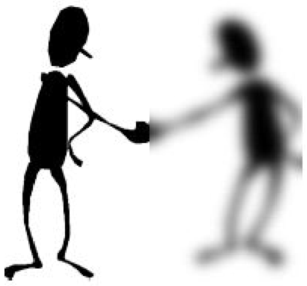
Problem A. Blur

Source file name:	blur.c, blur.cpp, blur.java, blur.py
Input:	`Standard`
Output:	`Standard`



You have a black and white image that is w pixels wide and h pixels high. You decide to represent this image with one number per pixel: black is 0, and white is 1. Your friend asks you to blur the image, resulting in various shades of gray. The way you decide to blur the image is as follows: You create a new image that is the same size as the old one, and each pixel in the new image has a value equal to the average of the 9 pixels in the 3×3 square centered at the corresponding old pixel. When doing this average, wrap around the edges, so the left neighbor of a leftmost pixel is in the rightmost column of the same row, and the top neighbor of an uppermost pixel is on the bottom in the same column. This way, the 3×3 square always gives you exactly 9 pixels to average together. If you want to make the image blurrier, you can take the blurred image and blur it again using the exact same process.

Given an input image and a fixed number of times to blur it, how many distinct shades of gray does the final image have, if all the arithmetic is performed exactly?

Warning: Floating point numbers can be finicky; you might be surprised to learn, for example, that $2/9+5/9$ *may not equal* $3/9 + 4/9$ *if you represent the fractions with floating point numbers! Can you figure out how to solve this problem without using floating point arithmetic?*

Input

The first line of input contains three space-separated integers w, h, and b ($3 \leq w, h \leq 100$, $0 \leq b \leq 9$), denoting the width and height of the image, and the number of times to blur the image, respectively. The following h lines of w space-separated integers describe the original image, with each integer being either 0 or 1, corresponding to the color of the pixel.

Output

Output, on a single line, a single integer equal to the number of distinct shades of gray in the final image.

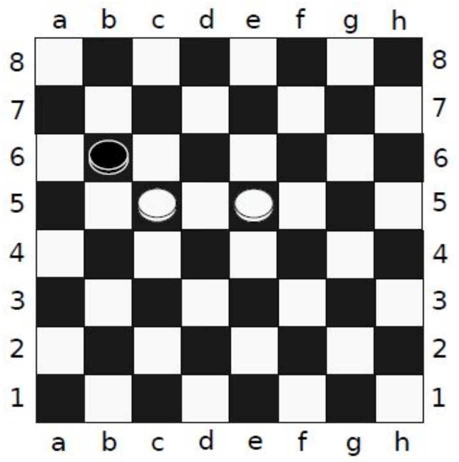
Example

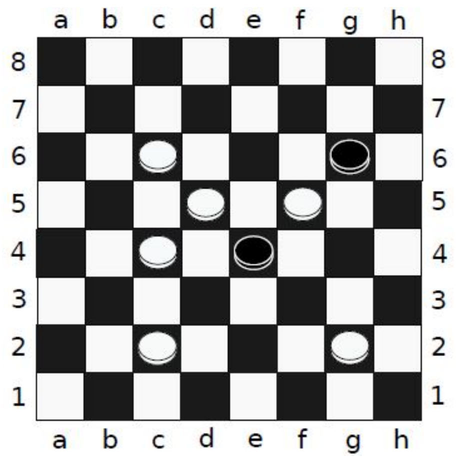
Input	Output
`5 4 1`	`3`
`0 0 1 1 0`	
`0 0 1 1 0`	
`0 0 1 1 0`	
`0 0 1 1 0`	
`3 3 2`	`1`
`1 0 0`	
`0 1 0`	
`0 1 0`	

Problem B. Checkers

Source file name:	checkers.c, checkers.cpp, checkers.java, checkers.py
Input:	`Standard`
Output:	`Standard`

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 `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`.

Example

Input	Output
`8`	`None`
`._._._._`	
`_._._._.`	
`.W._.B._`	
`_.W.W._.`	
`.W.B._._`	
`_._._._.`	
`.W._.W._`	
`_._._._.`	
`10`	`d2`
`._._._._._`	
`_.W.W._._.`	
`._._._._._`	
`_.W.W._._.`	
`._._._._._`	
`_.W.W.W.W.`	
`._._._._._`	
`_.W.W.W.W.`	
`.B.B.B._._`	
`_._._._._.`	

Problem C. Class Time

Source file name:	class.c, class.cpp, class.java, class.py
Input:	`Standard`
Output:	`Standard`

It's the first day of class! Tom is teaching class and first has to take attendance to see who is in class. He needs to call the students' names in alphabetical order by last name. If two students have the same last name, then he calls the students with that same last name in alphabetical order by first name. Help him!

Input

The first line of input contains an integer n ($1 \leq n \leq 100$), the number of students in Tom's class. Each of the following n lines contains the name of a single student: first name, followed by a single space, then last name. The first and last name both start with an uppercase letter ('`A`'-'`Z`') and then be followed by one or more lowercase letters ('`a`'-'`z`'). The first and last name of each student is no more than 10 letters long each.

It is guaranteed that no two students have exactly the same name, though students may share the same first name, or the same last name.

Output

Output n lines, the names of the students as Tom calls them in the desired order.

Example

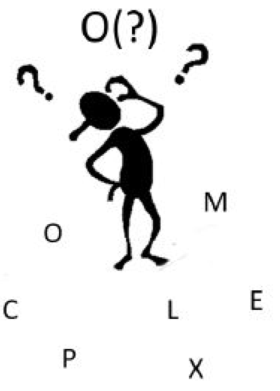
Input	Output

`3`
`John Adams`
`Bob Adam`
`Bob Adams`
	`Bob Adam`
	`Bob Adams`
	`John Adams`

`1`
`Coursera Educators`
	`Coursera Educators`

Problem D. Complexity

Source file name:	complexity.c, complexity.cpp, complexity.java, complexity.py
Input:	`Standard`
Output:	`Standard`



Define the *complexity* of a string to be the number of distinct letters in it. For example, the string `string` has complexity 6 and the string `letter` has complexity 4.

You like strings which have complexity either 1 or 2. Your friend has given you a string and you want to turn it into a string that you like. You have a magic eraser which will delete one letter from any string. Compute the minimum number of times you will need to use the eraser to turn the string into a string with complexity at most 2.

Input

The input consists of a single line that contains a single string of at most 100 lowercase ASCII letters (`'a'`-`'z'`).

Output

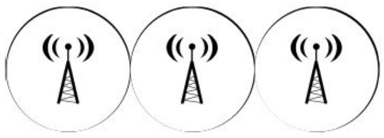
Print, on a single line, the minimum number of times you need to use the eraser.

Example

Input	Output
string	4
letter	2
aaaaaa	0
uncopyrightable	13
ambidextrously	12
assesses	1
assassins	2

Problem E. Coverage

Source file name:	coverage.c, coverage.cpp, coverage.java, coverage.py
Input:	`Standard`
Output:	`Standard`



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 ($1 \leq n \leq 5*10^3$), 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.

Example

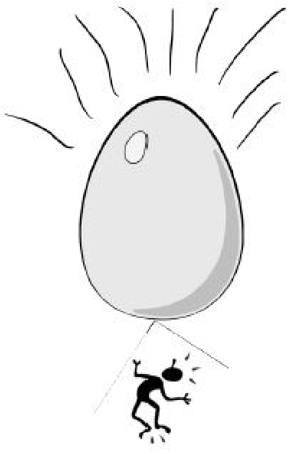
Input	Output

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

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

Problem F. Egg Drop

Source file name:	egg.c, egg.cpp, egg.java, egg.py
Input:	`Standard`
Output:	`Standard`



There is a classic riddle where you are given two eggs and a k-floor building and you want to know the highest floor from which you can drop the egg and not have it break.

It turns out that you have stumbled upon some logs detailing someone trying this experiment! The logs contain a series of floor numbers as well as the results of dropping the egg on those floors. You need to compute two quantities–the lowest floor that you can drop the egg from where the egg could break, and the highest floor that you can drop the egg from where the egg might not break.

You know that the egg will not break if dropped from floor 1, and will break if dropped from floor k. You also know that the results of the experiment are consistent, so if an egg did not break from floor x, it will not break on any lower floors, and if an egg did break from floor y, it will break on all higher floors.

Input

The first line of input contains two space-separated integers n and k ($1 \leq n \leq 100, \quad 3 \leq k \leq 100$), the number of egg drops and the number of floors of the building, respectively. Each of the following n lines contains a floor number and the result of the egg drop, separated by a single space. The floor number will be between 1 and k, and the result will be either `SAFE` or `BROKEN`.

Output

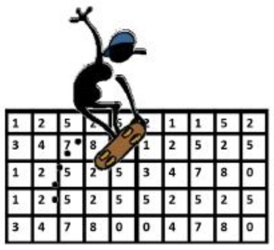
Print, on a single line, two integers separated by a single space. The first integer should be the number of the lowest floor from which you can drop the egg and it could break and still be consistent with the results. The second integer should be the number of the highest floor from which you can drop the egg and it might not break.

Example

Input	Output
2 10	5 6
4 SAFE	
7 BROKEN	
3 5	5 4
2 SAFE	
4 SAFE	
3 SAFE	
4 3	2 1
2 BROKEN	
2 BROKEN	
1 SAFE	
3 BROKEN	

Problem G. Grid

Source file name:	grid.c, grid.cpp, grid.java, grid.py
Input:	`Standard`
Output:	`Standard`



You are on the top left square of an $m \times n$ grid, where each square on the grid has a digit on it. From a given square that has digit k on it, a *move* consists of jumping exactly k squares in one of the four cardinal directions. What is the minimum number of moves required to get from the top left corner to the bottom right corner?

Input

The first line of input contains two space-separated positive integers m and n ($1 \leq m,\ n \leq 500$). It is guaranteed that at least one of m and n is greater than 1. The next m lines each consists of n digits, describing the $m \times n$ grid. Each digit is between 0 and 9.

Output

Print, on a single line, a single integer denoting the minimum number of moves needed to get from the top-left corner to the bottom-right corner. If it is impossible to reach the bottom-right corner, print `IMPOSSIBLE` instead.

Example

Input	Output

2 2
11
11
	2

2 2
22
22
	IMPOSSIBLE

5 4
2120
1203
3113
1120
1110
	6

Problem H. Millionaire

Source file name:	millionaire.c, millionaire.cpp, millionaire.java, millionaire.py
Input:	`Standard`
Output:	`Standard`

Congratulations! You were selected to take part in the TV game show *Who Wants to Be a Millionaire*! Like most people, you are somewhat risk-averse, so you might rather take \$250,000 than a 50% chance of winning \$1,000,000. On the other hand, if you happen to already be rich, then you might as well take a chance on the latter. Before appearing on the show, you want to devise a strategy to maximize the expected *happiness* derived from your winnings.

More precisely, if your present net worth is W dollars, then winning v dollars gives you $\ln(1+v/W)$ units of happiness. Thus, the game's *expected happiness* is $\Sigma_v P(v)\ln(1+v/W)$, where $P(v)$ is the probability that you'll win v dollars, and the summation is taken over all possible values of v. Since happiness units are too abstract, you will be asked to measure the value of the game in dollars. That is, compute D such that a guaranteed payout of D dollars makes you as happy as a chance on the show, assuming optimal play.

On the show, you will be presented with a series of questions on trivia, each associated with a prize value of v_i dollars. Your analysis of past episodes reveals that if you attempt the i-th question, your chances of being correct are p_i.

After answering correctly, you may choose to continue or to quit. If you quit, you win the value of the last correctly answered question; otherwise, the game continues and you must attempt the next question. If you correctly answer all the questions, you walk away with the value of the last question.

If you answer a question incorrectly, however, the game ends immediately and you win the value of the last correctly answered question that is labeled as `safe`, or nothing if you never solved a `safe` question.

For example, the game in the first sample input is worth $0.5\ln(1+5000/4000) \approx 0.405$ units of happiness. Getting \$2,000 would likewise grant $\ln(1+2000/4000) \approx 0.405$ happiness.

Input

The first line of input contains two space-separated integers n and W ($1 \le n \le 10^5, 1 \le W \le 10^6$). Line $i+1$ describes the i-th question. It starts with a string, which is one of `safe` or `unsafe`, indicating whether the i-th question is safe or not. The string is followed by a real number p_i and an integer v_i ($0 \le p_i \le 1,\ 1 \le v_i < v_{i+1} \le 10^6$).

Output

Print, on a single line, a \$ sign immediately followed by D, rounded and displayed to exactly two decimal places. See the samples for format clarification.

Example

Input	Output
1 4000 unsafe 0.5 5000	$2000.00
4 4000 unsafe 1 2000 safe 0.4 5000 unsafe 0.75 10000 safe 0.05 1000000	$2316.82
2 4000 safe 0.003 1 safe 0.03 10	$0.00

Problem I. Racing Gems

Source file name:	gems.c, gems.cpp, gems.java, gems.py
Input:	`Standard`
Output:	`Standard`

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 ($1 \leq n \leq 10^5$, $1 \leq r \leq 10$, $1 \leq w,\ h \leq 10^9$). Each of the following n lines contains two space-separated integers x_i and y_i, denoting the coordinate of the i-th gem ($0 \leq x_i \leq w,\ 0 < y_i \leq h$). 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.

Example

Input	Output
5 1 10 10	3
8 8	
5 1	
4 6	
4 7	
7 9	
5 1 100 100	3
27 75	
79 77	
40 93	
62 41	
52 45	
10 3 30 30	4
14 9	
2 20	
3 23	
15 19	
13 5	
17 24	
6 16	
21 5	
14 10	
3 6	

Problem J. Surf

Source file name:	surf.c, surf.cpp, surf.java, surf.py
Input:	`Standard`
Output:	`Standard`

Now that you've come to Florida and taken up surfing, you love it! Of course, you've realized that if you take a particular wave, even if it's very fun, you may miss another wave that's just about to come that's even more fun. Luckily, you've gotten excellent data for each wave that is going to come: you'll know exactly when it will come, how many *fun points* you'll earn if you take it, and how much time you'll have to wait before taking another wave. (The wait is due to the fact that the wave itself takes some time to ride and then you have to paddle back out to where the waves are crashing.) Obviously, given a list of waves, your goal will be to maximize the amount of fun you could have.

Consider, for example, the following list of waves:

Minute	Fun points	Wait time
2	80	9
8	50	2
10	40	2
13	20	5

In this example, you could take the waves at times 8, 10 and 13 for a total of 110 fun points. If you take the wave at time 2, you can't ride another wave until time 11, at which point only 20 fun points are left for the wave at time 13, leaving you with a total of 100 fun points. Thus, for this input, the correct answer (maximal number of fun points) is 110.

Given a complete listing of waves for the day, determine the maximum number of fun points you could earn.

Input

The first line of input contains a single integer n ($1 \leq n \leq 3*10^5$), representing the total number of waves for the day. The i-th line ($1 \leq i \leq n$) that follows will contain three space separated integers: m_i, f_i, and w_i, ($1 \leq m_i, f_i, w_i \leq 10^6$), representing the time, fun points, and wait time of the i-th wave, respectively. You can ride another wave occurring at exactly time $m_i + w_i$ after taking the i-th wave. It is guaranteed that no two waves occur at the same time. The waves may not be listed in chronological order.

Output

Print, on a single line, a single integer indicating the maximum amount of fun points you can get riding waves.

Example

Input	Output

4
8 50 2
10 40 2
2 80 9
13 20 5
	110

10
2079 809484 180
8347 336421 2509
3732 560423 483
2619 958859 712
7659 699612 3960
7856 831372 3673
5333 170775 1393
2133 989250 2036
2731 875483 10
7850 669453 842
	3330913

Problem K. You Shall Pass

Source file name:	passing.c, passing.cpp, passing.java, passing.py
Input:	`Standard`
Output:	`Standard`

Students failing COT 3100 (Introduction to Discrete Structures) has become a large concern for the professors at UCF. Matt and Sean have compiled a large amount of data, and after extensively analyzing it, they have generated some interesting probabilities. For each student, there are two basic probabilities known: the probability they will pass Matt's class and the probability they will pass Sean's class. Also known is a large table of probabilities that a student will pass based on being in the same class as another student. When a pair of students are in the same class, they may get together to form a study group, increasing their probability of passing the class. For each pair of students, i, j, a value a_{ij} is known. If students i and j are in the same class then student i is a_{ij} more likely to pass due to a study group. That is, i's probability of passing increases by a_{ij}. For example, if the initial probability of student i passing is 0.4 and $a_{ij} = 0.2$, then i's probability of passing with j in her class becomes 0.6 $(0.4 + 0.2 = 0.6)$.

Now Matt and Sean are trying to find a way to split the students into two classes such that the expected number of people passing is maximized.

Given the initial passing probabilities of the students and the passing probabilities from study groups, you are to determine the maximum expected number of passing students obtainable by splitting the students into two classes. Note that every student will be in exactly one class but one class may be empty.

Input

The first input line contains a positive integer, g, indicating the number of semesters to check. The data for each semester starts with an integer n $(2 \leq n \leq 50)$, which is the number of students who sign up for Discrete (COT 3100) in that semester. The following two input lines contain n decimal values each. The i-th value of the first line represents the probability of the i-th student passing Matt's class, and the i-th value of the second line represent the probability of the i-th student passing Sean's class. This is followed by n lines. The i-th line contains n non-negative decimal numbers. The j-th number on each line is a_{ij} as specified above.

For any given class configuration, assume that at no time a student will have a probability of passing higher than 1 or less than 0 (even after considering study groups). Each decimal number in input will be of the form "#.##", where "#" denotes a digit from 0 to 9, inclusive (i.e., each input value will be given to exactly two decimal places).

Output

For each semester, output a single decimal value representing the maximum expected number of students passing. Round the answers to two decimal places (e.g., 1.234 rounds to 1.23 and 1.235 rounds to 1.24).

Example

Input	Output
2	1.50
2	2.65
0.75 0.25	
0.25 0.75	
0.00 0.20	
0.20 0.00	
3	
0.20 0.60 0.95	
0.40 0.40 0.95	
0.00 0.00 0.55	
0.00 0.00 0.35	
0.00 0.00 0.00	

Problem L. Turing's Challenge

Source file name:	turing.c, turing.cpp, turing.java, turing.py
Input:	`Standard`
Output:	`Standard`

Knuth was looking through some of Turing's memoirs and found a rather interesting challenge that Turing had left for one of his successors. Naturally, Knuth has slyly decided to ask you, his best student, to write a computer program to solve the challenge, but plans on taking credit for the work. Since you know that co-authoring a paper with Knuth is to computer scientists what co-authoring a paper with Erdos is to mathematicians, you've decided to take the bait. Help Knuth solve Turing's problem!

The challenge is as follows:

Given positive integer values for X and N, define the set T as follows:

$T = \{T_i \mid 1 \leq i \leq N + 1\}$, where $T_i = \binom{N}{i-1} X^{i-1}$

The goal of the challenge is to pick a set S of maximal sum with $S \subseteq \{i \mid 1 \leq i \leq N + 1\}$, such that $\prod_{i \in S} T_i \equiv 2(mod\ 4)$.

In other words, we seek a subset of terms in the binomial expansion of $(1 + X)^N$ such that the product of the terms leaves a remainder of 2 when divided by 4 and the sum of the *indices* of those terms is maximal.

The goal of Turing's challenge is to determine this maximal sum.

As an example, consider $X = 3$ and $N = 5$. The corresponding terms are $T_1 = 1$, $T_2 = 15$, $T_3 = 90$, $T_4 = 270$, $T_5 = 405$, and $T_6 = 243$.

The product, $T_1T_2T_4T_5T_6 = 1 \times 15 \times 270 \times 405 \times 243 = 398580750 \equiv 2(mod\ 4)$, thus the solution to this specific challenge is $1 + 2 + 4 + 5 + 6 = 18$, since no other product of terms with a higher sum of indices is congruent to $2(mod\ 4)$.

Input

The first input line contains a positive integer, q $(1 \leq q \leq 500)$, indicating the number of queries. Each of the next q lines will contain a pair of space-separated integers, where the first integer is X $(1 \leq X < 2^{31})$, and the second integer is N $(1 \leq N < 2^{31})$, for that query.

Output

For each query, output on a line by itself, the desired maximal sum of indices. If no such subset of terms exists, output 0 instead.

Example

Input	Output

3
3 5
1 4
4 6
	18
	9
	0