acm International Collegiate Programming Contest

IBM event sponsor

2014 ACM ICPC Southeast USA Regional Programming Contest

15 November, 2014

Division 1

Hosted by:

Florida Institute of Technology

Georgia Institute of Technology

University of West Florida

A: Alchemy

Since the days of yore, alchemy has been studied and practiced. The practice makes alchemists able to transmute materials into other forms. Transmuting materials requires drawing a transmutation circle on the ground. A little known fact about transmutation circles is they can be drawn inside or outside other transmutation circles. By activating certain configurations in the correct order, more powerful transmutations can be produced. Activating circles incorrectly can have drastic effects on the alchemist's body.

A young alchemist named Nicholas Flamel would like to learn the ways of alchemy. He is going to draw several configurations of transmutation circles on the ground. When a circle is drawn it burns bright red representing the element of fire. The drawing of the circle itself produces no energy, but it has an effect on any and all circles that are already drawn inside! All of the circles inside the newly drawn circle quickly change to their complement elements. Fire changes to a cool blue representing water. Circles that were blue for water will burn fiery red once again. This transformation can either create or drain energy. Interestingly, it is the transformation, and not the drawing, that emanates energy. Beware, energy can go negative at any time draining the alchemist's life force.

Nicholas wants to get as much out of his transmutations as possible. To do so requires him to draw his circles in an order that releases the most energy. Determine the maximum amount of energy that can be released, and the order in which he should draw the circles.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will start with a line with a single integer ***n*** (1≤***n***≤2,000) indicating the number of circles. The next ***n*** lines will describe the circles, in order, from circle 1 to circle ***n***. Each line will describe its circle with 5 integers, with a single space between integers:

x y r a b

Where (***x***,***y***) is the center of the circle (-20,000≤***x***,***y***≤20,000), ***r*** is the radius of the circle (1≤***r***≤20,000), ***a*** is the energy released in the transition from fire to water, and ***b*** is the energy released in the transition from water to fire (-500≤***a***,***b***≤500). It is guaranteed that no two circles' edges will intersect.

Output

Output exactly two lines. On the first line output a single integer representing the maximum energy that can be produced by activating the circles. On the second line output the order of drawing the circles that can produce that energy. If more than one order will work, output the one that comes first lexicographically. Output a single space between integers. Output no extra spaces.

Sample Input

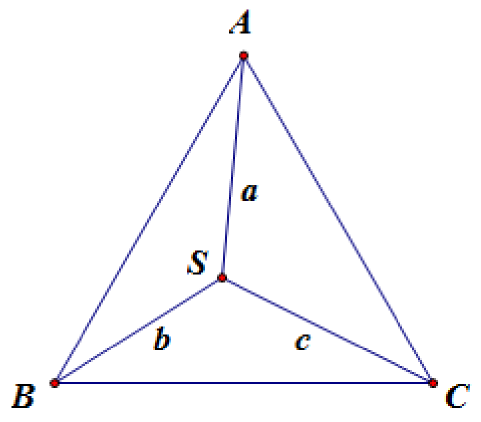
```
8
0 0 100 -100 -100
0 0 50 -10 -10
0 0 10 -100 1000
0 0 1 100 100
1000 1000 100 -1 1
1000 1000 50 -1 1
1000 1000 10 -1 1
1000 1000 1 -1 1
```

Sample Output

```
1200
4 3 1 2 5 6 7 8
```

B: Stained Carpet

The Algebraist Carpet Manufacturing (ACM) group likes to produce area carpets based upon various geometric figures. The 2014 ACM carpets are all equilateral triangles. Unfortunately, due to a manufacturing defect, some of the carpets are not as stain-resistant as intended. The ACM group is offering to replace each defective carpet that contains a stain.



The web form used to report the stained carpet requests the three distances that the stain is away from the corners of the rug. Based upon these three numbers, you need to compute the area of the rug that is to be sent to the customer, or indicate that the customer's carpet doesn't come from ACM.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will consist of a single line with three floating point numbers ***a***, ***b*** and ***c*** ($0<a,b,c\leq 100$) representing the distances from the stain to each of the three corners of the carpet. There will be a single space between ***a*** and ***b***, and between ***b*** and ***c***.

Output

Output a single line with a single floating point number. If there is a carpet that satisfies the constraints, output the area of this carpet. If not, output `-1.000`. Output this number to exactly three decimal places, rounded. Output no spaces.

Sample Input	**Sample Output**
1 1 1.732051	1.732
1 1 3.0	-1.000
1.732051 1.732051 1.732051	3.897

C: Containment

A 10x10x10 three-dimensional grid of tightly packed cubic atomic energy cells aboard the starship Fiugtuwf is reporting failures on several of its cells. The ship's engineer must set up enclosures that will contain all of the cells that are reported to be failing, in order to avoid a meltdown. It is imperative that the enclosures be finished in the shortest amount of time, even if that requires some healthy cells to be enclosed along with the defective ones. The enclosures are formed by square panels which fit perfectly between adjacent cells, or can be placed along the sides of the cells on the edges of the grid. Each panel is exactly the size and shape of a face of one of the cubic cells. For full containment, each enclosure must be completely closed. Given the coordinates of each defective cell, report the minimum number of panels required to contain the problem.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will start with a line with a single integer ***n*** (0≤***n***≤1,000) indicating the number of defective cells. Each of the next ***n*** lines will hold an (***x***,***y***,***z***) coordinate (0≤***x***,***y***,***z***≤9) indicating the location in the grid of a defective cell. All of the coordinates in a test case will be unique.

Output

Output a single line with a single integer, indicating the minimum number of panels required to contain the defective cells. Output no spaces.

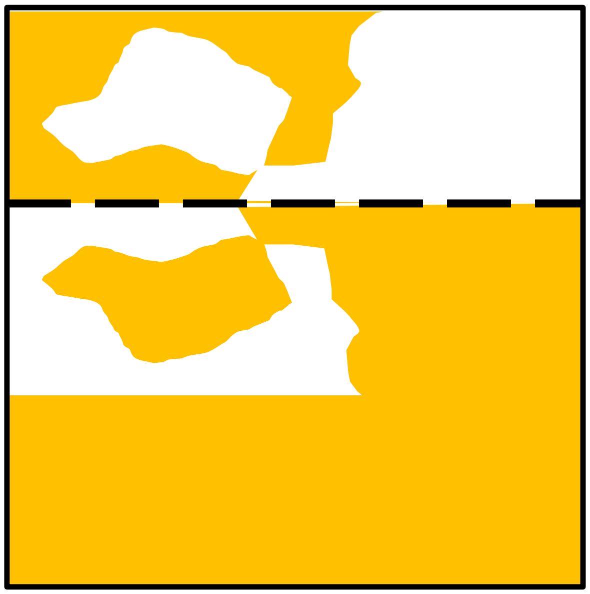
Sample Input	Sample Output
1 0 0 0	6

2
0 0 0
0 0 1
	10

3
0 0 0
0 0 1
0 1 1
	14

D: Gold Leaf

Gold Leaf is a very thin layer of gold, with a paper backing. If the paper gets folded and then unfolded, the gold leaf will stick to itself more readily than it will stick to the paper, so there will be patches of gold and patches of exposed paper. Note that the gold leaf will always stick to itself, rather than the paper. In the following example, the paper was folded along the dashed line. Notice how the gold leaf always sticks to one side or the other, never both.



Consider a crude digital image of a sheet of gold leaf. If the area covered by a pixel is mostly gold, that will be represented by a '**#**'. If it's mostly exposed paper, it will be represented by a '**.**'. Determine where the sheet was folded. The sheet was folded exactly once, along a horizontal, vertical, or 45 degree or 135 degree diagonal line. If the fold is horizontal or vertical, it is always between rows/columns. If the fold is diagonal, then the fold goes through a diagonal line of cells, and the cells along the fold are always '**#**'.

The Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line with two integers, ***n*** and ***m*** (2≤***n***,***m***≤25), where ***n*** is the number of rows, and ***m*** is the number of columns of the image. Each of the next ***n*** lines will contain exactly ***m*** characters, all of which will be either '**#**' or '**.**'. This represents a crudely collected digital image of the sheet of gold leaf. There is guaranteed to be at least one '**.**', and there is guaranteed to be a solution.

The Output

Output a single line with four integers, with a single space between integers, indicating the places where the fold hits the edges of the paper. Output no extra spaces. Output them in this order:

r1 c1 r2 c2

where (***r1***,***c1***) and (***r2***,***c2***) are row/column coordinates (***r***=row, ***c***=column). The top left character of the image is (1,1) and the bottom right is (***n***,***m***).

If the fold is horizontal or diagonal, list the left coordinates before the right. If the fold is vertical, list the top coordinates before the bottom.

If the fold is horizontal, use the coordinates above the fold. If the fold is vertical, use the coordinates to the left of the fold. If the fold is diagonal, use the coordinates of the edge pixels that the fold goes through.

If more than one fold is possible, choose the one with the smallest first coordinate, then the smallest second coordinate, then third, then fourth.

Sample Input / **Sample Output**

```
8 10
#.#..##..#
####..####
###.##....
...#..####
....##....
.#.##..##.
##########
##########
```

```
3 1 3 10
```

```
5 20
###########.#.#.#.#.
###########...#.###.
##########..##.#..##
###########..#.#.##.
###########.###...#.
```

```
1 15 5 15
```

```
5 5
.####
###.#
##..#
#..##
#####
```

```
4 1 1 4
```

E: Hill Number

A *Hill Number* is a positive integer, the digits of which possibly rise and then possibly fall, but never fall and then rise. For example:

> **12321** is a hill number.
>
> **12223** is a hill number.
>
> **33322111** is a hill number.
>
> **1232321** is **not** a hill number.

Given a positive integer, if it is a hill number, print the number of positive hill numbers less than or equal to it. If it is not a hill number, print -1.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will consist of a single integer ***n*** ($1 \le n \le 10^{18}$).

Output

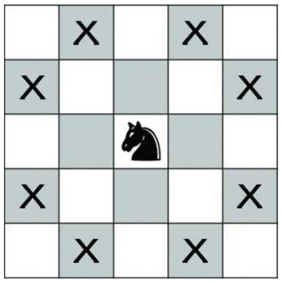
Output a single line with a single integer. If the input is a hill number, then output the number of hill numbers less than or equal to it. If the input is not a hill number, then output -1. Output no spaces.

Sample Input	**Sample Output**
10	10
55	55
101	-1
1000	715
1234321	94708

F: Knights

Magnus is the youngest chess grandmaster ever. He loves chess so much that he decided to decorate his home with chess pieces. To decorate his long corridor, he decided to use the knight pieces. His corridor is covered by beautiful square marble tiles of alternating colors, just like a chess board, with ***n*** rows and ***m*** columns. He will put images of knights on some (possibly none) of these tiles. Each tile will contain at most one knight.

The special thing about his arrangement is that there won't be any pair of knights can attack each other. Two knights can attack each other if they are placed in two opposite corner cells of a 2 by 3 rectangle. In this diagram, the knight can attack any of the Xs.



Given the dimension of the long corridor, your task is to calculate how many ways Magnus can arrange his knights. Two arrangements are considered different if there exists a tile which contains a knight in one arrangement but not in the other arrangement (in other words, rotations and reflections are considered different arrangements).

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will consist of a single line with two integers ***n*** and ***m*** ($1 \le n \le 4$, $1 \le m \le 10^9$) representing the dimensions of the carpet. There will be a single space between ***n*** and ***m***.

Output

Output a single line with a single integer representing the number of possible arrangements, modulo (10^9+9). Output no spaces.

Sample Input	Sample Output
`1 2`	`4`
`2 2`	`16`
`3 2`	`36`

G: Word Ladder

A *Word Ladder* is a puzzle in which you transform one word into another, by changing one letter at a time. But, there's a catch: every word that you form in each step must be in the dictionary! Here's an example of how to transform **CAT** into **GAS**:

CAT→CAR→WAR→WAS→GAS

Of course, you want to use the fewest number of transitions possible. These puzzles can be tough, and often you'll think to yourself: "Darn it! If only [*some word*] was in the dictionary!"

Well, now is your chance! Given a dictionary, and a starting and ending word, what ONE single word could you add to the dictionary to minimize the number of steps to get from the starting word to the ending word, changing only one letter at a time, and making sure that every word at every step is in the dictionary?

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will start with a line with a single integer ***n*** (2≤***n***≤1,000) which indicates the number of words in the dictionary. The dictionary will follow on the next ***n*** lines, with one word per line. All words will consist of between 1 and 8 capital letters only, and all of the words in a test case will be of the same length. The first word in the list will be the starting word of the word ladder, and the second will be the ending word of the word ladder.

Output

Output exactly two lines. The first line holds the one single word that you would add to the dictionary, and the second holds an integer indicating the minimum number of steps to get from the starting word to the ending word, adding your word. Output no spaces.

It is possible that there's more than one word you can add that will make your path as short as possible. In this case, output the solution word that comes first alphabetically.

It is possible that there's no word you can add that will that will make your path any shorter. In this case, output **0** (zero) as the word.

It is possible that there's no word you can add that makes the solution possible. In this case, output **0** (zero) as the word, and **-1** as the number of steps.

Sample Input	Sample Output
3 CAT DOG COT	COG 3
2 CAT DOG	0 -1
4 CAT DOG COT COG	0 3

H: Shuffles

The most common technique for shuffling a deck of cards is called the Riffle or Dovetail shuffle. The deck is split into two stacks, which are then interleaved with each other. The deck can be split anywhere, and the two stacks can be interleaved in any way.

For example, consider a deck with 10 unique cards:

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

Split them somewhere:

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

And interleave them in some way:

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

Do it again. Split them somewhere:

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

And interleave them in some way:

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

This is one possible ordering after 2 shuffles. Suppose there are ***n*** unique cards, and that they start out perfectly ordered: 1, 2, 3, ..., ***n***. Given an ordering of the deck, what is the smallest number of shuffles that could possibly put the deck in that order?

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a single integer ***n*** (1≤***n***≤1,000,000) indicating the number of cards in the deck. On the next line will be ***n*** unique integers ***c*** (1≤***c***≤***n***), with a single space between them, indicating an ordering of the ***n*** cards. The values ***c*** are guaranteed to be a permutation of the numbers 1..***n***.

Output

Output a single line with a single integer indicating the minimum number of shuffles that could possibly put the deck in the given order. Output no spaces.

Sample Input	Sample Output
10 1 2 7 3 8 9 4 5 10 6	1
10 3 8 1 9 4 5 2 7 10 6	2
8 2 1 4 3 6 5 8 7	3

I: Stamp Stamp

Bureaucrats love bureaucracy. This assertion seems fairly obvious but a less obvious observation is the amount of paperwork this means!

When paperwork is complete, a bureaucrat stamps the official document with their official stamp of office. Some bureaucrats are extra thorough and stamp the document multiple times. We are interested primarily in bureaucrats that stamp their documents exactly twice.

A bureaucrat stamp takes up some rectangular area. For example, the below is a bureaucrat stamp:

```
..#..#..
.######.
..#..#..
```

When the bureaucrat stamps the paper twice it is potentially moved to a different location on the paper, but it is not rotated. The stamp will always be axis aligned. The '#' symbol on a stamp covers the paper with black ink at the cell on the paper that is pressed. A '.' doesn't leave any marks on the paper nor does it remove a mark. If a cell of the paper is marked twice it is not discernable from a cell that is marked once.

You will be given a mark on a paper that was stamped twice by a stamp. Your task is to determine the minimum number of nubs ('#' symbols) that could have possibly been on the original stamp. The paper is guaranteed to be stamped twice by the entire stamp. (All of the stamp will be on the paper in both stampings)

Consider the following mark on paper:

```
..#..#..
.######.
.######.
..#..#..
```

It could have been made with the first stamp example, with the second stamping exactly one cell below the first. Or, it could have been made with a stamp that looks like this, stamped side-by-side:

```
..#..
.###.
.###.
..#..
```

In this case, 8 would be the correct answer.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line with two integers, ***n*** and ***m*** (1≤***n***,***m***≤500) indicating the height and width of the mark on the paper. Each of the next ***n*** lines will have exactly ***m*** characters, consisting only of '**#**' and '**.**', representing the mark. Every test case will have at least one '**#**'.

Output

Output a single line with a single integer, indicating the minimum number of nubs of a bureaucrat's stamp that could have possibly made the input mark by stamping exactly twice. Output no spaces.

Sample Input	**Sample Output**

`4 8`
`..#..#..`
`.######.`
`.######.`
`..#..#..`
	`8`

`3 3`
`...`
`.#.`
`...`
	`1`

`2 6`
`.#####`
`#####.`
	`5`

`2 5`
`.#.#.`
`#.#.#`
	`3`