Problem A. League of Legends

Source file name:	legends.c, legends.cpp, legends.java
Input:	`standard`
Output:	`standard`

Maybe you've heard of the Multiplayer Online Battle Arena (MOBA) game called "League of Legends." It's played with two teams of 5 players on each side. Each player controls a **champion**, with unique abilities specially designed to kill the other side's **champions**. Upon death, a **champion** waits a short while and respawns at the base.

After the game ends, it's customary to tell your teammates exactly how bad they are at playing: typically you ask them to uninstall the game. The performance of a player is measured using the **kill-death ratio (KDA)**, which is the number of kills divided by the number of deaths (if a player has no deaths, their **KDA** is ∞).

Surely the player with the lowest **KDA** on the losing team didn't perform well, so let's tell him to uninstall. We'll also tell everyone on his team with the same **KDA** as him to uninstall. Furthermore, anyone else on the losing team who had a **KDA** less than $\frac{1}{2}$ played badly, and should uninstall. The winning team is not exempt from having bad players: any player on the winning team with a **KDA** less than $\frac{1}{3}$ should also be told to uninstall.

Input

The input begins with an integer $T \leq 25$, the number of test cases. Each test case is 10 lines long. The first five lines are the 5 players on the winning team: each line begins with a string denoting the player's name (without spaces in the name), followed by two non-negative integers $K \leq 10^9$ and $D \leq 10^9$, representing that player's **kills** and **deaths**, respectively. Following this are 5 lines of similar format, but for the losing team.

Output

For each test case output one line for each person that should uninstall:
`<NAME> plz uninstall`

Print players in the same order they were given in the input. Do not print blank lines.

Example

Input	Output
1 JarvanIV 5 1 Udyr 1 1 Ahri 1 4 Vayne 4 1 Sona 1 4 Heimerdinger 0 5 Ashe 1 1 Kassadin 8 2 Darius 1 2 Elise 1 2	Ahri plz uninstall Sona plz uninstall Heimerdinger plz uninstall

Problem B. Cube on an Infinite Grid

Source file name:	cubegrid.c, cubegrid.cpp, cubegrid.java
Input:	`standard`
Output:	`standard`

A cube (which has its top face marked differently from the other faces) sits with the center of its bottom face at the origin on a grid of infinite size. The cube is moved by rolling it from lattice point to lattice point. The cube may only move orthogonally along the lattice points; i.e. each move changes either the x-coordinate or the y-coordinate of the cube's location by exactly 1.

You are given a destination set of coordinates (X, Y). You will roll the cube around on the grid until it reaches the destination square with the marked face on top. What is the minimum number of moves required to accomplish this?

Input

The input begins with an integer $T \leq 10000$, the number of test cases. Following this are T lines with two integers on each line: X and Y , representing the coordinates of the goal point. Neither X nor Y exceed 1000000000 in absolute value.

Output

Output one integer per test case: the minimum number of moves to reach the goal square with the marked face of the cube on top.

Example

Input	Output

2
0 3
4 4
	5
	8

One solution to the first test case is as follows: roll the cube right to (1,0), move it up three times to (1,3), and then one left roll to (0,3).

Problem C. Kenken You Do It?

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

KenKen is a popular logic puzzle developed in Japan in 2004. It consists of an $n \times n$ grid divided up into various non-overlapping sections, where each section is labeled with an integer target value and an arithmetic operator. The object is to fill in the entire grid with the numbers in the range 1 to n such that

- no number appears more than once in any row or column.
- in each section you must be able to reach the section's target using the numbers in the section and the section's arithmetic operator.

For this problem we are only interested in single sections of a KenKen puzzle, not the entire puzzle. Two examples of sections from an 8×8 KenKen puzzle are shown below along with some of their possible assignments of digits.

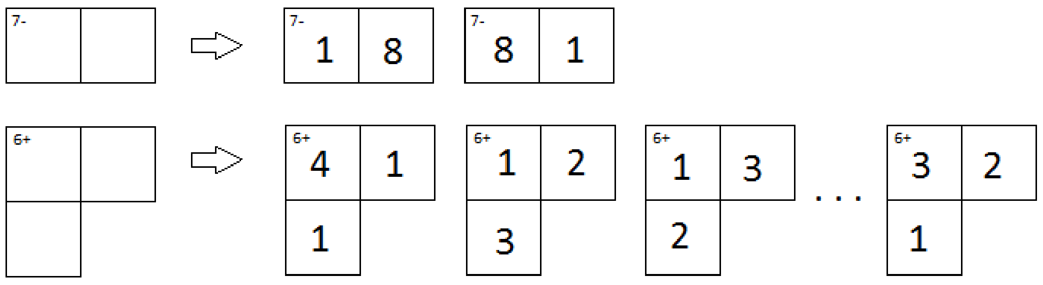


Figure C.1

Note that while sections labeled with a subtraction or division operator can consist of only two grid squares, those labeled with addition or multiplication can have any number. Also note that in a 9×9 puzzle the first example would have two more solutions, each involving the numbers 9 and 2. Finally note that in the first solution of the second section you could not swap the 1 and 4 in the first row, since that would result in two 1's in the same column.
You may be wondering: for a given size KenKen puzzle and a given section in the puzzle, how many valid ways are there to fill in the section? Well, stop wondering and start programming!

Input

The input will start with a single line of the form $n\ m\ t\ op$, where n is the size of the KenKen puzzle containing the section to be described, m is the number of grid squares in the section, t is the target value and op is either '+', '-', '*' or '/' indicating the arithmetic operator to use for the section.
Next will follow m grid locations of the form $r\ c$, indicating the row and column number of the grid square. These grid square locations will take up one or more lines.
All grid squares in a given section will be connected so that you can move from any one square in the section to any other by crossing shared lines between grid squares.
The values of n, m and t will satisfy $4 \le n \le 9$, $2 \le m \le 10$, $0 < t$ and $1 \le r,\ c \le n$.

Output

Output the number of valid ways in which the section could be filled in for a KenKen puzzle of the given size.

Example

Input	Output
`8 2 7 -`	`2`
`1 1 1 2`	
`9 2 7 -`	`4`
`1 1 1 2`	
`8 3 6 +`	`7`
`5 2 6 2 5 1`	

Problem D. Rings

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

Dee Siduous is a botanist who specializes in trees. A lot of her research has to do with the formation of tree rings, and what they say about the growing conditions over the tree's lifetime. She has a certain theory and wants to run some simulations to see if it holds up to the evidence gathered in the field.
One thing that needs to be done is to determine the expected number of rings given the outline of a tree. Dee has decided to model a cross section of a tree on a two dimenional grid, with the interior of the tree represented by a closed polygon of grid squares. Given this set of squares, she assigns rings from the outer parts of the tree to the inner as follows: calling the non-tree grid squares "ring 0", each ring n is made up of all those grid squares that have at least one ring $(n-1)$ square as a neighbor (where neighboring squares are those that share an edge).
An example of this is shown in the figure below.

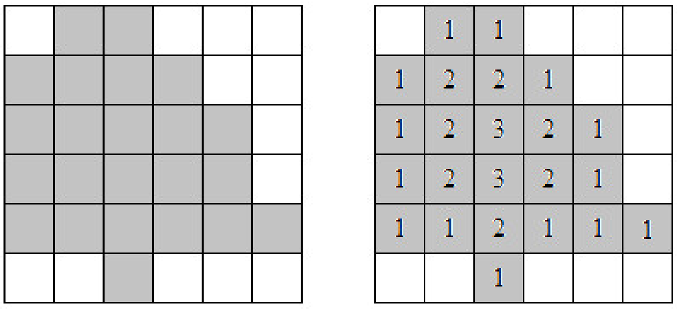


Figure D.1

Most of Dee's models have been drawn on graph paper, and she has come to you to write a program to do this automatically for her. This way she'll use less paper and save some ... well, you know.

Input

The input will start with a line containing two positive integers n m specifying the number of rows and columns in the tree grid, where $n, m \leq 100$. After this will be n rows containing m characters each. These characters will be either 'T' indicating a tree grid square, or '.'.

Output

Output a grid with the ring numbers. If the number of rings is less than 10, use two characters for each grid square; otherwise use three characters for each grid square. Right justify all ring numbers in the grid squares, and use '.' to fill in the remaining characters.
If a row or column does not contain a ring number it should still be output, filled entirely with '.'s.

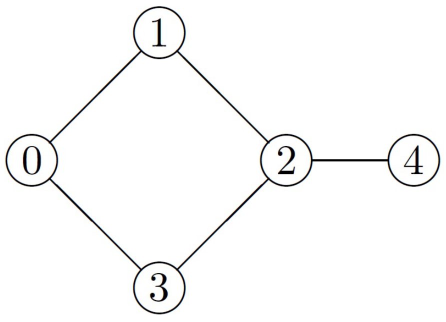
Example

Input	Output
6 6	...1.1......
.TT...	.1.2.2.1....
TTTT..	.1.2.3.2.1..
TTTTT.	.1.2.3.2.1..
TTTTT.	.1.1.2.1.1.1
TTTTTT	.....1......
..T...	
3 4	.1.1....
TT..	.1.1....
TT..	....
....	

Problem E. Squawk Virus

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

Oh no! Hackers are threatening to shut down Twitface, the premier social networking site. By taking advantage of lax security protocols, nefarious cyber-bandits have developed a virus that spreads from user to user, amplifying over time and eventually bringing the network to its knees from massive congestion. Normally users have to manually send messages to one another (squawking), but these ne'er-do-wells have figured out how to circumvent that rule, and have created squawks that spawn more squawks without user intervention. In particular, any time a user gets an infected squawk, one minute later it broadcasts an infected squawk to all its neighbors in the network (for purposes of this problem we assume that each neighbor gets the squawk exactly 1 minute after the initial user is infected). If a user receives multiple squawks at any point, the next minute it broadcasts that many squawks to all of its neighbors. For example, consider the following network:



If user 0 is infected at time $t = 0$, then at time $t = 1$ users 1 and 3 get 1 squawk each, at time $t = 2$ users 0 and 2 get 2 squawks each, and at time $t = 3$, users 1 and 3 get 4 squawks each and user 4 gets 2 squawks.
Given the layout of a social network and an initial infection site, you need to determine how many squawks are made at some given time t. In the above example the number of squawks would be 2, 4 and 10 at times 1, 2 and 3, respectively.

Input

The input will start with a line containing 4 integers n m s t indicating the number of users ($1 \leq n \leq 100$), the number of links between users ($0 \leq m \leq n(n-1)/2$), the index of the initially infected user ($s < n$), and the number of minutes ($t < 10$). Next will follow m lines, each consisting of two integers x y, ($0 \leq x, y < n$) indicating that users x and y are connected. Connections are symmetric and no two connections will be the same.

Output

Output the number of squawks sent at the specified time t.

Example

Input	Output

4 3 1 4
0 1
1 2
2 3
	8

5 5 0 3
0 1
0 3
1 2
2 3
2 4
	10

Problem F. Transportation Delegation

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

You have just been hired by Amalgamated, Inc. in the country of Acmania to oversee the transportation of raw materials to the company's factories. Each supplier of raw materials and each factory resides in one of Acmania's states. No state has both a factory and a supplier (and never more than one of either) and there are arcane laws governing which transportation companies can transport materials across state lines. Because of the fierce competition between factories and between suppliers each transportation company handles the output of at most one raw material site and delivers to at most one factory (or to another transportation company). Each supplier can produce enough material to contract with at most one factory and no factory will contract with more than one supplier. Your job is to determine the maximum number of factories that can be supplied with raw materials.
For example, suppose that there are three suppliers in states A, B and C, and three factories in states D, E and F. Let's say you contract three transportation firms: firm 1 can transport between states A, E and G; firm 2 can transport between states A, C and E; and firm 3 can transport between states B, D and F. In this case, you can supply at most two factories (for example, factory E can be supplied from supplier A using firm 1, and factory F can be supplied from supplier B using firm 3). If you find a fourth firm that transports between states G and F then you can supply all three factories: factory D can be supplied from B using firm 3, factory E can be supplied from C using firm 2, and factory F can be supplied from A using firms 1 and 4.

Input

The input will start with four positive integers s r f t indicating the number of states, raw material sites, factories and transportation companies, where $1 \leq r, f \leq 200$, $r + f \leq s \leq 600$ and $1 \leq t \leq 1000$.
Next will follow a line containing r state names, one for each raw material site.
The next line will contain f state names, one for each factory site.
Finally there will be t lines, one for each transportation company. Each of these lines will start with an integer n, $1 \leq n \leq s$, indicating the number of states the company is allowed to work in, followed by n state names. No state will contain both a raw material site and a factory site.
All state names will be alphabetic strings with no blanks.

Output

Output the maximum number of factories that can be supplied with raw materials.

Example

Input	Output

```
7 3 3 3
A B C
D E F
3 A E G
3 A C E
3 B D F
```
	2

```
7 3 3 4
A B C
D E F
3 A E G
3 A C E
3 B D F
2 G F
```
	3

Problem G. Tray Bien

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

André Claude Marzipan is the head chef at the French restaurant Le Chaud Chien. He owns a vast number of baking trays, which come in two sizes: 1 foot by 1 foot, and 1 foot by 2 feet. He stores them along 3-foot deep shelves of various lengths. For example, on a shelf that is 5 feet long, he might store baking trays in either of the two ways shown below:

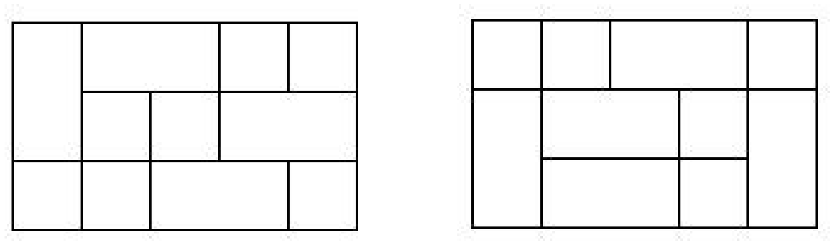


Figure G.1

Of course, there are many more than just these two ways, and in his off hours André often wonders how many different ways he can place trays on a given shelf. André is a bit of *un maniaque du rangement* (neat freak), so he insists that the trays are always aligned along the two axes defined by the shelf edges, that the edges of the trays are always 1 foot multiples away from any edge, and that no portion of a tray extends beyond the shelf. The matter is complicated by the fact that often there are locations on the shelf where he does not want to put any baking trays, due to leaks above the shelf, dents in the shelf's surface, etc. Since André is more adept at cuisine than counting, he needs a little help.

Input

The input consists of two lines: the first line will contain two integers m n, where $1 \leq m \leq 24$ indicates the length of the shelf (which is always 3-feet deep) and n indicates the number of bad locations on the shelf. The next line will contain n coordinate pairs x y indicating the locations where trays should not be placed, where $0 < x < m$ and $0 < y < 3$. No location will have integer coordinates and coordinates are specified to the nearest hundredth. If $n = 0$, this second line will be blank.

Output

Output the number of ways that trays could be placed on the shelf.

Example

Input	Output
`4 0`	`823`

`4 2`
`0.29 2.44 2.73 1.8`
	`149`

Problem H. Paperboy

Source file name:	paperboy.c, paperboy.cpp, paperboy.java
Input:	`standard`
Output:	`standard`

A paperboy wants you to help him optimize his delivery route, which is on one side of a [very] long, linear street. He spends time delivering papers, and time moving between houses. The time to deliver a paper is 1.

The time to move between a pair of houses equals the difference in the house numbers. The paperboy may only carry at most 5 papers with him at once.

The papers are dropped off in front of house number x. The paperboy may only leave excess papers at this house. Whats the smallest amount of time it will take him to deliver the papers?

Input

You will be presented with several test cases composed only of positive integers, one test case per line. Each line will begin with the number $n < 50$, the number of houses, followed by $n + 1$ house numbers. The first n of these are where the paperboy delivers his papers, the last house number is where all n papers are dropped off. (This is the house where he starts his paper route.) House numbers never exceed 100000.

The input ends on EOF.

Output

For each test case, output the minimum amount of time it takes for the paperboy to finish his route.

Example

Input	Output

`4 10 20 30 40 10`
`6 5 15 25 35 45 55 35`
	`34`
	`76`

Problem I. What's on the Grille?

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

The *grille cipher* is a technique that dates back to 1550 when it was first described by Girolamo Cardano. The version we'll be dealing with comes from the late 1800's and works as follows. The message to be encoded is written on an $n \times n$ grid row-wise, top to bottom, and is overlaid with a card with a set of holes punched out of it (this is the grille).
The message is encrypted by writing down the letters that appear in the holes, row by row, then rotating the grille 90 degrees clockwise, writing the new letters that appear, and repeating this process two more times. Of course the holes in the grille must be chosen so that every letter in the message will eventually appear in a hole (this is actually not that hard to arrange).
An example is shown below, where the message "Send more monkeys" is encrypted as "noeesrksdmnyemoj", after adding a random letter to fill out the grid (this example corresponds to the first sample input.)

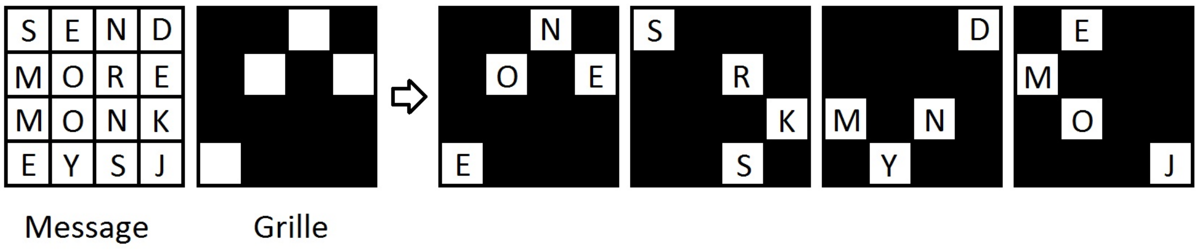


Figure I.1

If the message is larger than the $n \times n$ grid, then the first n^2 letters are written in the grid and encrypted, then the next n^2 are encrypted, and so on, always filling the last grid with random letters if needed. Here, we will only be dealing with messages of length n^2.
Your job, should you choose to accept it, is to take an encrypted message and the corresponding grille and decrypt it. And we'll add one additional twist: the grille given might be invalid, i.e., the holes used do not allow every location in the grid to be used during the encryption process. If this is the case, then you must indicate that you can't decrypt the message.

Input

The input starts with a line containing a positive integer $n \leq 10$ indicating the size of the grid and grille. The next n lines will specify the grille, using '.' for a hole and 'X' for a non-hole. Following this will be a line containing the encrypted message, consisting solely of lowercase alphabetic characters. The number of characters in this line will always be n^2.

Output

Output the decrypted text as a single string with no spaces, or the phrase "invalid grille" if the grille is invalid.

Example

Input	Output
4	sendmoremonkeysj
XX.X	
X.X.	
XXXX	
.XXX	
noeesrksdmnyemoj	
4	invalid grille
.XX.	
XXXX	
XXXX	
.XX.	
abcdefghijklmnop	
2	baby
X.	
XX	
aybb	

Problem J. The Jury

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

You're in charge of a major biochemical release, and you're onto the last step: a jury must approve your product. As you know, a jury consists of an odd number of members, and the majority vote wins. The good news is that you're allowed to choose whatever jury you want!
Each jury member has a fixed probability of approving your product, p_i. Your goal is simple: find the set of people that maximizes your chances of getting approved, and output the probability that you're successful.

Input

The input consists of a series of test cases, one per line. Each test case is on its own line. The test case begins with an integer N, the number of candidate jury members. Following this are $N \leq 1337$ positive integers, representing percentage probabilities $1 \leq p_i \leq 99$ that this jury member approves your product.

Output

For each test case output a single number to 5 decimal places - the best possible acceptance probability.

Example

Input	Output

`3 99 50 50`
`2 75 75`
	`0.99000`
	`0.75000`