in half, and the two halves are perfectly interleaved. For example, a deck consisting of eight cards ABCDEFGH (where A is the top card of the deck) would be divided into two halves ABCD and EFGH and then interleaved to get AEBFCGDH. Note that in this shuffle the original top card (A) stays on top — this type of perfect shuffle is called an *out-shuffle*. An equally valid perfect shuffle would start with the first card from the second half and result in EAFBGCHD — this is known as an *in-shuffle*.

While normal shuffling does a good job at randomizing a deck, perfect shuffles result in only a small number of possible orderings. For example, if we perform multiple out-shuffles on the deck above, we obtain the following:

$$\text{ABCDEFGH} \rightarrow \text{AEBFCGDH} \rightarrow \text{ACEGBDFH} \rightarrow \text{ABCDEFGH} \rightarrow \cdots$$

So after 3 out-shuffles, the deck is returned to its original state. A similar thing happens if we perform multiple in-shuffles on an 8-card deck, though in this case it would take 6 shuffles before we get back to where we started. With a standard 52 card deck, only 8 out-shuffles are needed before the deck is returned to its original order (talented magicians can make use of this result in many of their tricks). These shuffles can also be used on decks with an odd number of cards, but we have to be a little careful: for out-shuffles, the first half of the deck must have 1 more card than the second half; for in-shuffles, it's the exact opposite. For example, an out-shuffle on the deck ABCDE results in ADBEC, while an in-shuffle results in CADBE.

For this problem you will be given the size of a deck and must determine how many in- or out-shuffles it takes to return the deck to its pre-shuffled order.

Input

The input consists of one line containing a positive integer $n \leq 1000$ (the size of the deck) followed by either the word `in` or `out`, indicating whether you should perform in-shuffles or out-shuffles.

Output

For each test case, output the case number followed by the number of in- or out-shuffles required to return the deck to its original order.

Sample Input 1	Sample Output 1
`8 out`	`3`

Sample Input 2

```
8 in
```

Sample Output 2

```
6
```

Sample Input 3

```
52 out
```

Sample Output 3

```
8
```

Sample Input 4

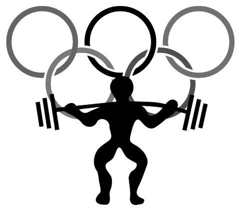
```
53 out
```

Sample Output 4

```
52
```

Problem M

Olympics



The weightlifting event is up next at the Olympic games, and it's time to impress your fans! To accomplish your sequence of lift attempts, you have a constant strength S and a decreasing energy reserve E. For each attempt, you may choose any positive (not necessarily integer) weight W. If $S \geq W$, the lift succeeds and your energy goes down by E_{succ}. If $S < W$, the lift fails and your energy goes down by E_{fail}. You may continue attempting lifts as long as $E > 0$. If at any point $E \leq 0$, you can make no further attempts. Your score is the maximum weight in kg that you successfully lift, or 0 if all attempts failed.

Ideally, you should lift at exactly your strength limit. However, you do not know your strength. You only know that you can definitely lift the 25 kg Olympic bar, and that the maximum conceivable lift adds 100 kg on each side for a total of 225 kg. How close to an optimal score can you guarantee? That is, what's the smallest d for which you can ensure a score of at least $S - d$?

Input

The input consists of a single line containing three space-separated integers E, E_{succ}, and E_{fail} $(1 \leq E, E_{\text{succ}}, E_{\text{fail}} \leq 10^7)$.

Output

Print, on a single line, the minimum d, rounded and displayed to exactly 6 decimal places.

Sample Input	Sample Output
`1 3 3`	`112.500000`

Sample Input	Sample Output
`12 3 3`	`13.333333`

Sample Input	Sample Output
`3000 2 3`	`0.000000`