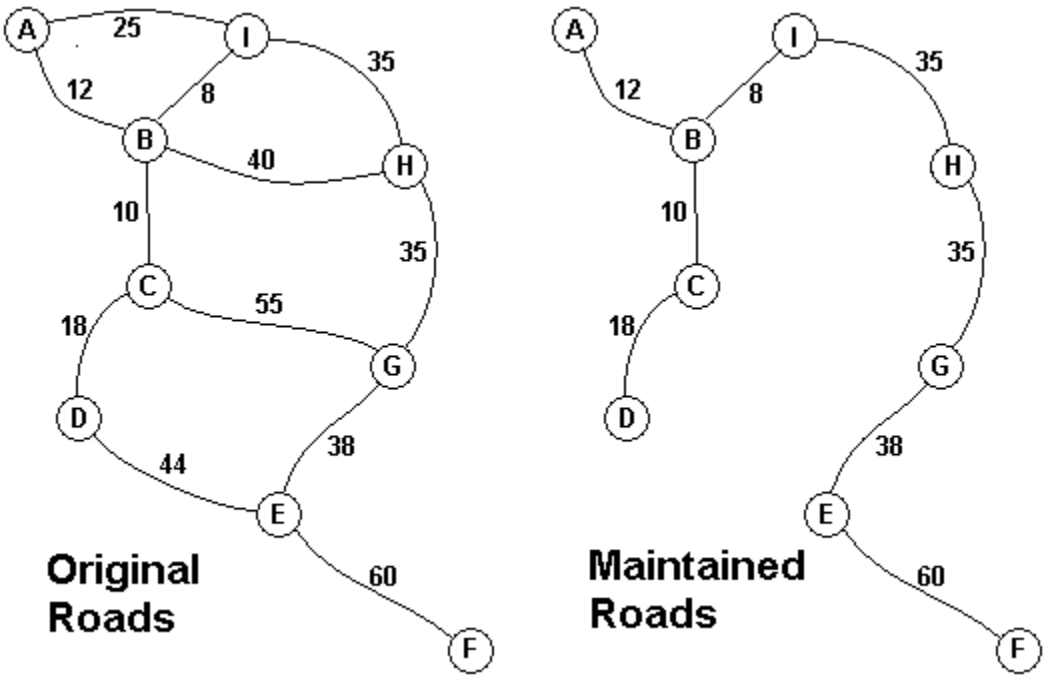
Problem G: Jungle Roads

Source file: `jungle.{c, cpp, java, pas}`
Input file: `jungle.in`
Output file: `jungle.out`



The Head Elder of the tropical island of Lagrishan has a problem. A burst of foreign aid money was spent on extra roads between villages some years ago. But the jungle overtakes roads relentlessly, so the large road network is too expensive to maintain. The Council of Elders must choose to stop maintaining some roads. The map above on the left shows all the roads in use now and the cost in aacms per month to maintain them. Of course there needs to be some way to get between all the villages on maintained roads, even if the route is not as short as before. The Chief Elder would like to tell the Council of Elders what would be the smallest amount they could spend in aacms per month to maintain roads that would connect all the villages. The villages are labeled A through I in the maps above. The map on the right shows the roads that could be maintained most cheaply, for 216 aacms per month. Your task is to write a program that will solve such problems.

The input consists of one to 100 data sets, followed by a final line containing only 0. Each data set starts with a line containing only a number n, which is the number of villages, $1 < n < 27$, and the villages are labeled with the first n letters of the alphabet, capitalized. Each data set is completed with n-1 lines that start with village labels in alphabetical order. There is no line for the last village. Each line for a village starts with the village label followed by a number, k, of roads from this village to villages with labels *later* in the alphabet. If k is greater than 0, the line continues with data for each of the k roads. The data for each road is the village label for the other end of the road followed by the monthly maintenance cost in aacms for the road. Maintenance costs will be positive integers less than 100. All data fields in the row are separated by single blanks. The road network will always allow travel between all the villages. The network will never have more than 75 roads. No village will have more than 15 roads going to other villages (before or after in the alphabet). In the sample input below, the first data set goes with the map above.

The output is one integer per line for each data set: the minimum cost in aacms per month to maintain a road

system that connect all the villages. **Caution**: A brute force solution that examines every possible set of roads will not finish within the one minute time limit.

Example input:

```
9
A 2 B 12 I 25
B 3 C 10 H 40 I 8
C 2 D 18 G 55
D 1 E 44
E 2 F 60 G 38
F 0
G 1 H 35
H 1 I 35
3
A 2 B 10 C 40
B 1 C 20
0
```

Example output:

```
216
30
```

Last modified on October 26, 2002 at 7:20 PM.

Problem H — limit 5 seconds

Speed Racer

Description

Speed Racer must go go go rescue Trixie at the top of Mount Domo! He must get there as quickly as possible, but his Mach 5 only holds a specific amount of fuel, and there is no way to refuel on the way. Luckily, he knows precisely how much fuel is consumed at a particular speed, taking into account air resistance, tire friction, and engine performance. For a given speed v in kilometers per hour, the amount of fuel consumed in liters per hour is

$$av^4 + bv^3 + cv^2 + dv$$

Assuming Speed travels at a constant speed, his tank holds t liters of fuel, and the top of Mount Domo is m kilometers away, how fast must he drive?

Input

The input will be one problem per line. Each line will contain six nonnegative floating point values representing a, b, c, d, m, and t, respectively. No input value will exceed 1000. The values of c, d, m, and t will be positive. There will always be a solution.

The output should be formatted as a decimal number with exactly two digits after the decimal point, and no leading zeros. The output value should be such that the Speed Racer will not run out of fuel (so truncate, rather than round, the final result). You are guaranteed that no final result will be within 10^{-6} of an integer multiple of 0.01.

Output

The required output will be a single floating point value representing the maximum speed in kilometers per hour that Speed Racer can travel to reach the top of Mount Domo without running out of gas.

Sample Input	Sample Output
0.000001 0.0001 0.029 0.2 12 100 2.8e-8 7.6e-6 0.0013 0.47 11.65 20.81 1.559e-7 1.8195e-5 0.0022233 0.31292 58.902 85.585	134.41 257.45 142.65

Problem A

Robots on a grid

You have recently made a grid traversing robot that can find its way from the top left corner of a grid to the bottom right corner. However, you had forgotten all your AI programming skills, so you only programmed your robot to go rightwards and downwards (that's after all where the goal is). You have placed your robot on a grid with some obstacles, and you sit and observe. However, after a while you get tired of observing it getting stuck, and ask yourself "How many paths are there from the start position to the goal position?", and "If there are none, could the robot have made it to the goal if it could walk upwards and leftwards?"

So you decide to write a program that, given a grid of size $n \times n$ with some obstacles marked on it where the robot cannot walk, counts the different ways the robot could go from the top left corner s to the bottom right t, and if none, tests if it were possible if it could walk up and left as well. However, your program does not handle very large numbers, so the answer should be given modulo $2^{31} - 1$.

Input specifications

On the first line is one integer, $1 \leq n \leq 1000$. Then follows n lines, each with n characters, where each character is one of `'.'` and `'#'`, where `'.'` is to be interpreted as a walkable tile and `'#'` as a non-walkable tile. There will never be a wall at s, and there will never be a wall at t.

Output specifications

Output one line with the number of different paths starting in s and ending in t (modulo $2^{31} - 1$) or `THE GAME IS A LIE` if you cannot go from s to t going only rightwards and downwards but you can if you are allowed to go left and up as well, or `INCONCEIVABLE` if there simply is no path from s to t.

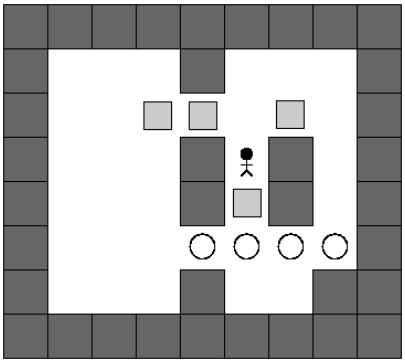
Sample input 1	**Sample output 1**
5 ..... #..#. #..#. ...#.	6

Sample input 2	**Sample output 2**
7 ......# ####... .#..... .#...#. .#..... .#..### .#.....	THE GAME IS A LIE

Problem G: Sokoban

Source file: `sokoban.{c, cpp, java}`
Input file: `sokoban.in`

Soko-ban is a Japanese word for a warehouse worker, and the name of a classic computer game created in the 1980s. It is a one-player game with the following premise. A single worker is in an enclosed warehouse with one or more boxes. The goal is to move those boxes to a set of target locations, with the number of target locations equalling the number of boxes. The player indicates a direction of motion for the worker using the arrow keys (up, down, left, right), according to the following rules.



- If the indicated direction of motion for the worker leads to an empty location (i.e., one that does not have a box or wall), the worker advances by one step in that direction.
- If the indicated direction of motion would cause the worker to move into a box, and the location on the other side of the box is empty, then both the worker and the box move one spot in that direction (i.e., the worker pushes the box).
- If the indicated direction of motion for a move would cause the worker to move into a wall, or to move into a box that has another box or a wall on its opposite side, then no motion takes place for that keystroke.

The goal is to simultaneously have all boxes on the target locations. In that case, the player is successful (and as a formality, all further keystrokes will be ignored).

The game has been studied by computer scientists (in fact, one graduate student wrote his entire Ph.D. dissertation about the analysis of sokoban). Unfortunately, it turns out that finding a solution is very difficult in general, as it is both NP-hard and PSPACE-complete. Therefore, your goal will be a simpler task: simulating the progress of a game based upon a player's sequence of keystrokes. For the sake of input and output, we describe the state of a game using the following symbols:

Symbol	Meaning
.	empty space
#	wall
+	empty target location
b	box
B	box on a target location
w	worker
W	worker on a target location

For example, the initial configuration diagrammed at the beginning of this problem appears as the first input case below.

Input: Each game begins with a line containing integers R and C, where $4 \leq R \leq 15$ represents the number of

rows, and $4 \leq C \leq 15$ represents the number of columns. Next will be R lines representing the R rows from top to bottom, with each line having precisely C characters, from left-to-right. Finally, there is a line containing at most 50 characters describing the player's sequence of keystrokes, using the symbols `U`, `D`, `L`, and `R` respectively for up, down, left, and right. You must read that entire sequence from the input, even though a particular game might end successfully prior to the end of the sequence. The data set ends with the line `0 0`.

We will guarantee that each game has precisely one worker, an equal number of boxes and locations, at least one initially misplaced box, and an outermost boundary consisting entirely of walls.

Output: For each game, you should first output a line identifying the game number, beginning at 1, and either the word `complete` or `incomplete`, designating whether or not the player successfully completed that game. Following that should be a representation of the final board configuration.

Example input:	**Example output:**
8 9 ######### #...#...# #..bb.b.# #...#w#.# #...#b#.# #...++++# #...#..## ######### ULRURDDDUULLDDD 6 7 ####### #..#### #.+.+.# #.bb#w# ##....# ####### DLLUDLULUURDRDDLUDRR 0 0	Game 1: incomplete ######### #...#...# #..bb...# #...#.#.# #...#.#.# #...+W+B# #...#b.## ######### Game 2: complete ####### #..#### #.B.B.# #.w.#.# ##....# #######

Problem B: Robots

Source file: `robots.{c, cpp, java, pas}`
Input file: `robots.in`
Output file: `robots.out`

Your company provides robots that can be used to pick up litter from fields after sporting events and concerts. Before robots are assigned to a job, an aerial photograph of the field is marked with a grid. Each location in the grid that contains garbage is marked. All robots begin in the Northwest corner and end their movement in the Southeast corner. A robot can only move in two directions, either to the East or South. Upon entering a cell that contains garbage, the robot will pick it up before proceeding. Once a robot reaches its destination at the Southeast corner it cannot be repositioned or reused. Since your expenses are directly proportional to the number of robots used for a particular job, you are interested in finding the minimum number of robots that can clean a given field. For example, consider the field map shown in Figure 1 with rows and columns numbered as shown and garbage locations marked with a '`G`'. In this scheme, all robots will begin in location 1,1 and end in location 6, 7.

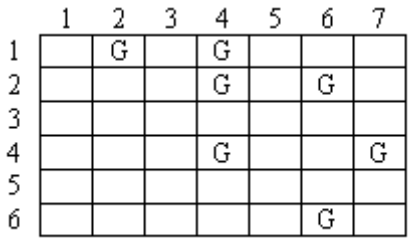


Figure 1 - A Field Map

Figure 2 below shows two possible solutions, the second of which is preferable since it uses two robots rather than three.

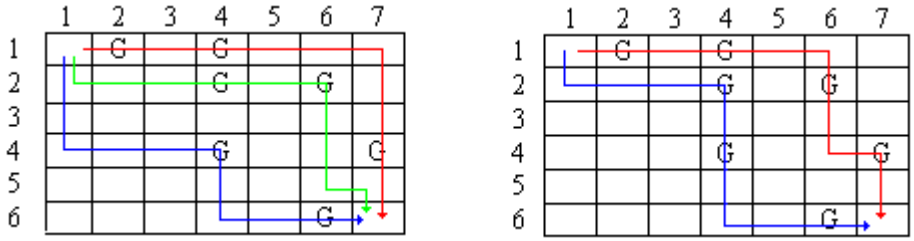


Figure 2 - Two Possible Solutions

Your task is to create a program that will determine the minimum number of robots needed to pick up all the garbage from a field.

An input file consists of one or more field maps followed by a line containing -1 -1 to signal the end of the input data. A field map consists of one or more lines, each containing one garbage location, followed by a line containing 0 0 to signal the end of the map. Each garbage location consists of two integers, the row and column, separated by a single space. The rows and columns are numbered as shown in Figure 1. The garbage locations will be given in row-major order. No single field map will have more than 24 rows and 24 columns. The sample input below shows an input file with two field maps. The first is the field map from Figure 1.

The output will consist of a single line for each field map containing the minimum number of robots needed to clean the corresponding field.

Example input:	Example output:
1 2 1 4 2 4 2 6 4 4 4 7 6 6 0 0 1 1 2 2 4 4 0 0 -1 -1	2 1

Last modified on October 24, 2003 at 8:49 AM.

Problem A: Fire Station

A city is served by a number of fire stations. Some residents have complained that the distance from their houses to the nearest station is too far, so a new station is to be built. You are to choose the location of the fire station so as to reduce the distance to the nearest station from the houses of the disgruntled residents.

The city has up to 500 intersections, connected by road segments of various lengths. No more than 20 road segments intersect at a given intersection. The location of houses and firestations alike are considered to be at intersections (the travel distance from the intersection to the actual building can be discounted). Furthermore, we assume that there is at least one house associated with every intersection. There may be more than one firestation per intersection.

The Input

The first line of input contains one integer, the number of test cases. There follow at most 10 test cases. The first line of each test case contains two positive integers: *f*,the number of existing fire stations (*f* <= 100) and *i*, the number of intersections (*i* <= 500). The intersections are numbered from 1 to *i* consecutively. *f* lines follow; each contains the intersection number at which an existing fire station is found. A number of lines follow, each containing three positive integers: the number of an intersection, the number of a different intersection, and the length of the road segment connecting the intersections. All road segments are two-way (at least as far as fire engines are concerned), and there will exist a route between any pair of intersections. After all road segments is a line containing three zeroes, signaling the end of the test case. This line is not a road.

The Output

For each test case, you are to output a single integer on a line by itself: the lowest intersection number at which a new fire station should be built so as to minimize the maximum distance from any intersection to the nearest fire station.

Sample Input

```
1
1 6
2
1 2 10
2 3 10
3 4 10
4 5 10
5 6 10
6 1 10
0 0 0
```

Output for Sample Input

```
5
```

Bushy Bushes

A graph is called a *bushy bush* if it can be built in the following way:

- At the center of the bushy bush there a cycle of some length $B \geq 3$. B is the bigness of the bushy bush. One of the vertices of this cycle is labeled as the root.
- At each vertex, other than the root, of this cycle, can be attached another cycle of length any integer at least 3 and less than B. Each vertex can have 0 or 1 such cycle attached in this way.
- Each of these small cycles can have up to one even smaller cycle attached to it at each of the vertices other than the one at which it's attached to the big cycle.
- This process may be continued for every such smaller cycle, and so on.

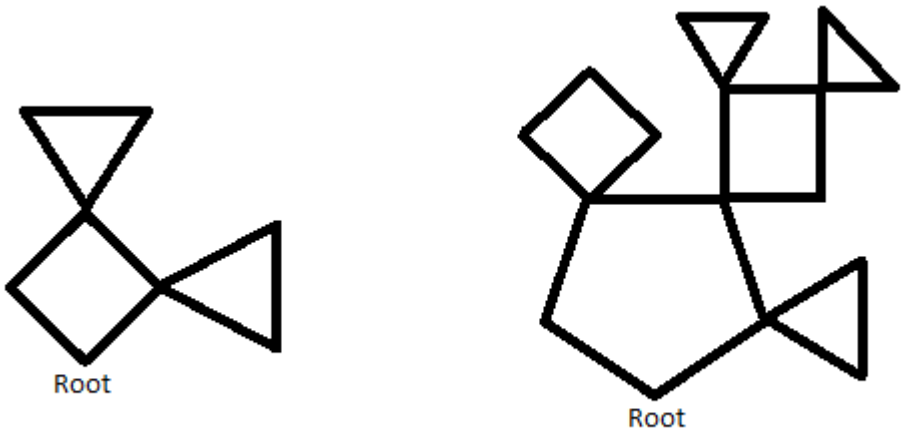


Figure 1: Examples of Bushy Bushes with Bigness 4 and 5

Given an integer B, you need to compute the number of bushy bushes of bigness B. Since this number may be very large, you actually only need to output this number modulo (%) 1000003.

Input: The first line will contain a number n, $1 \leq n \leq 10000$, the number of cases. The next n lines will each contain a number $B_k, 3 \leq B_k \leq 100000$, the bigness for which you need to find the number of bushy bushes.

Output: n lines of output, one for each case. The kth line of your output should contain a single number: the number of bushy bushes of bigness B_k, modulo 1000003.

Sample Input:
2
4
5

Output for Sample Input:
8
10000