Studious Student

You've been given a list of words to study and memorize. Being a diligent student of language and the arts, you've decided to not study them at all and instead make up pointless games based on them. One game you've come up with is to see how you can concatenate the words to generate the lexicographically lowest possible string.

Input

As input for playing this game you will receive a text file containing an integer **N**, the number of word sets you need to play your game against. This will be followed by **N** word sets, each starting with an integer **M**, the number of words in the set, followed by **M** words. All tokens in the input will be separated by some whitespace and, aside from **N** and **M**, will consist entirely of lowercase letters.

Output

Your submission should contain the lexicographically shortest strings for each corresponding word set, one per line and in order.

Constraints

1 <= **N** <= 100
1 <= **M** <= 9
1 <= all word lengths <= 10

Example input

```
5
6 facebook hacker cup for studious students
5 k duz q rc lvraw
5 mybea zdr yubx xe dyroiy
5 jibw ji jp bw jibw
5 uiuy hopji li j dcyi
```

```
Example output

cupfacebookforhackerstudentsstudious
duzklvrawqrc
dyroiymybeaxeyubxzdr
bwjibwjibwjijp
dcyihopjijliuiuy
```

Double Squares

A double-square number is an integer **X** which can be expressed as the sum of two perfect squares. For example, 10 is a double-square because $10 = 3^2 + 1^2$. Your task in this problem is, given **X**, determine the number of ways in which it can be written as the sum of two squares. For example, 10 can only be written as $3^2 + 1^2$ (we don't count $1^2 + 3^2$ as being different). On the other hand, 25 can be written as $5^2 + 0^2$ or as $4^2 + 3^2$.

Input

You should first read an integer **N**, the number of test cases. The next **N** lines will contain **N** values of **X**.

Constraints

$0 \leq \mathbf{X} \leq 2147483647$
$1 \leq \mathbf{N} \leq 100$

Output

For each value of **X**, you should output the number of ways to write **X** as the sum of two squares.

Example input

```
5
10
25
3
0
1
```

Example output

```
1
2
0
1
1
```

Peg Game

At the arcade, you can play a simple game where a ball is dropped into the top of the game, from a position of your choosing. There are a number of pegs that the ball will bounce off of as it drops through the game. Whenever the ball hits a peg, it will bounce to the left with probability 0.5 and to the right with probability 0.5. The one exception to this is when it hits a peg on the far left or right side, in which case it always bounces towards the middle.

When the game was first made, the pegs where arranged in a regular grid. However, it's an old game, and now some of the pegs are missing. Your goal in the game is to get the ball to fall out of the bottom of the game in a specific location. Your task is, given the arrangement of the game, to determine the optimal place to drop the ball, such that the probability of getting it to this specific location is maximized.

The image below shows an example of a game with five rows of five columns. Notice that the top row has five pegs, the next row has four pegs, the next five, and so on. With five columns, there are four choices to drop the ball into (indexed from 0). Note that in this example, there are three pegs missing. The top row is row 0, and the leftmost peg is column 0, so the coordinates of the missing pegs are (1,1), (2,1) and (3,2). In this example, the best place to drop the ball is on the far left, in column 0, which gives a 50% chance that it will end in the goal.

```
x.x.x.x.x
 x...x.x
x...x.x.x
 x.x...x
x.x.x.x.x
 G

'x' indicates a peg, '.' indicates empty space.
```

Input

You should first read an integer **N**, the number of test cases. Each of the next **N** lines will then contain a single test case. Each test case will start with integers **R** and **C**, the number of rows and columns (**R** will be odd). Next, an integer **K** will specify the target column. Finally, an integer **M** will be followed by **M** pairs of integer $\mathbf{r_i}$ and $\mathbf{c_i}$, giving the locations of the missing pegs.

Constraints

- 1 ≤ **N** ≤ 100
- 3 ≤ **R**,**C** ≤ 100
- The top and bottom rows will not have any missing pegs.
- Other parameters will all be valid, given **R** and **C**

Output

For each test case, you should output an integer, the location to drop the ball into, followed by the probability that the ball will end in columns **K**, formatted with exactly six digits after the decimal point (round the last digit, don't truncate).

Notes

The input will be designed such that minor rounding errors will not impact the output (i.e. there will be no ties or near -- up to 1E-9 -- ties, and the direction of rounding for the output will not be impacted by small errors).

Example input

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

```
Example output

0 0.375000
XXX
1 1.000000
0 1.000000
0 0.500000
```