icpc International Collegiate Programming Contest

The 2025 ICPC North America Southern California Regional Contest

Official Problem Set

2025/2026 SOUTHERN CALIFORNIA REGIONAL INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 1
GPA Computation

Lydia wants to know if she is valedictorian!

Lydia's school computes the grade point averages of its students as follows—for each class that a student takes, they get assigned a letter grade from A to E. An A is worth 4 points, a B is worth 3 points, a C is worth 2 points, a D is worth 1 point, and an E is worth no points. The *unweighted* grade point average is therefore derived by computing the sum of these point values and dividing by the number of classes Lydia took.

To compute the *weighted* grade point average, each of the classes is assigned a tier from 1 to 3. If a student gets an A, B, or C in a tier 1 class, they get an additive bonus of 0.05 points. If a student gets an A, B, or C in a tier 2 class, they get an additive bonus of 0.025 points. These are the only ways to get additive bonuses. The *weighted* grade point average is computed by adding together all the additive bonuses to the *unweighted* grade point average.

Given Lydia's transcript, compute her weighted grade point average.

The first line of input contains a single integer, n ($1 \leq n \leq 50$), giving the number of grades on the transcript. Each of the next n lines contains a two-character string, the letter grade for one of the classes Lydia took followed by the tier of the class. It is guaranteed the first character will be in `ABCDE` and the second character will be in `123`.

Print a line with a single real number, Lydia's weighted grade point average. Your answer will be considered correct if it has absolute or relative error at most 10^{-6} from the judges' reference value.

Sample Input

```
5
A1
B2
C3
D1
E2
```

Output for the Sample Input

```
2.075
```

2025/2026 SOUTHERN CALIFORNIA REGIONAL INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 2
Rubik's Cube

This is an interactive problem.

You are given a Rubik's Cube—a $3 \times 3 \times 3$ cube, where each side has a 3×3 grid of cells. Each cell is colored in one of 6 colors: white, yellow, orange, red, green or blue. It is not a traditional Rubik's Cube, though—there can be more or less than 9 squares of each color and the cube may not be solvable. You can only see the front side and your job is to find out the colors of all the squares. To do that, you can rotate the side walls of the cube. You can rotate every side by 90 degrees (any direction). After each rotation, you can look at the front side again. Using at most 100 moves, print the full starting configuration of the cube. Note that determining the middle squares of all sides but the front one is impossible, and you do not have to guess them.

At the beginning and after every move you will receive a 3×3 front side of the cube. Each cell will be in one of 6 colors: W (white), Y (yellow), O (orange), R (red), G (green), or B (blue). If you want to rotate a side, print a line with one letter representing the side you want to rotate: F (front), B (back), U (up), D (down), L (left), or R (right). Printing the letter as is will rotate it clockwise if you were to have this side as your front. Adding a prime (an apostrophe, e. g. "`F'`") will rotate it counterclockwise. Figure 1 shows an example of what each rotation does, assuming we are at the start of Sample Interaction 1.

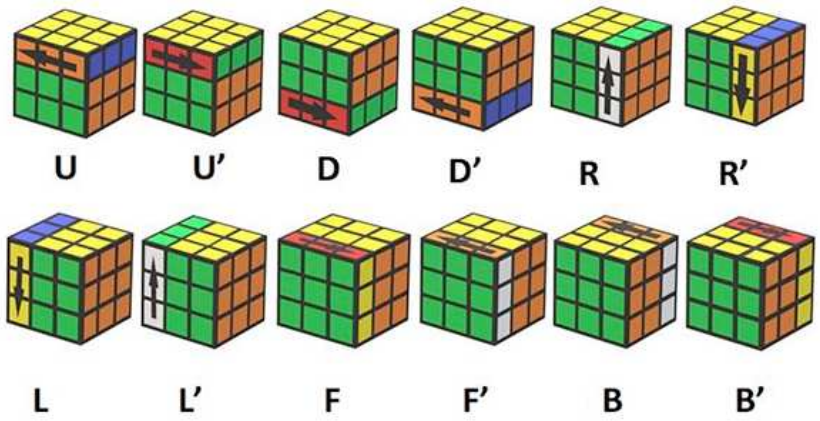

Figure 1. Possible cube side wall rotations, with the green side on the front. (The on-line PDF is in color.)

Once your program has determined the starting configuration of the cube, print a line with one letter: "`A`". After that, output the starting configuration of the cube. Print the starting configuration in the cube projection format, as demonstrated in the example interaction. In this projection, the front of the cube is the center of the "t". The top side is above the center, the left side is left of center, the right side is right of center, below the center is the cube bottom and the bottom of the "t" is the rear of the cube. A diagram of such a projection (formally known as a cube net) is shown in Figure 2. For all middle squares of the sides that are not visible, print "`?`". Make sure that leading spaces are printed correctly, so the output visually looks like a "t" projection. Trailing spaces do not matter.

Note: While the sample interactions are formally correct, they clearly guess the final configurations, as it is not possible to determine all squares in only three or four moves.

Problem 2
Rubik's Cube (continued)

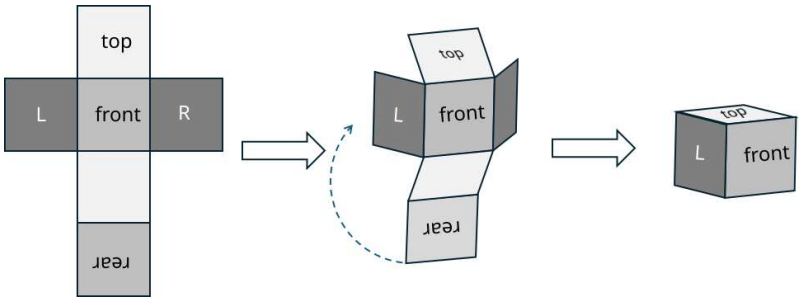

Figure 2. Diagram demonstrating how a T-projection cube net folds.

Sample Interaction 1

Read	*Write*
GGG	
GGG	
GGG	
	U
OOO	
GGG	
GGG	
	R'
OOY	
GGY	
GGY	
	F
GGO	
GGO	
YYY	
	A
	YYY
	Y?Y
	YYY
	RRRGGGOOO
	R?RGGGO?O
	RRRGGGOOO
	WWW
	W?W
	WWW
	BBB
	B?B
	BBB

Sample Interaction 2

Read	*Write*
BBB	
WWW	
WWW	
	R
BBW	
WWW	
WWW	
	R
BBW	
WWW	
WWB	
	R
BBW	
WWW	
WWW	
	R
BBB	
WWW	
WWW	
	A
	WWW
	W?W
	WWW
	BBBBBBBBB
	W?WWWWW?W
	WWWWWWWWW
	WWW
	W?W
	WWW
	WWW
	W?W
	BBB

Problem 2
Rubik's Cube (still continued)

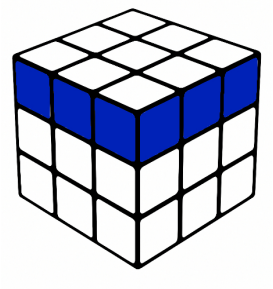

Figure 3. Cube from Sample Interaction 2.

Judging Notes on Interactive Problems

- In most programming environments, program output is buffered to speed up I/O operations. With interactive problems, it is crucial to make sure the output is actually sent from your program and not simply stored in internal buffers. This typically means flushing the output buffers after each line of output (that is, after each newline character) as follows:
  - In C (or C++ using *cstdio*), you can use `fflush(stdout)`.
  - A C++ output stream is flushed automatically each time you write the `endl` manipulator. When using other means or if you want to be sure, call `cout.flush()`.
  - In Java and Kotlin, the `System.out` stream has so-called "auto-flush" functionality and its buffer is therefore flushed automatically with each newline character. When using other streams or if you want to be sure, invoke the `flush()` method of the stream.
  - In Python, you can use `sys.stdout.flush()`.
- Interactive problems are judged in a way similar to other problems, but there are some differences:
  - When your program attempts to read data, it will wait until more data are available or until the judge program terminates the input (unless you read in a non-blocking way, which is beyond the scope of these notes). Thus, if your program attempts to read more input than can currently be provided (e. g., because you forgot to flush your previous output, or because of some other reason), then the program will stall indefinitely and your submission will get Time Limit Exceeded.
  - As usual, the judgment given to an incorrect submission is the first error discovered, but this does not always mean exactly the same thing as for traditional problems. For instance, if your submission to a traditional problem is too slow on a particular test case, you would get Time Limit Exceeded on that test case. With an interactive problem, the judgment may be Wrong Answer if the solution exhibits an incorrect answer before it runs out of time.
  - The time limit for an interactive problem is how much time your submission may spend; the time spent by the judge program is not counted towards this.
  - Because of timing between the interactive judge program and your submission, judgments for incorrect submissions to interactive problems are not necessarily deterministic. We can guarantee the following: a judgment of Wrong Answer means that your submission produced incorrect output, and a judgment of Run Time Error means that your submission returned a non-zero exit code. If your submission does both, you may receive either judgment.

2025/2026 SOUTHERN CALIFORNIA REGIONAL INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 3
Snakes and Ladders

Do you remember the classic childhood game of Snakes and Ladders? The game is played on a 10×10 board with square fields numbered 1 through 100 in a serpentine pattern. Figure 1 shows an example of such a board.

The player starts at Square 1. The objective is to reach Square 100, by moving according to the roll of a six-sided die. Each square is either an ordinary square, the head of a snake, or the start of a ladder.

1. The player rolls the die and moves forward that many squares. From Square 1, a roll of 5 would move the player to Square 6.
2. If the square the player lands on is the start of a ladder, the player must climb to the end of the ladder (such as Square 13 to Square 46 in Figure 1). Ladders always move a player to a square with a higher number.
3. If the square the player lands on is the head of a snake, the player must slide down to the tail of the snake (such as Square 27 to Square 5 in Figure 1). Snakes always move a player to a square with a lower number.
4. If the player lands on any other square, it is an ordinary square and the player remains on that square until their next turn. Squares with snake tails, ladder ends, or that are crossed by ladder(s) or snake(s) are ordinary squares.
5. The first player to land on or cross Square 100 wins the game. For example, starting from Square 98, any roll of 2 or higher will win.

Multiple ladder ends and/or snake tails can reach the same square. Squares 1 and 100 are always ordinary squares. All game boards number squares the same way, but vary the positions of snakes and ladders. Any given game board may contain zero or more snakes, and zero or more ladders.

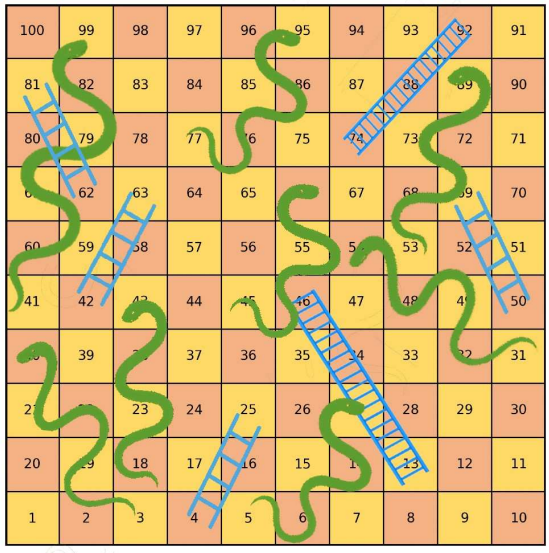

Figure 1. Example Snakes and Ladders board corresponding to Sample Input 1.

As every child knows, landing on a snake and sliding down is the most embarrassing action in the game. Therefore, you want to play the game without doing so even once. Even more, you would like to win the game in as few dice throws as possible. This brings up a question: what is the fastest possible win while avoiding all the snakes, assuming you always roll the best possible number? If a snake cannot be avoided, you want to know that as well.

Problem 3
Snakes and Ladders (continued)

Input to your program is 10 lines, each line representing a row of the game board. The first line corresponds to the top row of the board, and the last line corresponds to the bottom row. Each row contains 10 strings separated from each other by single spaces. Each string represents a square on the row. The string can be either a single period, meaning the square is ordinary; "`S`x", meaning the square has the head of a snake with a tail on square x; or "`L`x", meaning the square has the start of a ladder that ends on square x.

Your program is to print a line containing the string "`Win in` x", where x is the minimum number of moves to win without stepping on a snake even once. If stepping on a snake cannot be avoided, print a line containing the string "`Snakes awaiting`".

Sample Input 1

```
. S41 . . . S77 . . . .
. . . . . . . . S53 .
. . . . . . L92 . . .
. L81 . . . S45 . . . .
. . . . . . S31 . . .
. L63 S18 . . . . . . L69
S3 . . . . . . . . . .
. . . . . . S5 . . .
. . . . . . . L46 . .
. . . L25 . . . . . .
```

Output for Sample Input 1

```
Win in 6
```

Sample Input 2

```
. S41 . . . S77 . . . .
. . . . . . . . S53 .
. . . . . . L92 . . .
. L81 . . . S45 . . . .
. . . . . . S31 . . .
. L63 S18 . . . . . . L69
S3 . . . . . . . . . .
. . . . . . S5 . . .
. . . . . . . L46 . .
. S1 S1 S1 S1 S1 S1 . . .
```

Output for Sample Input 2

```
Snakes awaiting
```

2025/2026 SOUTHERN CALIFORNIA REGIONAL INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 4
Alumni Connections

Some time ago the Swamp County College Alumni Association set up a social network for alumni. This has resulted in the usual friend connection graph. A tiny example of such a graph is shown in Figure 1.

Your assignment is to examine some clustering statistics of the graph. Alumni are anonymised and are represented by integers $0 \leq i < 10000$. The graph is presented as a series of edges, a pair of integers representing the alumni the edge connects. Some edges may be duplicates; because the graph is undirected, (0 1) is a duplicate of (1 0). No alumnus connects to themselves.

We are interested in two types of connections among three alumni: closed triples and open triples. A closed triple is when the immediate friends of an alumnus are also immediate friends of each other. In figure 1, 770 and 528 are immediate friends of 875, and they are immediate friends of each other, so this is a closed triple. On the other hand, 770 and 920 are also immediate friends of 875, but they are not immediate friends of each other, so this is an open triple.

In Figure 1, the closed triples are (528 770 875) and (528 920 875). The open triples are (528 920 770), (875 920 770), (875 770 336), (875 920 336), (528 875 336), and (974 879 303).

The first line of input contains c, the number of edge lines to follow, with one edge per line ($1 \leq c \leq 5000$). Each edge line has two integers n and m which are the alumnus numbers separated by whitespace. They may be preceded and followed by whitespace.

Your program is to print one line with two integers: the number of closed triples followed by the number of open triples separated by whitespace.

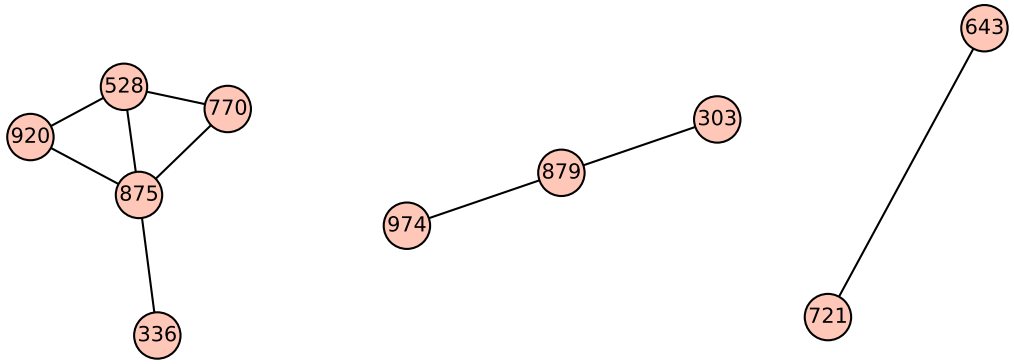

Figure 1. Friend connection graph corresponding to the Sample Input.

Problem 4
Alumni Connections (continued)

Sample Input

```
11
770 528
920 875
528 875
  770 875
 920 528
875 336
643 721
303 879
974 879
875 770
920 528
```

Output for the Sample Input

```
2 6
```

2025/2026 SOUTHERN CALIFORNIA REGIONAL INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 5
Pair Linked Mokepon

The popular game of Mokepon can be represented as a list of n stations with special items located at them and a player beginning at station 1. Each item has an identifier id ($2 \leq id \leq n$, and no two items may have the same identifier), and the player is required to have the item with identifier i to be able to move to station i from station $i-1$. Stations may have multiple (including none) items, and the player can pick up and hold an unlimited number of items. The objective is to reach the last station: station n.

You and your friend find this game too easy, so you decide to link up two games of Mokepon: your game has n_{A} stations and your friend's game has n_{B} stations. However, special items may appear in either game, and are now uniquely identified by the pair of (id, A or B): its identifier, and which game it comes from. To progress to station i in a given game, either one of you must have picked up the item with identifier i for that game.

Items get placed randomly in each of the games, and the items may be placed in such a way that it is not possible to win. Your goal is to count the number of distributions of items such that it is possible for the two of you to both reach the end station in your respective games. Since this number may be large, calculate it modulo a specified prime p.

We say two distributions of items to stations are different if there is at least one station where the set of items at that station differ.

The only line of input contains three space separated integers, n_{A}, n_{B}, and p ($2 \leq n_{\text{A}}, n_{\text{B}} \leq 3 \cdot 10^3$, $10^8 < p \leq 10^9 + 7$)—the number of stations in your and your friend's game, respectively, and the modulo to calculate with respect to. It is guaranteed that p is prime.

Print a line containing a single integer, the number of distributions of items such that both players win their respective games, modulo p.

Sample Input 1

```
2 2 1000000007
```

Output for Sample Input 1

```
8
```

Sample Input 2

```
15 20 998244353
```

Output for Sample Input 2

```
937612
```

Problem 5
Pair Linked Mokepon (continued)

Explanation for Sample Input 1

In the first sample, there are two special items: $(2, \mathrm{A})$ and $(2, \mathrm{B})$ corresponding to the two games A and B respectively, and thus $4^2 = 16$ configurations of stations where they could be located. The eight winning possibilities (in the notation (station, game)) are listed below. Figure 1 shows the first two possibilities.

(1, A); (1, A): Both players can immediately progress to their respective end stations.
(1, A); (2, A): The first player unlocks their final station, at which point the second player can progress.
(1, A); (1, B)
(1, B); (1, A)
(1, B); (1, B)
(1, B); (2, A)
(2, B); (1, A)
(2, B); (1, B)

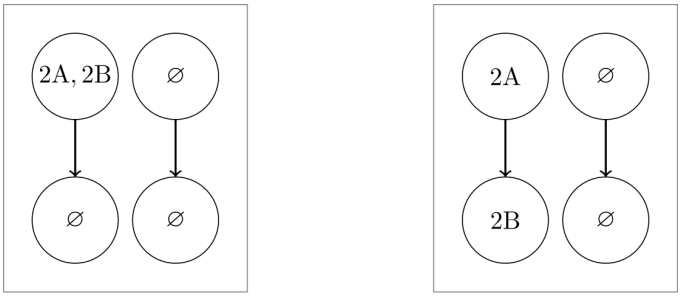

Figure 1. First two possibilities for Sample Input 1.

2025/2026 SOUTHERN CALIFORNIA REGIONAL INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 6
Fractal Painting

A fractal painting consists of an infinite number of line segments. The first segment, called A, connects points $(0, 0)$ and (x_0, y_0).

The next two segments B and C connect (x_0, y_0) to (x_1, y_1) and (x_0, y_0) to (x_2, y_2), respectively.

The rest of the painting is defined recursively. We draw two segments D and E from (x_1, y_1) so that the segments B, D, E are *similar* to the segments A, B, C. Here, *similar* segments mean that they can be matched point-to-point by performing translations, rotations, and uniform scaling on the original segments.

Similarly, we draw segments F and G from (x_2, y_2) so that the segments C, F, G are similar to the segments A, B, C. Figure 1 shows this process.

This procedure continues indefinitely.

Find out whether it is possible to find a rectangle (of any size) that contains the entire fractal painting.

The first line of input contains a single integer T ($1 \le T \le 10{,}000$), representing the number of test cases. Each of the next T lines describes a single test case. Each test case consists of a single line with six integers x_0, y_0, x_1, y_1, x_2, and y_2 in order. All coordinates are between -10^4 and 10^4, inclusive. It is guaranteed that $(0, 0)$, (x_0, y_0), (x_1, y_1), and (x_2, y_2) are all distinct points.

For every test case, print a line with the string "`YES`" if the entire fractal painting can fit in some rectangular frame. Print a line with the string "`NO`" if there is no such rectangle.

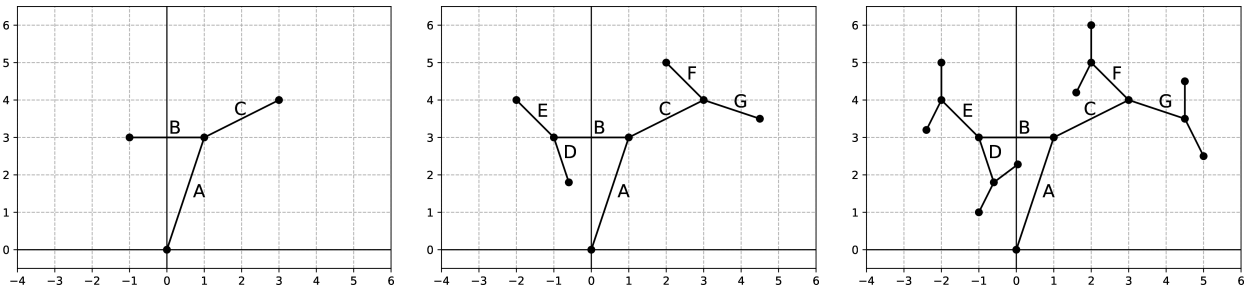

Figure 1. The first three iterations of the first fractal painting in the Sample Input.

Sample Input

```
3
1 3 -1 3 3 4
1 1 67 0 0 67
67 67 1 0 0 1
```

Output for the Sample Input

```
YES
NO
YES
```

2025/2026 SOUTHERN CALIFORNIA REGIONAL INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 7
Child's Play

Your younger sibling decided to engage in their favorite way to pass time: Cube Stacking. They have N cubes with various side lengths s_i (measured in inches) and they stacked each cube on top of one another, gluing each cube to one another to make it stick, in a particular order. When one cube is stacked on another, the surface of the smaller cube is completely glued to the surface of the larger. The surfaces of two cubes of equal size are completely glued to each other.

After building a tall tower, your sibling decided they wanted it to be painted a different color, so they asked your parents to paint the entire structure green. Your parents turned to you and asked you to figure out the minimum amount of paint that they would need to buy to paint the whole thing.

Given the order in which the cubes were stacked, determine the total area that your parents will need to paint in order to repaint the whole structure.

For example in the first sample case, there is only a single cube that needs to be fully painted, needing paint to cover 6 square inches of surface area. The bottom of the cube also needs to be painted. See Figure 1 for a depiction of the second sample case.

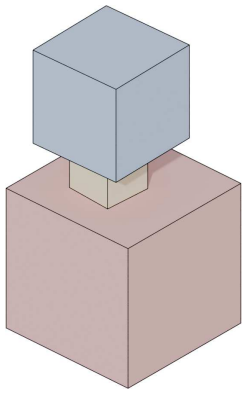

Figure 1. Cube stack tower corresponding to Sample Input 2.

The first line of input contains one integer N ($1 \leq N \leq 10^4$) representing the number of cubes that your sibling stacks. The next line of input contains N space separated integers s_i ($1 \leq s_i \leq 100$) representing the side length of the ith cube in inches. You know that for all $i \geq 1$, cube i was stacked on top of cube $i-1$ where cube 0 is the bottom-most cube.

Print a line with a single number P, representing the number of square inches that your parents will need to paint in order to paint the whole tower structure green.

Sample Input 1

```
1
1
```

Output for Sample Input 1

```
6
```

Problem 7
Child's Play (continued)

Sample Input 2

```
3
3 1 2
```

Output for Sample Input 2

```
80
```

2025/2026 SOUTHERN CALIFORNIA REGIONAL INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 8
Pair Sums

You are trying to solve a variation of the Two-Sum problem where you are asked to find how many pairs of values exist in an array of non-negative integers A that sum to some target value T. More formally, you want to count the number of distinct pairs of indices (i, j) that exist in A such that $A_i + A_j = T$ and $i \neq j$. (i, j) and (j, i) represent the same pair.

You were supposed to be given all N values in the array, but unfortunately there was corruption in the data. You received M values instead, meaning many values could be missing.

You want to determine the maximum number of pairs of values S that sum to T that could have existed in the original array.

For example, in the first sample input, the original array could have been $A = [0, 1, 2, 2, 3, 3, 3]$. This array has 6 pairs of values that sum to 5, which is maximal. One of these, identified by the 0-indexed position in the array, is $(2, 4)$ because $A_2 + A_4 = 2 + 3 = 5$.

The first line of input contains three space separated integers M ($1 \leq M \leq 10^5$), N ($M \leq N \leq 10^9$), and T ($0 \leq T \leq 10^9$) representing the number of elements of the array that you receive, the number of elements in the original array, and the target value to sum to respectively.

The second line of input contains M space separated integers $A_1\ A_2\ \ldots\ A_M$, where ($0 \leq A_i \leq 10^9$) are the values in the portion of the array that you received. The array values are not necessarily distinct.

Print a line with a single number S, representing the maximum number of value pairs that sum to T that could have existed in the original array.

Sample Input 1

```
4 7 5
0 1 2 3
```

Output for Sample Input 1

```
6
```

Sample Input 2

```
3 4 5
8 1 10
```

Output for Sample Input 2

```
1
```

2025/2026 SOUTHERN CALIFORNIA REGIONAL INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 9
Penniless in America

In May of 2025, the U. S. Treasury placed its last order for penny blanks (metal discs that have not yet been stamped into pennies). Production of pennies is expected to cease in early 2026. Item prices in stores can continue to be posted to the cent, and electronic transactions will charge to the cent.

However, as pennies disappear from circulation, cash transactions will need to be rounded to the nearest nickel (five cents). A common rounding rule is as follows: If the final digit of a purchase ends in 3, 4, 8, or 9 cents, the total will be rounded up; if it ends in 1, 2, 6, or 7 cents, it will be rounded down. Transactions ending in 0 or 5 cents are not rounded. Your program must read a series of dollar and cent values, then print the amount rounded to the nearest five cents.

Input is a series of transaction amounts, one amount per line, expressed as dollars and cents (dollars, a decimal point, and exactly two digits after the decimal point), ending with end-of-file. There are at least 1 and at most 100 transaction amounts. Each transaction amount is between 0.03 and 10000.00 inclusive.

For each transaction amount, print a line with the transaction amount rounded to the nearest five cents. Express the value as dollars and cents (dollars, a decimal point, and two digits after the decimal point for cents, without a dollar sign or other punctuation). For amounts under one dollar, print a leading zero before the decimal point.

Sample Input

```
0.03
1.24
1.22
99.99
100.00
```

Output for the Sample Input

```
0.05
1.25
1.20
100.00
100.00
```

2025/2026 SOUTHERN CALIFORNIA REGIONAL INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 10
San Francisco Distances

You are visiting your friend in San Francisco, and you are trying to find your way around town. Unlike some cities, where streets are laid out in a uniform grid, downtown San Francisco is divided into two half grids, at a 45° angle to each other. You would like to find the shortest distance between points in the city.

For the purpose of this problem, we model San Francisco as an infinite 2-D plane, as shown in Figure 1. The horizontal axis is a street, Market Street. Streets on the lower half of the plane (South of Market Street) are axis-aligned, and have a street corner at the origin. Streets on the upper half of the plane (North of Market Street) are at a 45° angle to the coordinate axes, and also have a street corner at the origin. Both grids are uniformly spaced with the same spacing as each other. You can only walk along streets, and you can only change streets where they are coincident to each other.

Coordinates are specified as a tuple (X, Y, D), where X and Y are integers ($|X| \leq 10^9$, $|Y| \leq 10^9$) and D is one of the characters "`S`" or "`N`".

- If D is "`S`" the coordinate denotes a point South of Market Street. It is X streets rightward and Y streets downward from the origin, and it is additionally guaranteed that $Y \geq 0$.
- Otherwise, D is "`N`", and the coordinate denotes a point North of Market Street. It is X streets diagonally-up-right from the origin and Y streets diagonally-up-left from the origin, and it is additionally guaranteed that $(X + Y) \geq 0$.

The first line of input contains one integer, Q ($1 \leq Q \leq 100$), the number of queries you must handle. The next Q lines each contain a query composed of six space-separated tokens X_1 Y_1 D_1 X_2 Y_2 D_2, indicating that you should compute the shortest distance between (X_1, Y_1, D_1) and (X_2, Y_2, D_2). It is guaranteed that these refer to two distinct points on the coordinate plane.

For each query, print a line containing a single real number, the shortest walking distance between the points in the query. Your answer will be accepted if it has an absolute difference of less than 10^{-6} from the judges' reference value.

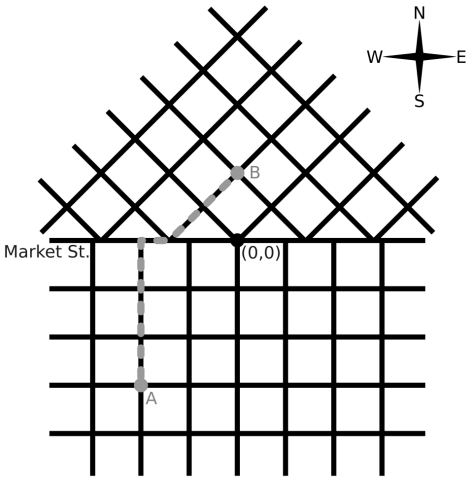

Figure 1. The San Francisco two half-grids, showing the first Sample Input query.

Problem 10
San Francisco Distances (continued)

Sample Input

```
4
-2 3 S 1 1 N
3 5 S -1 0 S
1 1 N 3 0 N
3 -1 N 0 2 N
```

Output for the Sample Input

```
5.58578643763
9
3
5.41421356237
```

2025/2026 SOUTHERN CALIFORNIA REGIONAL INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 11
Farthest City

There are n cities in the State of Confusion, numbered from 1 to n. There are n bidirectional roads that connect the cities. The ith road connects cities a_i and b_i, and is d_i miles long.

All n cities are connected. In other words, regardless of which city you start from, it is always possible to reach any other city using the given roads. In addition, no road connects a city to itself, and there is at most one road between any pair of cities.

You are interested in holding a statewide event in one of the cities. One of the important factors to consider is how far away the rest of the cities are from where the event will be held. The distance between two cities is the length of the shortest path that connects them.

Given the information about all roads, find the distance to the farthest city from every city in the state.

The first line of the input contains a single integer n ($3 \leq n \leq 3 \cdot 10^5$), representing the number of cities in the state. Each of the next n lines contains three integers a_i, b_i, d_i ($1 \leq d_i \leq 10^7$), separated by spaces, describing one of the roads.

Output n integers on a single line, separated by spaces, where the ith number denotes the distance of the city that is the farthest from city i.

Sample Input

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

Output for the Sample Input

```
4 5 3 5
```

2025/2026 SOUTHERN CALIFORNIA REGIONAL INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 12
TV Remotes

You are on an infinite number line where there is a TV placed at every integral X coordinate. You also have a series of N universal TV remotes where remote i is placed at coordinate x_i with some range r_i. A universal TV remote is special, because when remote i is pressed, it can send a power signal to any single TV in the interval $[x_i - r_i, x_i + r_i]$. This means that exactly one TV will receive the signal, but we cannot predict which one it will be!

All the TVs are initially off. You decide to press the power button on every remote exactly once. If the power input is applied to a TV that is off, it will turn on. If the power input is applied to a TV that is on, it will turn off.

Compute the minimum number of TVs that could be on after pressing the power button on every TV remote.

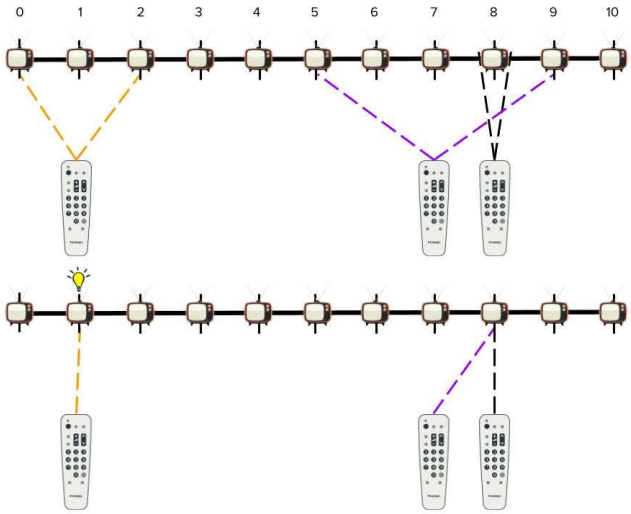

Figure 1. Diagram representing the Sample Input.

Figure 1 is an illustration of the Sample Input. The first half shows the different televisions that each remote can connect to, and the second image shows one way that the remotes could have actually paired, that would cause a minimum number of one television to remain on at the end.

The first line of input contains one integer N ($1 \leq N \leq 10^5$) representing the number of TV remotes. Each of the next N lines of input contain two space separated integers x_i r_i where $-10^9 \leq x_i \leq 10^9$ and $0 \leq r_i \leq 10^9$.

Print a line with a single number T, representing the minimum number of TVs that could be on after pressing the power button on every TV remote.

Sample Input

```
3
7 2
1 1
8 0
```

Problem 12
TV Remotes (continued)

Output for the Sample Input

```
1
```

2025/2026 SOUTHERN CALIFORNIA REGIONAL INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 13
Two-For-One Crosswords

In 1996, the New York Times made history by publishing a crossword with seemingly two solutions. Now you are solving a crossword, and you wonder: does your puzzle have multiple solutions?

A crossword is a puzzle consisting of a grid of black and white squares. The white squares are grouped into slots. A slot is a group of two or more contiguous white squares in the same row or column that is preceded and followed by a black square or the grid edge (in that same row or column). Every such group is a slot, and every white square is guaranteed to be in at least one slot. A slot with squares in the same row is a horizontal slot; otherwise, it is a vertical slot.

To solve a crossword, a player must write a single letter in each white square so that every slot contains a valid word when read from left-to-right (for horizontal slots) or from top-to-bottom (for vertical slots). In this problem, you have already determined exactly two possible valid words for each slot.

You must determine the number of ways to solve the puzzle using the possible valid words. Because it may be large, give your answer modulo $(10^9 + 7)$.

Note that slots are ordered according to their top-left-most square. First they are ordered from top-to-bottom by the row of the top-left-most square, and then from left-to-right by the column of the top-left-most-square (if the rows are the same). If a horizontal slot and vertical slot both have the same top-left-most square, the horizontal one comes first.

The first line of input contains three integers R ($1 \leq R \leq 10^3$), C ($1 \leq C \leq 10^3$) and S ($1 \leq S$), representing the number of rows, columns, and slots, respectively, in the grid.

The next R lines describe the grid. Each line contains a string of length C characters, each of which is either "`#`" to represent a black square or "`.`" to represent a white square.

The next S lines contain the possible valid words. The ith of these lines contain two distinct strings a_i and b_i composed of uppercase letters A-Z, which are the possible valid words for the ith slot, according to the ordering described above. It is guaranteed that the lengths of a_i and b_i are the same as the number of squares in slot i.

Print a line with a single number, the number of ways to solve the puzzle using the possible valid words modulo $(10^9 + 7)$.

Sample Input 1

```
3 4 3
.##.
....
.##.
WIN BUG
ACE TOP
ICPC INFO
```

Output for Sample Input 1

```
2
```

Problem 13
Two-For-One Crosswords (continued)

Sample Input 2

```
3 4 4
....
.##.
....
ACCA BCCB
ACA BCB
ACA BCB
BCCA ACCB
```

Output for Sample Input 2

```
0
```