Saturnalia (15pts)

Problem

It is the eve of Saturnalia in the Roman Empire, and Caterina is preparing the stables for the next day's chariot race. Part of her job is to write instructions and notes, print them on her printer (she's ahead of her time), and put them on the stable walls. That's simple, but because Saturnalia is an important festival, she wants to make them beautiful. Caterina needs a computer program that reads a message and outputs it back, decorated with a box. The program needs to be able to handle many messages at once. Can you help her?

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow. Each line contains a text message.

Output

For each test case, output four lines. The first one should contain "Case #x:", where x is the test case number (starting from 1). The next 3 lines should contain the original message surrounded by a box of '+', '-', and '|' characters, with a space character added on each side of the message. See examples below for the exact formatting requirements. Pay special attention to the spaces.

Limits

Small dataset (Test set 1 - Visible)

1 ≤ **T** ≤ 100.
Time limit: 20 seconds.
Memory limit: 1 GB.
Each input line will contain between 1 and 70 characters.
Each character will either be an English letter, a space, or one of the following punctuation characters: ,?!'. (comma, question mark, exclamation point, apostrophe, or period).

Sample

Input

```
5
Merry Saturnalia, Giovanni!
Equus, you're the best!
Caballus, you try really hard!
 
w
```

Output

```
Case #1:
+-----------------------------+
| Merry Saturnalia, Giovanni! |
+-----------------------------+
Case #2:
+-------------------------+
| Equus, you're the best! |
+-------------------------+
Case #3:
+--------------------------------+
| Caballus, you try really hard! |
+--------------------------------+
Case #4:
+-----+
|     |
+-----+
Case #5:
+---+
| w |
+---+
```

Note that the input for Case #4 is a line with 3 space characters on it, so the output is a box with five space characters inside.

Zathras (16pts, 24pts)

Problem

It is year 2025 on planet Zathras -- a world populated exclusively by semi-sentient robots called Zathrinians. There are two kinds of Zathrinians: acrobots and bouncoids. Once a year, the Great Mind makes its Great Decision for that year, and chooses how the Zathrinians will reproduce and be decommissioned. When it's making the Great Decision, it takes into account two Eternal Parameters: **α** and **β**. These parameters, being Eternal, do not change from year to year.

Reproduction: If there are **A** acrobots and **B** bouncoids when the Great Mind makes the Great Decision, the Great Mind will create K = min(**A**, **B**) reproductive pairs by pairing together an acrobot and a bouncoid. Any remaining robots will be unpaired. The next day, 2% of those K couples (rounded down) will produce one baby Zathrinian each.

Out of all the baby Zathrinians produced, **α**% (rounded down) are acrobots, and **β**% (rounded down) are bouncoids. The remaining baby Zathrinians are split evenly between acrobots and bouncoids; if there's an odd number, the extra baby becomes a bouncoid.

Decommissioning: When the Great Mind makes its Great Decision, 1% of acrobots (rounded down) and 1% of bouncoids (rounded down) are marked for decommissioning. Two days later, they will all be disassembled. Note that the 1% figure is calculated on the day of the Great Decision, before the new Zathrinians are born.

After the Great Decision has been made (day 1), the reproduction has occurred (day 2), and the unlucky Zathrinians have been disassembled (day 3), the entire world continues to function in harmony until next year's Great Decision takes place at the time scheduled in the Eternal Specification.

Example

If we start with a population of 12345 acrobots and 12890 bouncoids, 123 acrobots and 128 bouncoids will be marked for decommissioning. The number of couples will be min(12345, 12890), which is 12345. This means that 246 offspring will be created that year. Let's say that **α**=10 and **β**=13, so more bouncoids than Zathrinians are created each year. This means that 24 offspring will be acrobots (10% of 246, rounded down); 31 will be bouncoids (13% of 246, rounded down); and the remaining 191 will be split between 95 more acrobots and 96 more bouncoids.

Overall, we started with 12345 acrobots and 12890 bouncoids. One day later, there will be 12464 acrobots and 13017 bouncoids. The next day, there will be 12341 acrobots and 12889 bouncoids. 99 years later, there will be 11993 acrobots and 12676 bouncoids. After a total of 5049 years, we will be down to only 3099 acrobots and 3199 bouncoids -- a huge drop in total population size. After that, the populations will remain the same forever.

Given the values of **A**, **B**, **α**, **β**, and **Y**, can you compute the acrobot and bouncoid population sizes at the end of **Y** years?

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow. Each line contains 5 integers: **A**, **B**, **α**, **β**, and **Y**.

Output

For each test case, output one line containing "Case #x: A_Y B_Y", where x is the test case number (starting from 1) and (A_Y, B_Y) are the populations of acrobots and bouncoids after **Y** years, respectively.

Limits

1 ≤ **T** ≤ 100.
Time limit: 20 seconds per test set.
Memory limit: 1 GB.
0 ≤ **α**.
0 ≤ **β**.
α + **β** ≤ 100.

Small dataset (Test set 1 - Visible)

0 ≤ **A** ≤ 20000.
0 ≤ **B** ≤ 20000.
0 ≤ **Y** ≤ 100.

Large dataset (Test set 2 - Hidden)

0 ≤ **A** ≤ 10^6.
0 ≤ **B** ≤ 10^6.
0 ≤ **Y** ≤ 10^{15}.

Sample

Input

```
4
12345 12890 10 13 0
12345 12890 10 13 1
12345 12890 10 13 100
12345 12890 10 13 5049
```

Output

```
Case #1: 12345 12890
Case #2: 12341 12889
Case #3: 11993 12676
Case #4: 3099 3199
```

Seating Chart (17pts, 28pts)

Problem

Some people believe that the easiest way to ruin a conference is to do a bad job of planning the seating arrangements. The conference's chairperson, Saanvi, is planning seating for the dinner after the keynote address, with **N** people, and she wants to manually review all possible seating arrangements in order to pick the absolutely best one. To figure out whether that's feasible, she's planning to write a program to compute the number of possible seating arrangements.

There are **K** round tables at the dinner, numbered 1 through **K**. It is important to have exactly the same number of people sitting at each table. If that is impossible (**N** is not divisible by **K**), then the table with the most people must have at most one more person sitting at it than the table with the fewest people.

Each of the **N** people will be assigned a unique number between 0 and **N** - 1. What matters is who is sitting next to whom, and not exactly where they're sitting. In other words, two arrangements, A and B, are considered different if there exists a pair of numbers, α and β, such that persons α and β are sitting next to each other at the same table in arrangement A, but they are not sitting next to each other in arrangement B.

For example, if **N** is 5, and **K** is 2, we must have 3 people seated at one of the tables, and 2 people seated at the other table. Here is the list of all 10 of the possible arrangements:

```
[[0, 1, 2], [3, 4]]
[[0, 1, 3], [2, 4]]
[[0, 1, 4], [2, 3]]
[[0, 2, 3], [1, 4]]
[[0, 2, 4], [1, 3]]
[[0, 3, 4], [1, 2]]
[[1, 2, 3], [0, 4]]
[[1, 2, 4], [0, 3]]
[[1, 3, 4], [0, 2]]
[[2, 3, 4], [0, 1]]
```

All other arrangements are similar to one of the arrangements above and are not counted as different. In particular, all of the following arrangements are considered to be the same:

```
[[0, 1, 2], [3, 4]]
[[2, 0, 1], [3, 4]]
[[1, 2, 0], [4, 3]]
[[0, 2, 1], [3, 4]]
[[3, 4], [0, 2, 1]]
```

This is because the following pairs of people (and no other pairs) are sitting next to each other in each of these 5 arrangements:

```
0 and 1
0 and 2
```

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

Another example is **N** = 5 and **K** = 3, which requires having two tables with two people each, and one table with a single person sitting at it. There are 15 possible arrangements in this case:

```
[[0, 1], [2, 3], [4]]
[[0, 1], [2, 4], [3]]
[[0, 1], [3, 4], [2]]
[[0, 2], [1, 3], [4]]
[[0, 2], [1, 4], [3]]
[[0, 2], [3, 4], [1]]
[[0, 3], [1, 2], [4]]
[[0, 3], [1, 4], [2]]
[[0, 3], [2, 4], [1]]
[[0, 4], [1, 2], [3]]
[[0, 4], [1, 3], [2]]
[[0, 4], [2, 3], [1]]
[[1, 2], [3, 4], [0]]
[[1, 3], [2, 4], [0]]
[[1, 4], [2, 3], [0]]
```

In this final example, **N** = 5 and **K** = 1, which means that we only have a single table, seating all 5 guests. Here, the answer is 12:

```
[[0, 1, 2, 3, 4]]
[[0, 1, 2, 4, 3]]
[[0, 1, 3, 2, 4]]
[[0, 1, 3, 4, 2]]
[[0, 1, 4, 2, 3]]
[[0, 1, 4, 3, 2]]
[[0, 2, 1, 3, 4]]
[[0, 2, 1, 4, 3]]
[[0, 2, 3, 1, 4]]
[[0, 2, 4, 1, 3]]
[[0, 3, 1, 2, 4]]
[[0, 3, 2, 1, 4]]
```

Input

The first line of the input gives the number of test cases, **T**. **T** lines, each one containing two integers, **N** and **K**.

Output

For each test case, output one line containing "Case #x: y", where x is the test case number (starting from 1) and y is the number of different possible seating arrangements.

Limits

Time limit: 20 seconds per test set.
Memory limit: 1 GB.
1 ≤ **K** ≤ **N**.

Small dataset (Test set 1 - Visible)

1 ≤ **T** ≤ 36.
1 ≤ **N** ≤ 8.

Large dataset (Test set 2 - Hidden)

1 ≤ **T** ≤ 210.
1 ≤ **N** ≤ 20.

Sample

Input Output

```
5
5 2      Case #1: 10
5 3      Case #2: 15
5 4      Case #3: 10
5 1      Case #4: 12
1 1      Case #5: 1
```

I/O Error (7pts)

Problem

Our computers are so excited about the upcoming Google I/O that they've started storing their ones as capital letter Is and their zeroes as capital letter Os! For example, the character `A`, which is 65 in ASCII, would normally be stored as the byte `01000001`, but our computers are storing it as `OIOOOOOI`.

Given a string of 8-character "bytes" consisting of `I`s and `O`s, can you translate it using ASCII? Every "byte" is guaranteed to translate to a printable character (a decimal value between 32 and 126, inclusive). Note that one of these characters (the one with decimal value 32) is a space. No translated message will begin or end with a space, but there may be internal space characters.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow; each consists of two lines. The first line of each test case contains an integer representing the number **B** of "bytes" in the string to be translated. The second line of each test case contains 8 * **B** characters, all of which are either `I` or `O`.

Output

For each test case, output one line containing "Case #x: y", where x is the test case number (starting from 1) and y is the translated message.

Limits

Small dataset (Test set 1 - Visible)

1 ≤ **T** ≤ 100.
Time limit: 20 seconds.
Memory limit: 1 GB.
1 ≤ **B** ≤ 1000.

Sample

Input

```
2
2
OIOOIIIIOIOOIOII
21
OIOOIOOIOOIOOOOOOOIOOIIIOOIIIIOOOOIIOOIIOOIOOIIIOOIOOOOOOOIOOOIOOIOOOOIIOOIIOOOOOIIOO
IOOOOIIOOIIOOIOOOOOOIOOIOIOOOIIOIOOOIIOIIOIOOIOOOIOOOIOOOOIOOIOOOOOOOIIIOIOOOIOIOOI
```

Output

```
Case #1: OK
Case #2: I '<3' "C0d3 J4m"! :)
```

Power Levels (9pts, 16pts)

Problem

A *multifactorial* -- that is, a number N followed by some nonzero number E of exclamation points -- is defined as the product of all **positive** numbers (N - K*E) for which K is a nonnegative integer. Normal factorials meet the definition of multifactorials, for example:

5! = 5 * (5-1) * (5-2) * (5-3) * (5-4) = 120

Here are the other multifactorials of 5:

5!! equals 5 * (5-2) * (5-4) = 15

5!!! equals 5 * (5-3) = 10

5!!!! equals 5 * (5-4) = 5

5 followed by five or more !s = 5

5 with no exclamation points is not considered a multifactorial.

The villainous Anima and her underling Minera love three things: multifactorials, yelling "IT'S OVER 9000" followed by some number of exclamation points, and going around the universe looking for people to fight.

When Anima and Minera encounter a potential opponent, Anima asks Minera to use her scanner device to determine the opponent's power level, which is always a positive integer that does not begin with any leading zeroes. Today, the display on Minera's scanner is blurry, and she can only tell Anima the number of digits **D** in the opponent's power level, and not what any of those digits are.

Anima wants to yell IT'S OVER followed by a multifactorial of 9000 to accurately describe the opponent's power level. She will never yell something that might not be true, and she will never use more exclamation points than she needs to. For example, if **D** = 31682, Anima can't yell IT'S OVER 9000!, because she knows that 9000! also has 31682 digits and the opponent's power level might be a 31682-digit number that is less than or equal to 9000!. However, since 9000!! has fewer than 31682 digits, she can be confident that the opponent's power level is greater than 9000!!, so IT'S OVER 9000!! is guaranteed to be true. Although the opponent's power level is also definitely greater than 9000!!!, 9000!!!!, and so on, she won't use more exclamation points than she needs to. So, in this example, she will yell IT'S OVER 9000!!

If there is no multifactorial of 9000 that is definitely less than the opponent's power level, Anima will remain dramatically silent. We represent dramatic silence as an ellipsis: `...`

What should Anima say?

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow. Each contains a positive integer **D**, the number of digits in an opponent's power level.

Output

For each test case, output one line containing "Case #x: y", where x is the test case number (starting from 1) and y is either `...` if Anima must remain silent, or `IT'S OVER 9000` followed by some positive number of exclamation points. Make sure your output exactly matches these specifications. You may wish to copy our apostrophe character into your code to avoid possible Unicode issues.

Limits

Time limit: 20 seconds per test set.
Memory limit: 1 GB.

Small dataset (Test set 1 - Visible)

T = 19.
1 ≤ **D** ≤ 19.

Large dataset (Test set 2 - Hidden)

1 ≤ **T** ≤ 100.
1 ≤ **D** ≤ 100000.

Sample

Input

```
5
```

```
1
19
206
31682
31683
```

Output

```
Case #1: ...
Case #2: IT'S OVER
9000!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Case #3: IT'S OVER
9000!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Case #4: IT'S OVER 9000!!
Case #5: IT'S OVER 9000!
```

Note that sample cases 3 through 5 are outside the range of the Small data set.

In Case #1, Anima knows that all multifactorials of 9000 are at least 9000, so the opponent's single-digit power level, whatever it is, can't exceed any of them.

The output for Case #2, which is the largest possible input for the Small dataset, has 1990 exclamation points, in case you don't wish to count them.

In Case #3, both 9000 with 156 exclamation points and 9000 with 157 exclamation points have 205 digits, whereas 9000 with 155 exclamation points has 207 digits. Anima could make the claim with either 156 or 157, but she always chooses as few exclamation points as possible. This output has 156 exclamation points.

Googlander (11pts, 19pts)

Problem

Eric Googlander is a fashion model who performs by walking around on a stage made of squares that form a grid with **R** rows and **C** columns. He begins at the leftmost bottom square, facing toward the top edge of the stage, and he will perform by making a series of moves. Googlander knows only the following two moves:

1. Take one step forward in the direction he is currently facing

2. Make a single 90 degree turn to the right, then take one step forward in the new direction he is facing following the turn

(Note that Googlander does not know how to make a 90 degree *left* turn.)

If a move would take Googlander off of the stage or onto a square he has already visited, that move is *unfashionable*. Whenever Googlander is in a position for which neither of the two possible moves is unfashionable, he is free to choose either move (independently of any other choices he has made in the past), but he must choose one of them. Whenever one of the possible moves he can make is unfashionable, he must make the other move. If at any point both of the possible moves are unfashionable, the show immediately ends without Googlander making another move. Note that Googlander cannot stop the show early -- he must keep moving until both possible moves become unfashionable.

How many different paths is it possible for Googlander to walk? (Two paths are the same if and only if they visit the same squares in the same order.)

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow; each consists of two space-separated integers **R** and **C**.

Output

For each test case, output one line containing "Case #x: y", where x is the test case number (starting from 1) and y is the number of different paths that Googlander can walk.

Limits

1 ≤ **T** ≤ 100.
Time limit: 20 seconds per test set.
Memory limit: 1 GB.

Small dataset (Test set 1 - Visible)

1 ≤ **R, C** ≤ 10.
The limits ensure that the answer will always fit in a 32-bit signed integer.

Large dataset (Test set 2 - Hidden)

1 ≤ **R, C** ≤ 25.
The limits ensure that the answer will always fit in a 64-bit signed integer.

Sample

Input Output

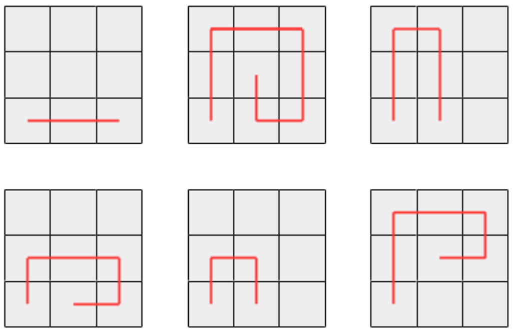
```
3
1 1
1 3
3 3
```

```
Case #1: 1
Case #2: 1
Case #3: 6
```

In Case #1, Googlander cannot make any moves. The only possible path is the trivial one consisting of the only square.

In Case #2, Googlander cannot take a step straight ahead, because it would take him off the stage, but he can turn right and then take a step. Once he has done so, he cannot turn right and then take a step, but he can take a step straight ahead. At that point, there are no more moves he can make and the show is over. This is the only possible path he can take.

In Case #3, these are the possible paths:



Dreary Design (8pts, 10pts, 20pts)

Problem

One way to represent a color is as a triple of component values (each of which can range from 0 to **K**, inclusive) representing levels of red, green, and blue. For example, in the color system where **K** = 3, (0, 2, 3) and (0, 3, 2) would be two of the possible distinct colors.

We will consider a color to be *bland* if and only if all pairs of its component values differ by no more than **V**. For example, in a system with **K** = 2 and **V** = 1, the color (2, 1, 1) is bland, because its red and green components differ by 1, its red and blue components differ by 1, and its green and blue components differ by 0, and none of these differences exceeds 1. But (2, 1, 0) is not bland, because the red and blue components differ by more than 1.

Mr. Turner loves to create gloomy landscape images and wants to design a color system in which there are many bland colors available. Given values for **K** and **V**, can you tell him how many distinct bland colors are there?

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow. Each contains two space-separated integers **K** and **V**.

Output

For each test case, output one line containing "Case #x: y", where x is the test case number (starting from 1) and y is the number of distinct bland colors.

Limits

1 ≤ **T** ≤ 100.
Time limit: 20 seconds per test set.
Memory limit: 1 GB.
V ≤ **K**.

Small dataset (Test set 1 - Visible)

0 ≤ **K** ≤ 255.
0 ≤ **V** ≤ 100.
All answers are guaranteed to fit in a 32-bit signed integer.

Large dataset 1 (Test set 2 - Hidden)

0 ≤ **K** ≤ 2,555.
0 ≤ **V** ≤ 555.
All answers are guaranteed to fit in a 32-bit signed integer.

Large dataset 2 (Test set 3 - Hidden)

0 ≤ **K** ≤ 2,000,000,000.
0 ≤ **V** ≤ 1,000.
All answers are guaranteed to fit in a 64-bit signed integer.

Sample

Input	Output

```
4
1 1
1 0
255 0
0 0
```
	```
	Case #1: 8
	Case #2: 2
	Case #3: 256
	Case #4: 1
	```

In Case #1, there are eight possible colors -- (0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1), (1, 0, 0), (1, 0, 1), (1, 1, 0), and (1, 1, 1) -- and all of them meet the definition of bland for **V** = 1.

In Case #2, the same eight colors are possible, but only two of them -- (0, 0, 0) and (1, 1, 1) -- meet the definition of bland for **V** = 0.

Grid Escape (7pts, 12pts)

Problem

You are designing a new "escape" adventure that uses a rectangular grid of rooms (unit cells) with **R** rows and **C** columns. Each room has four doors oriented in the four orthogonal directions of the grid: north, south, east, and west. The doors on the border of the grid lead outside, and all of the other doors lead to other rooms.

The adventure will be played by exactly **R** × **C** players, with each player starting in a different one of the **R** × **C** rooms. Once everyone is in position and the game starts, all of the doors close, and there is a mechanical trick: one of the four doors in each room can be opened from inside the room, and the other three doors cannot be opened. This remains consistent throughout the adventure; in a given room, it is always the same door that can be opened. Notice that it is possible that a door that connects two rooms might be able to be opened from one side but not the other.

Each player moves independently of all other players. Players can only go through doors that they opened themselves, and they must close doors behind them. Each player will keep going through doors until they go through a door that leads outside (and therefore they escape), or they have made **R** × **C** moves without escaping (at which point they are considered to have failed, and they do not escape).

You want to choose which door in each room can be opened, such that exactly **K** of the players will be able to escape. Can you find a way to do this, or determine that it is `IMPOSSIBLE`?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each consists of one line containing three integers **R**, **C**, and **K**, as described above.

Output

For each test case, output one line containing `Case #x: y`, where `x` is the test case number (starting from 1) and `y` is `IMPOSSIBLE` if there is no solution, or `POSSIBLE` if there is. If there is a solution, output **R** more lines of **C** characters each, representing the grid of rooms. The j-th character on the i-th of these lines represents the room in the i-th row and j-th column of the grid; each character must be either uppercase `N`, `S`, `E`, or `W`, according to whether the door that opens from that room is the one that leads north, south, east, or west.

If multiple answers are possible, you may output any one of them.

Limits

1 ≤ **T** ≤ 100.
Time limit: 20 seconds per test set. (10 seconds per test run.)
Memory limit: 1GB.
1 ≤ **R**.
1 ≤ **C**.
0 ≤ **K** ≤ **R** × **C**.

Test set 1 (Visible)

1 ≤ (**R** × **C**) ≤ 8.

Test set 2 (Hidden)

1 ≤ (**R** × **C**) ≤ 100.

Sample

Input	Output

```
2
2 3 2
1 1 0
```
	```
	Case #1: POSSIBLE
	SES
	SNW
	Case #2: IMPOSSIBLE
	```

In our solution for Sample Case #1, the two players who start in the westernmost two rooms will go south until they escape, whereas the four players who start in the other four rooms will travel between those rooms in an endless clockwise circle and cannot escape.

In Sample Case #2, there is only one room, so the player can definitely escape regardless of which particular door can be opened.

Parcel Posts (7pts, 13pts)

Problem

You just bought a parcel of land that is **K** kilometers long; it is so narrow that, for the purposes of this problem, we will consider it to be a polyline that runs from west to east, varying in elevation. You know the elevations of the land (in meters) at **K** + 1 regularly spaced *measurement marks* $\mathbf{M_0}$, $\mathbf{M_1}$, ..., $\mathbf{M_K}$. These marks are 0, 1, ..., **K** km, respectively, from the western end.

In this region, a wooden post denotes the boundary between two adjacent parcels of land. Wooden posts can only be placed at measurement marks, and there can be at most one post at each mark. Right now, there are two posts: one at the 0 km mark, and one at the **K** km mark. A measurement mark with a post is considered to be part of both of the parcels it demarcates, so your parcel of land includes all measurement marks between 0 and **K** km, inclusive.

A parcel is considered *desirable* if it contains three measurement marks such that the west-most and east-most of those three marks are both strictly higher than the remaining one of the three marks, or both strictly lower than the remaining one of the three marks. People like some variation in their landscapes! Notice that these three marks are not necessarily consecutive, and the west-most and east-most of the three marks are not necessarily the west-most and east-most marks of the parcel.

Consider an example with **K** = 5 and $\mathbf{M_0}$, $\mathbf{M_1}$, ..., $\mathbf{M_K}$ = 5, 6, 6, 1, 2, 4. The measurement marks with elevations 5, 2, and 4 satisfy the condition, as do the measurement marks with elevations 6, 1, and 2. However, the measurement marks with elevations 6, 6, and 1 do not satisfy the condition, nor do the measurement marks with elevations 1, 2, and 4. Any three measurement marks that satisfy the condition make the whole parcel desirable; for example, a parcel containing the measurement marks 4 7 6 7 is desirable because of the first three values.

Your parcel is desirable, but you think it may be possible to extract even more value from it! You want to add additional posts to this parcel to divide it up into multiple parcels, all of which must be desirable, since you do not want to waste any land. What is the largest number of posts you can add?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each case begins with one line containing an integer **K**: the length, in kilometers, of your parcel of land. Then, there is one more line with **K** + 1 integers $\mathbf{M_0}$, $\mathbf{M_1}$, ..., $\mathbf{M_k}$; where $\mathbf{M_i}$ is the elevation (in meters) at the measurement mark that is i km from the western end of your land.

Output

For each test case, output one line containing `Case #x: y`, where `x` is the test case number (starting from 1) and `y` is the largest possible number of posts you can add, as described above.

Limits

Time limit: 20 seconds per test set. (10 seconds per test run.)
Memory limit: 1GB.
$1 \leq \mathbf{T} \leq 100$.
$0 \leq \mathbf{M_i} \leq 1000$, for all i.
$(\mathbf{M_i} - \mathbf{M_j}) \times (\mathbf{M_k} - \mathbf{M_j}) > 0$, for some $i < j < k$. (Your starting parcel is desirable.)

Test set 1 (Visible)

$4 \leq \mathbf{K} \leq 10$.

Test set 2 (Hidden)

$4 \leq \mathbf{K} \leq 1000$.

Sample

Input	Output

```
4
4
4 8 7 3 5
4
4 8 7 7 5
7
1 2 2 1 2 1 2 1
6
2 1 3 10 9 12 20
```
	```
	Case #1: 1
	Case #2: 0
	Case #3: 2
	Case #4: 1
	```

In Sample Case #1, you can add one post at 2 km to get a total of two desirable parcels. The parcel from 0 to 2 km is desirable because 4 < 8 and 8 > 7. The parcel from 2 to 4 km is desirable because 7 > 3 and 3 < 5.

In Sample Case #2, there is no way to add another post. If you added one at 1 km or 3 km, one of the parcels would include only two measurement marks and could not be desirable. If you added one at 2 km, the parcel between 0 and 2 km would be desirable, but the parcel between 2 and 4 km would not.

In Sample Case #3, posts can be added at 3 km and 5 km.

In Sample Case #4, a post can be added at 2 km. Notice that the parcel from 2 km to 6 km is desirable because 10 > 9 and 9 < 12. However, there is no way to add a second post.

Sheepwalking (9pts, 17pts)

Problem

Bleatrix Trotter is a sheep who lives in a two-dimensional field that is an infinite grid of unit cells. Her home is in a unit cell that we denote as (0, 0) — that is, all coordinates are given relative to Bleatrix's home cell. However, because she has been sleepwalking, she is currently in the unit cell at the coordinates (**X**, **Y**) — that is, in a cell **X** columns east of, and **Y** rows north of, her home cell. The two sheepdogs who have been assigned to protect Bleatrix have just noticed that she is missing, and now they want to herd her back to her home cell.

Before each of Bleatrix's moves, the two sheepdogs can move to any grid cells that they want, except that they cannot both move to the same cell, and neither one can move to Bleatrix's current cell. Once the sheepdogs are in place, Bleatrix, who is sleepwalking, will make a random unit move in a direction that would not take her into a cell with a sheepdog. That is, she takes the set of four possible unit moves (north, south, west, east), discards any that would move her into a cell with a sheepdog, and then chooses uniformly at random from the remaining moves. Then the sheepdogs can position themselves again, and so on (notice that, unlike Bleatrix, the sheepdogs do not have to make unit moves).

Once Bleatrix arrives at her home at (0, 0), she stops sleepwalking, wakes up, and grazes peacefully, and does not make any more moves thereafter.

If the sheepdogs coordinate their movements to minimize the expected number of Bleatrix's moves to reach her home, what is that expected number?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each case consists of one line with two integers **X** and **Y**, representing the coordinates of Bleatrix's starting cell, as described above.

Output

For each test case, output one line containing `Case #x: y`, where `x` is the test case number (starting from 1) and `y` is the expected number of Bleatrix's moves, as described above. `y` will be considered correct if it is within an absolute or relative error of 10^{-6} of the correct answer. See the Competing section of the FAQ for an explanation of what that means, and what formats of real numbers we accept.

Limits

Time limit: 20 seconds per test set. (10 seconds per test run.)
Memory limit: 1GB.
(**X**, **Y**) ≠ (0, 0).

Test set 1 (Visible)

1 ≤ **T** ≤ 48.
-3 ≤ **X** ≤ 3.
-3 ≤ **Y** ≤ 3.

Test set 2 (Hidden)

1 ≤ **T** ≤ 100.
-500 ≤ **X** ≤ 500.
-500 ≤ **Y** ≤ 500.

Sample

Input	Output

`1`
`-1 1`
	`Case #1: 4.000000`

Notice that the values of **X** and/or **Y** may be negative. An **X** value of -1, for example, means that the cell is one unit west of Bleatrix's home cell. (Similarly, a negative value of **Y** means the cell is south of Bleatrix's home cell.)

In the sample case, Bleatrix starts off one cell to the north of, and one cell to the west of, her home. Before she makes her first move, the two sheepdogs could position themselves in cells (-2, 1) and (-1, 2). Then, whichever direction she might choose, she would end up only one step away from her home… but the sheepdogs could not guarantee that she would go there on her next move! The remaining details are left for you to discover.

War of the Words (11pts, 24pts)

Problem

Finally, you are face to face with the leader of the alien robots! You have managed to distract it with its favorite word game, while your fellow Resistance members try to shut down the robots' power supply.

The game uses the robot language, which is made up of 10^5 words: the set of all five-letter strings made up of uppercase English letters between `A` and `J`, inclusive. For example, `AAAAA`, `FGHIJ`, and `BEIGE` are among the valid words.

You know that these words have a rank order, with no ties, such that a higher-ranked word beats any lower-ranked word, with the one exception that the lowest-ranked word beats the highest-ranked word. Unfortunately, you do not know the rank order of the words in the robot language!

The game has the following rules:

- Turn 0: You start by naming a word W_0.
- Turn 1: The robot names a word W_1. If W_1 beats W_0, the robot scores a point.
- This continues; on turn i, the active player is you if i is even, or the robot if i is odd. The active player names a word W_i and scores a point if W_i beats W_{i-1}. (If the two words are the same, no point is scored.) The score is not announced — in particular, you will not know whether each of your words has scored a point!
- This continues for a total of 201 turns.
- Notice that you get the last turn (naming W_{200}), and that at the end of that turn, each player's score is between 0 and 100, inclusive.

Thanks to some great work by the Resistance's spies, you know the strategy that the robot will use for every turn of the game. It only cares about scoring 100 points, so on every turn, it will choose a word uniformly at random from all possible words that will score a point on that turn, and independently of all of its previous word choices. The robot knows the rank order of the language, so it has no trouble choosing words!

If you do not score at least **N** points, the robot will become bored and stop playing with you, so your plan (and the universe) will be doomed!

Input and output

This problem is interactive, which means that the concepts of input and output are different than in standard Code Jam problems. You will interact with a separate process that both provides you with information and evaluates your responses. All information comes into your program via standard input; anything that you need to communicate should be sent via standard output. Remember that many programming languages buffer the output by default, so make sure your output actually goes out (for instance, by flushing the buffer) before blocking to wait for a response. See the FAQ for an explanation of what it means to flush the buffer. Anything your program sends through standard error is ignored, but it might consume some memory and be counted against your memory limit, so do not overflow it.

In addition, sample solutions to a previous Code Jam interactive problem (in various languages) are provided in the Analysis section of this past problem.

Initially, your program should read a single line containing two integers **T** and **N**: the number of test cases, and the number of points you must score in each case. (Notice that the **N** parameter is the same for every test case in a test set.) Then, you need to process **T** test cases.

At the start of a test case, the judge chooses a rank order for the words in the robot language, uniformly at random from all such orders, and independently of any previous choices of rank order. (However, the judge uses a deterministic seed, so if your program is deterministic, its interaction with the judge will be the same even across submissions for this problem.)

At the start of a test case, you must output one line with a five-letter string of uppercase English letters in the inclusive range `A` through `J`: a valid word W_0 in the robot language. Then you will have 100 exchanges with the judge; the j-th of these (counting starting from 1) proceeds as follows:

1. The judge will output one line with a valid word $\mathbf{W_{2j-1}}$: the robot's word.
2. You output one line with another valid word W_{2j}.

After you send your last word for a test case, your program should terminate if it was the last test case; otherwise, it should output a word to begin the next test case.

If your program outputs something wrong (e.g., gives an invalid word), the judge will send `-1` to your input stream and it will not send any other output after that. If your program continues to wait for the judge after receiving `-1`, your program will time out, resulting in a Time Limit Exceeded error. Notice that it is your responsibility to have your program exit in time to receive a Wrong Answer judgment instead of a Time Limit Exceeded error. As usual, if the total time or memory is exceeded, or your program gets a runtime error, you will receive the appropriate judgment.

You should not send additional information to the judge after processing all test cases. In other words, if your program keeps printing to standard output after the last test case, you will get a Wrong Answer judgment.

Limits

$1 \le \mathbf{T} \le 50$.
Time limit: 40 seconds per test set. (10 seconds per test run.)
Memory limit: 1GB.
$\mathbf{W_i}$ is five characters long and consists only of characters in the set of uppercase letters `ABCDEFGHIJ`, for all odd i. (The robot plays valid words.)

Test set 1 (Visible)

$\mathbf{N} = 25$.

Test set 2 (Visible)

$\mathbf{N} = 50$.

Sample Interaction

Notice that interactive problems do not have sample input and output like other Code Jam problems. Your code will not be run against samples, and the following interaction is just an example.

```
  t, n = readline_int_list()   // reads 50, 25 into t, n
  // Before the case starts, the judge silently chooses a rank order for the
  // 100000 words. Suppose that the word FADED is the lowest-ranked word,
  // AHEAD is the highest-ranked word, and CAGED and HIGCJ are ranked
  // such that FADED < CAGED < HIGCJ < AHEAD. As the contestant,
  // though, we do not know this!
  printline AHEAD to stdout    // you name AHEAD as word 0. What luck!
  flush stdout
  robot_word = readline_str()  // reads FADED into robot_word
  printline CAGED to stdout    // you name CAGED as word 2
  flush stdout
  robot_word = readline_str()  // reads HIGCJ into robot_word
  printline GAFFED to stdout   // you name an invalid word as word 4
  flush stdout
  robot_word = readline_str()  // reads -1 (judge has decided our solution is
                               //   incorrect)
  exit                         // exits to avoid an ambiguous TLE error
```

Interactive Runner

To test this problem locally, download the following as interactive_runner.py, and run it in conjunction with the testing tool downloadable below it.

```
# This is a small program that runs two processes, connecting the
# stdin of each one to the stdout of the other.
# It doesn't perform a lot of checking, so many errors may
# be caught internally by Python (e.g., if your command line has incorrect
# syntax) or not caught at all (e.g., if the judge or solution hangs).
#
# Run this as:
# python interactive_runner.py  --
#
# For example:
# python interactive_runner.py python testing_tool.py 0 -- ./my_binary
#
# This will run the first test set of a python judge called
# "testing_tool.py" that receives the test set number (starting from 0)
# via command line parameter with a solution compiled into a binary called
# "my_binary".
#
# This is only intended as a convenient tool to help contestants test solutions
# locally. In particular, it is not identical to the implementation on our
# server, which is more complex.

from __future__ import print_function
import sys, subprocess, threading

class SubprocessThread(threading.Thread):
  def __init__(self,
               args,
               stdin_pipe=subprocess.PIPE,
               stdout_pipe=subprocess.PIPE,
               stderr_pipe=subprocess.PIPE):
    threading.Thread.__init__(self)
    self.p = subprocess.Popen(
        args,
        stdin=stdin_pipe,
        stdout=stdout_pipe,
        stderr=stderr_pipe)

  def run(self):
    try:
      self.return_code = self.p.wait()
      self.stdout = "" if self.p.stdout is None else self.p.stdout.read()
      self.stderr = "" if self.p.stderr is None else self.p.stderr.read()
    except (SystemError, OSError):
      self.return_code = -1
      self.stdout = ""
      self.stderr = "The process crashed or produced too much output."

assert sys.argv.count("--") == 1, (
    "There should be exactly one instance of '--' in the command line.")
sep_index = sys.argv.index("--")
judge_args = sys.argv[1:sep_index]
sol_args = sys.argv[sep_index + 1:]

t_sol = SubprocessThread(sol_args)
t_judge = SubprocessThread(judge_args, stdin_pipe=t_sol.p.stdout,
                           stdout_pipe=t_sol.p.stdin)
t_sol.start()
t_judge.start()
t_sol.join()
t_judge.join()
print("Judge return code:", t_judge.return_code)
print("Judge standard error:", t_judge.stderr.decode())
print("Solution return code:", t_sol.return_code)
print("Solution standard error:", t_sol.stderr.decode())
```

Local Testing Tool

To better facilitate local testing, we provide you the following script. Instructions are included inside. You are encouraged to add more test cases for better testing. Please be advised that although the testing tool is intended to simulate the judging system, it is **NOT** the real judging system and might behave differently.

If your code passes the testing tool but fails the real judge, please check here to make sure that you are using the same compiler as us.

Burger Optimization (5pts, 9pts)

Problem

In 2017, Google learned about a serious Android bug: in the burger emoji, the cheese was directly on top of the lower bun, rather than on the patty itself. Really, who makes a burger that way? Sundar, our CEO, vowed to drop everything and address this issue immediately.

To prevent this sort of situation in the future, the Code Jam team has constructed a mathematical model for understanding burgers. A burger consists of a stack of **K** ingredients between two buns, with each ingredient appearing exactly once. We are interested in the *distance-to-bun* value of each ingredient. The distance-to-bun value of an ingredient is the minimum number of other ingredients between the that ingredient and a bun:

- If **K** is even, then the distance-to-bun values for the ingredients (starting with the ingredient at the top of the stack) are: 0, 1, ..., **K**/2 - 1, **K**/2 - 1, ..., 1, 0.
- If **K** is odd, then they are: 0, 1, ..., ((**K** - 1) / 2) - 1, (**K** - 1) / 2, ((**K** - 1) / 2) - 1, ..., 1, 0.

After carrying out a lot of focus group testing (and eating a lot of burgers), we have determined that the i-th of each of our **K** ingredients has an optimal distance-to-bun value of $\mathbf{D_i}$. We think our burger emoji users will be happiest if we choose an ordering for our ingredients that minimizes the *error* value, which we define as the sum of the squared differences between each ingredient's optimal and actual distance-to-bun values.

For example, if we have five ingredients A, B, C, D, and E with optimal distance-to-bun values of 0, 2, 1, 1, and 2, and we place them between the buns in that order, then the error is $(0-0)^2 + (2-1)^2 + (1-2)^2 + (1-1)^2 + (2-0)^2 = 6$. If we instead place them in the order C, E, B, D, A, then the error is $(1-0)^2 + (2-1)^2 + (2-2)^2 + (1-1)^2 + (0-0)^2 = 2$, which turns out to be the minimum possible error for these ingredients.

Given the list of optimal distance-to-bun values for our ingredients, can you help us determine the smallest possible error?

Input

The first line of the input gives the number of test cases, **T**; **T** test cases follow. Each begins with one line containing an integer **K**: the number of ingredients in our burger. Then, there is one more line containing **K** integers $\mathbf{D_i}$, the optimal distance-to-bun values of our ingredients.

Output

For each test case, output one line containing `Case #x: y`, where `x` is the test case number (starting from 1) and `y` is the smallest possible error, as described above.

Limits

1 ≤ **T** ≤ 100.
Time limit: 20 seconds per test set.
Memory limit: 1GB.
0 ≤ $\mathbf{D_i}$ ≤ floor((**K**-1)/2), for all i. (Each optimal distance-to-bun value is within the range of attainable distance-to-bun values.)

Small dataset (Test set 1 - Visible)

1 ≤ **K** ≤ 8.

Large dataset (Test set 2 - Hidden)

1 ≤ **K** ≤ 100.

Sample

Input	Output
`3`	`Case #1: 2`

```
5           Case #2: 0
0 2 1 1 2   Case #3: 10
1
0
6
2 2 2 2 2 2
```

Sample Case #1 is the one illustrated in the problem statement.

In Sample Case #2, there is only one ingredient in the burger; that is not much of a burger, but our model has to be able to handle this base case! There is no confusion over how to place the one ingredient, and the error is 0.

In Sample Case #3, there are six ingredients, but all of them have an optimal distance-to-bun of 2. Any way of placing them is equivalent, and the error is $2^2 + 1^2 + 0^2 + 0^2 + 1^2 + 2^2 = 10$.

CEO Search (8pts, 13pts)

Problem

The CEO of Code Jam has just retired to spend more time with *The Art of Computer Programming*, so we need your help finding a new one!

Every Code Jam employee has an *experience level* that is a nonnegative integer. When we hire our new CEO, we must organize the Code Jam team as follows:

- Every employee other than the CEO must have a single *direct manager* who is another employee with an experience level greater than that employee's own experience level. (The CEO cannot have a direct manager.)
- An employee (including the CEO) with experience level E can be a direct manager for between 0 and E other employees, inclusive. Note that if employee A is the direct manager of employee B, and B is the direct manager of C, A is *not* also a direct manager of C.
- Because of office politics, the new CEO cannot be one of the existing employees, no other new employees can be added, and no existing employees can be removed.

Of course, hiring a more experienced CEO is more expensive! What is the minimum possible experience level for the new CEO such that the Code Jam team can be organized according to the rules above?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each begins with one line with an integer **L**: the number of different experience levels present among the existing employees. Then, **L** lines follow; the i-th of these contains two integers $\mathbf{N_i}$ and $\mathbf{E_i}$, and indicates that there are $\mathbf{N_i}$ existing employees that have the experience level $\mathbf{E_i}$.

Output

For each test case, output one line containing `Case #x: y`, where `x` is the test case number (starting from 1) and `y` is the minimum possible experience level for the new CEO, as described above.

Limits

$1 \le \mathbf{T} \le 100$.
Time limit: 20 seconds per test set.
Memory limit: 1GB.
For all $i < j$, $\mathbf{E_i} < \mathbf{E_j}$.

Small dataset (Test set 1 - Visible)

$1 \le \mathbf{L} \le 10$.
$1 \le \mathbf{N_i} \le 10$.
The sum of all $\mathbf{N_i} \le 10$. $0 \le \mathbf{E_i} \le 10$.

Large dataset (Test set 2 - Hidden)

$1 \le \mathbf{L} \le 1000$.
$1 \le \mathbf{N_i} \le 10^{12}$.
$0 \le \mathbf{E_i} \le 1000$.

Sample

Input Output

```
2
3
2 0   Case #1: 4
2 2   Case #2: 5
1 3
1
5 0
```

In Sample Case #1, there are five existing employees: one with an experience level of 3, two with an experience level of 2, and two with an experience level of 0. We can hire a new CEO with an experience level of 4; then, for example, we can have the new CEO directly manage the level 3 and one level 0, and have the level 3 directly manage the two level 2s and the other level 0. (Other valid arrangements are possible.) Moreover, we know that the new CEO must be at least level 4, or else there would be nobody who could directly manage the existing level 3. So 4 is both an upper and lower bound, and must be the correct answer.

In Sample Case #2, all five of the existing employees have an experience level of 0 and cannot directly manage other employees. The new CEO must personally directly manage all of them, which requires an experience level of at least 5.

Centrists (9pts, 14pts)

Problem

Exactly three candidates are running for office. Each candidate's name is a single string of length **L** that is made up only of uppercase letters of the English alphabet, and no two candidates have the same name.

This area has a law to ensure that candidates will not consistently be at an advantage or disadvantage based on their names. Before each election, an ordering of the English alphabet will be chosen from among all possible orderings, and that ordering will be used to alphabetize the names before listing them on the ballot. (To determine which of two different names comes earlier in the list on the ballot, start by comparing the first letter of each name. If they are different, then the name with the first letter that comes earlier in the chosen ordering is the one that comes earlier in the list on the ballot. If they are the same, move on to comparing the second letter of each name, and so on.)

Each of the three candidates wants to adopt a "middle-of-the-road" image, and so they think that being in the middle of the list of three names on the ballot will be advantageous. For each candidate, determine whether there is at least one ordering of the English alphabet for which their name would be the second of the three names on the ballot. Note that we are considering each candidate independently.

For example, suppose that our candidates are named `BCB`, `CAB`, and `CBC`. Since the letters `D` through `Z` are not used in these names, we will only consider the relative ordering of the letters `A`, `B`, and `C`. If the ordering `A`, `B`, `C` is chosen, for example, then the candidates will be listed in the order `BCB`, `CAB`, `CBC`; this demonstrates that it is possible for `CAB` to be in the middle. If the ordering `A`, `C`, `B` is chosen, then the candidates will be listed in the order `CAB`, `CBC`, `BCB`; this demonstrates that it is possible for `CBC` to be in the middle. However, even under any of the other four possible orderings, it turns out that `BCB` can never be in the middle.

Input

The first line of the input gives the number of test cases, **T**; **T** test cases follow. Each begins with one line containing one integer **L**: the length of each candidate's name. Then, there is one line with three different strings $\mathbf{N_i}$ of uppercase letters of the English alphabet; the i-th of these is the name of the i-th candidate.

Output

For each test case, output one line containing `Case #x: y1 y2 y3`, where `x` is the test case number (starting from 1) and each `yi` is `YES` if it is possible for the i-th candidate to be in the middle, as described in the problem statement, and `NO` otherwise.

Limits

1 ≤ **T** ≤ 100.
Time limit: 20 seconds per test set.
Memory limit: 1GB.
1 ≤ **L** ≤ 100.
$\mathbf{N_i}$ is of length **L**, for all i.
$\mathbf{N_i} \neq \mathbf{N_j}$, for all i ≠ j.

Small dataset (Test set 1 - Visible)

$\mathbf{N_i}$ contains only uppercase letters from the set {`A`, `B`, `C`}, for all i.

Large dataset (Test set 2 - Hidden)

N$_i$ contains only uppercase letters from the English alphabet, for all i.

Sample

```
Input                   Output



3
3
BCB CAB CBC             Case #1: NO YES YES
2                       Case #2: NO YES NO
CC CA AA                Case #3: YES YES YES
6
MEDIAN MEDIAL MEDIAS
```

Note that the last sample case would not appear in the Small dataset.

Sample Case #1 is the one described in the problem statement.

In Sample Case #2, no matter which of the two possible relative orderings of `A` and `C` is chosen, `CA` will be in the middle.

In Sample Case #3, any of the names can end up in the middle, depending on which of `L`, `N`, and `S` comes second in the relative order of those three letters in the ordering.

Tricky Trios (15pts, 27pts)

Problem

The game of Tricky Trios is played using a deck of 3**N** cards consisting of three identical cards labeled 1, three identical cards labeled 2, and so on, up to three identical cards labeled **N**. The cards are shuffled (such that all possible card orderings have an equal probability of appearing), and then dealt out onto a table, face down, so that all the numbers are hidden.

Each round of the game proceeds as follows:

- Choose one of the cards and flip it over to reveal its number.
- Choose a second card and flip it over to reveal its number. If that number is not equal to the revealed number on the first card, the round is over and you may not flip a third card. Otherwise:
- Choose a third card and flip it over to reveal its number. If that number is not equal to the revealed number on the second card, the round is over. Otherwise, you have found a trio, and you can remove all three cards from the game; then the round is over.
- Once the round is over, if there are no more cards remaining, you have won the game. Otherwise, before beginning the next round, you must flip all revealed cards back over to hide their numbers, but you have an amazing memory and you can remember where they are for the rest of the game.

Note that you may choose to flip a card even if you already know its number. Also, even if you know the locations of all of the cards in a trio, you must actually flip all three cards in the trio on the same round in order to remove it.

You would like to win as quickly as possible, so you will use a strategy that minimizes the expected number of rounds needed to end the game. What is that expected number of rounds?

Input

The first line of the input gives the number of test cases, **T**; **T** test cases follow. Each consists of one line with an integer **N**, as described above.

Output

For each test case, output one line containing `Case #x: y`, where `x` is the test case number (starting from 1) and `y` is a rational: the minimal expected number of rounds needed to end the game, as described above. `y` will be considered correct if it is within an absolute or relative error of 10^{-6} of the correct answer. See the FAQ for an explanation of what that means, and what formats of real numbers we accept.

Limits

Time limit: 20 seconds per test set.
Memory limit: 1GB.

Small dataset (Test set 1 - Visible)

1 ≤ **T** ≤ 5.
1 ≤ **N** ≤ 5.

Large dataset (Test set 2 - Hidden)

1 ≤ **T** ≤ 100.
1 ≤ **N** ≤ 100.

Sample

Input Output

```
3
1
2
5
```

```
Case #1: 1.000000
Case #2: 3.400000
Case #3: 9.842024
```

In Sample Case #1, all three cards have the same number, so flipping them over in any order will end the game in one round.

In Sample Case #2:

- If the first two cards we flip over are different, our round ends and we cannot flip over a third card. Then, we can use the next round to flip over two more of the unknown cards.
- If they match, then we already know where the remaining third card is, and then we can use one more round to flip over the remaining trio, taking three rounds total. The chances of this scenario are 3/5 × 1/3 = 1/5.

- Otherwise, our second round ends, but once we have flipped over another unknown card on the third round, we know how to finish building both trios, taking four rounds total. The chances of this scenario are 3/5 × 2/3 = 2/5.
- If the first two cards we flip over are the same, we leave the details as an exercise for the solver.

The answer is 3 × 1/5 + 4 × 2/5 + ... = 17/5.

Ticket Trouble (5pts, 10pts) Problem

A group of **F** friends is attending a conference held at an amphitheater, and they have bought tickets to see a concert there afterwards. The amphitheater is a grid of seats with **S** rows and **S** columns. For each seat, the amphitheater has sold a single ticket (although some of the tickets might not have been sold to this group of friends). Each ticket is normally labeled with a pair of integers giving the row and column numbers of one seat, in that order. For example, a ticket might normally say (2, 1), meaning row 2, column 1, or (2, 2), meaning row 2, column 2.

When the tickets were printed, there was a malfunction, and the two numbers in each pair always came out in sorted (that is, nondecreasing) order! So, for example, a ticket labeled (1, 2) might actually be for the seat in row 1, column 2, or it might actually be for the seat in row 2, column 1. If two friends have tickets labeled (1, 2), then one must actually be for row 1, column 2, and the other must actually be for row 2, column 1.

The friends will consult the box office on the day of the concert to find out what their actual seat numbers are, but for now, it is unclear! Given the printed pairs on the tickets, what is the largest possible number of the friends that could actually be seated all in the same-numbered row of seats? (The friends do *not* necessarily need to be seated in consecutive seats in that row.)

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each begins with one line with two integers **F** and **S**, representing the number of friends and the dimension of the grid of seats. Then, **F** more lines follow. The i-th of those lines has two integers $\mathbf{A_i}$ and $\mathbf{B_i}$, representing the two numbers printed on the i-th friend's ticket.

Output

For each test case, output one line containing `Case #x: y`, where `x` is the test case number (starting from 1) and `y` is largest possible number of the friends that could actually be seated all in the same-numbered row of seats.

Limits

Time limit: 20 seconds per test set.
Memory limit: 1GB.
$\mathbf{F} \le \mathbf{S}^2$.
$1 \le \mathbf{A_i} \le \mathbf{B_i} \le \mathbf{S}$, for all i.
No pair appears more than twice in a test case.
No pair containing the same number twice appears more than once in a test case.

Small dataset (Test set 1 - Visible)

1 ≤ **T** ≤ 50.
2 ≤ **F** ≤ 3.
2 ≤ **S** ≤ 3.

Large dataset (Test set 2 - Hidden)

1 ≤ **T** ≤ 100.
2 ≤ **F** ≤ 100.
2 ≤ **S** ≤ 100.

Sample

Input Output

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

```
Case #1: 1
Case #2: 3
Case #3: 2
```

In sample case #1, one ticket must actually be for row 1, column 2, and the other must actually be for row 2, column 1, even though we do not know which is which. So we know that the friends are not seated in the same row, and the largest number of friends in any row is 1. Also note that the seats have a third row and column, but none of the tickets use the third row or column.

In sample case #2, one of the tickets is definitely for seat 2 in row 2, and it is possible that two of the other tickets could be for seats 1 and 3 in row 2. So there may be as many as 3 friends in the same row.

In sample case #3, either there are two friends in row 1 and one in row 2, or there are two friends in row 2 and one in row 1. In either case, the answer is 2.

Understudies (5pts, 15pts)

Problem

You are a casting director for an upcoming musical. The musical has **N** roles, and for each role, you want to cast two performers: one primary performer and one understudy. A primary performer or understudy trains for only one particular role, and the job of the understudy is to play the role if the primary performer becomes unavailable. At least one of the two performers for each role must be available for the show to succeed.

You have selected 2**N** performers to be in the musical. They are all quite talented, and any of them can be cast as a primary performer or understudy for any of the roles. However, you are worried that some of them may be tempted to run away to join the cast of *Hamiltonian!*, the smash hit musical about quantum mechanics, before your show opens. Luckily, you are an excellent judge of character. You know that the i-th performer has a probability $\mathbf{P_i}$ of becoming unavailable. (These probabilities are all independent of each other, and a given performer has their probability regardless of their assigned role or whether they are a primary performer or understudy.)

You wish to assign one primary performer and one understudy for each role in a way that maximizes the probability that the show will succeed. That is, you want to minimize the probability that there will be at least one role for which the primary performer and understudy both become unavailable.

If you make optimal casting choices, what is the probability that your show will succeed?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow; each consists of two lines. The first line contains a single integer **N**: the number of roles. The second line contains 2**N** rational numbers $\mathbf{P_i}$; the i-th of these gives the probability that the i-th performer will become unavailable for your show. All of these probabilities are specified to exactly four decimal places of precision.

Output

For each test case, output one line containing `Case #x: y`, where `x` is the test case number (starting from 1) and `y` is the probability that your show will succeed. `y` will be considered correct if it is within an absolute or relative error of 10^{-6} of the correct answer. See the FAQ for an explanation of what that means, and what formats of real numbers we accept.

Limits

$1 \leq \mathbf{T} \leq 100$.
Time limit: 20 seconds per test set.
Memory limit: 1GB.
$0.0000 \leq \mathbf{P_i} \leq 1.0000$, for all i.

Small dataset (Test set 1 - Visible)

$1 \leq \mathbf{N} \leq 4$.

Large dataset (Test set 2 - Hidden)

$1 \leq \mathbf{N} \leq 40$.

Sample

Input

```
3
2
0.2500 0.5000 0.5000 0.2500
3
```

```
0.0000 0.0000 0.0000 0.0009 0.0013 0.1776
1
1.0000 0.1234
```

Output

```
Case #1: 0.765625
Case #2: 1.000000
Case #3: 0.876600
```

In sample case #1, one optimal casting choice is to make the two 0.5000 performers leads for the two roles, and the two 0.2500 performers understudies. For a given role, the probability that both performers will become unavailable is 0.5 × 0.25 = 0.125. So the probability that a role will be filled by at least one of its actors is 1 - 0.125 = 0.875. The probability that both roles will be filled (and thus that the show will succeed) is 0.875 × 0.875 = 0.765625.

If we instead cast the two 0.5000 performers for one role and the two 0.2500 performers for the other role, the probability of success would be (1 - 0.50 × 0.50) × (1 - 0.25 × 0.25) = 0.703125, which is lower.

In sample case #2, the show will succeed for sure as long as you cast exactly one of the 0.0000 performers (who will never become unavailable) in each role.

In sample case #3, the 1.0000 performer will always become unavailable, so the probability of success is equal to 1 minus the probability that the other performer will become unavailable.

Word Search (10pts, 15pts)

Problem

In honor of Google I/O 2017, we would like to make an I/O-themed word search grid. This will be a rectangular grid in which every cell contains one of the three characters `I`, `/`, or `O`. The people solving our word search will look for all instances of the string `I/O` that appear contiguously forwards or backwards in a row, column, or diagonal. For example, the following grid contains eight instances of `I/O`, representing all eight possible directions in which the string can appear:

```
OOOOO
O///O
O/I/O
O///O
OOOOO
```

To control the difficulty level of our word search, we would like the string to appear *exactly* **N** times in the grid. Moreover, we do not want the grid to be too large; it cannot have more than **D** rows or more than **D** columns.

Can you help us design a grid that meets these specifications?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case consists of one line with two integers **D** and **N**, as described above.

Output

For each test case, first output one line containing `Case #x:`. Then output R lines of exactly C characters each, representing the rectangular grid. Each of those characters must be either `I`, `/`, or `O`. You may choose any values of R and C as long as both are at least 1 and neither exceeds **D**. Your grid must contain *exactly* **N** instances of the string `I/O`, per the rules described in the statement.

If there are multiple valid answers, you may output any of them.

Limits

Time limit: 20 seconds per test set.
Memory limit: 1GB.
0 ≤ **N** ≤ 287.
It is guaranteed that at least one valid grid exists for each test case.

Small dataset (Test set 1 - Visible)

1 ≤ **T** ≤ 25.
D = 50.

Large dataset (Test set 2 - Hidden)

1 ≤ **T** ≤ 100.
D = 15.

Sample

Input Output

```
     Case #1:
     O
     /
     I
4    Case #2:
50 1 IO
50 0 Case #3:
50 3 IIIOOO
50 8 /I/O/O
     IIIOOO
     Case #4:
     OOOOO
     O///O
     O/I/O
     O///O
```

```
00000
```

The sample output displays one set of answers to the sample cases. Other answers may be possible. Note that these cases would only appear in the Small dataset.

Where Ya Gonna Call? (15pts, 25pts)

Problem

Gooli is a huge company that owns **B** buildings in a hilly area. The buildings are numbered from 1 to **B**.

Last year, they built a set of slides between buildings that are now the favorite form of transportation between buildings. Slides have been upgraded with suction technology to make them two-way, so a slide between two buildings can be used to travel between those buildings in either direction. Some slides were built with turns, so their lengths do not necessarily follow common sense; for instance, they do not necessarily comply with the triangle inequality. Also, there is at most one slide between any pair of buildings.

Gooli is going to choose a location to install a special super secure phone for the CEO to talk to other important people. They want to minimize the distance by slide from any building to the meeting location, so as to minimize the time that it would take the CEO to reach it from any building. Gooli does not have any more carbon kilotubes to build more slides, and the CEO refuses any other type of transportation, so Gooli's communication security team needs to find the best location that is reachable using only already existing slides. The location could be in a building or a point somewhere within a slide.

When traveling using the slides, the CEO may use a slide, arrive at a building, then use a slide that starts there, arrive at another building, and so on, until she arrives at the desired location. Slides used from end to end contribute their full length to the total distance. If the CEO enters a slide and stops inside it because she found the phone, on the other hand, only the used part of the slide contributes to the total distance. When measuring distance, only the slide distance is important. Distance traveled within buildings to connect to a new slide or reach the phone is considered to be zero.

Given the buildings and slides in existence, can you find any optimal location for the super secure phone and return the distance from a farthest building to it? Note that the distance is the same for any optimal location.

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow. Each test case starts with one line with a single integer **B**, the number of buildings on Gooli's campus. Then, **B** - 1 lines follow. For i = 2, 3, ..., **B**, the (i-1)-th of these lines contains (i-1) integers $\mathbf{D}_{i1}$, $\mathbf{D}_{i2}$, ..., $\mathbf{D}_{i(i-1)}$. $\mathbf{D}_{ij}$ is -1 if there is no slide between the i-th building and the j-th building, or the length of that slide otherwise. All buildings are reachable from any other building using only slides, possibly passing through intermediate buildings.

Output

For each test case, output one line containing `Case #x: y`, where `x` is the test case number (starting from 1) and `y` is the distance from an optimal location for the phone to a building farthest from it. `y` will be considered correct if it is within an absolute or relative error of 10^{-6} of the correct answer. See the FAQ for an explanation of what that means, and what formats of real numbers we accept.

Limits

$1 \le \mathbf{T} \le 100$.
Time limit: 100 seconds per test set.
Memory limit: 1GB.
$2 \le \mathbf{B} \le 50$.
All buildings are reachable from any other building using only slides, possibly passing through intermediate buildings.
$\mathbf{D}_{ij} \ne 0$, for all i, j.

Small dataset (Test set 1 - Visible)

$-1 \le \mathbf{D}_{ij} \le 2$, for all i, j.

Large dataset (Test set 2 - Hidden)

$-1 \le \mathbf{D}_{ij} \le 10^9$, for all i, j.

Sample

```
Input     Output


4
3
-1
1 2
3
1         Case #1: 1.500000
1 1       Case #2: 1.000000
3         Case #3: 2.500000
4         Case #4: 8.500000
2 3
4
9
10 7
7 -1 5
```

Note that the last two cases would not appear in the Small dataset.

In Case #1, all buildings are in a line. The only optimal location is of course the middle point of the line, as any other location would make one of the buildings at the end of the line be farther away.

Case #2 depicts an equilateral triangle. Any of the three buildings would be an optimal location for the phone.

Case #3 is also a triangle, but with sides of different lengths. If we pick any building, the farthest building would be at distance at least 3 from it. On the other hand, if we choose a location inside the slide of size 3, at distance 0.5 from building 3, the distance to a farthest building is improved to 2.5.

In Case #4, the optimal location is inside the slide of length 10 between buildings 1 and 3, at distance 1.5 from building 3 and distance 8.5 from building 1.

Cody's Jams (10pts, 10pts)

Problem

Cody, the owner of the legendary Cody's Jams store, is planning a huge jam sale. To make things simple, he has decided to sell every item in his store at a 25% discount — that is, each item's sale price is exactly 75% of its regular price. Since all of his regular prices happened to be integers divisible by four, his sale prices are conveniently also all integers.

To prepare for the sale, he placed an order to print new labels for all of his items at their sale prices. He also placed an order to print new labels for all of his items at their regular prices, to use once the sale is over.

Cody just came back from picking up his order. Unfortunately, the printer gave him both orders in one combined stack, sorted by price. Both the sale price and the regular price label for each item are present somewhere in the stack. However, both types of labels look the same, and since he does not remember the price of every item, he is not sure which labels are the sale price labels. Can you figure that out?

For instance, if the regular prices were 20, 80, and 100, the sale prices would be 15, 60, and 75, and the printer's stack would consist of the labels 15, 20, 60, 75, 80, and 100.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case consists of two lines. The first line contains a single integer **N**, the number of items in Cody's store. The second line contains 2**N** integers $\mathbf{P_1}$, $\mathbf{P_2}$, ..., $\mathbf{P_{2N}}$ in non-decreasing order by the price printed on each label given by the printer.

Output

For each test case, output one line containing `Case #x: y`, where `x` is the test case number (starting from 1) and `y` is a list of **N** integers: the labels containing sale prices, in non-decreasing order.

Limits

$1 \le \mathbf{T} \le 100$.
Time limit: 20 seconds per test set.
Memory limit: 1GB.
$1 \le \mathbf{P_i} \le 10^9$, for all i.

$\mathbf{P_i} \le \mathbf{P_{i+1}}$, for all i. (The prices are in non-decreasing order.)
It is guaranteed that a unique solution exists.

Small dataset (Test set 1 - Visible)

1 ≤ **N** ≤ 4.

Large dataset (Test set 2 - Hidden)

1 ≤ **N** ≤ 100.

Sample

Input	Output

`2`
`3`
`15 20 60 75 80 100`
`4`
`9 9 12 12 12 15 16 20`
	`Case #1: 15 60 75`
	`Case #2: 9 9 12 15`

Case #1 is the one described in the problem statement.

Notice in Case #2 that it is possible for multiple items to have the same price, and for an item to have a regular price that equals the sale price of another item.

Dance Around The Clock (10pts, 15pts)

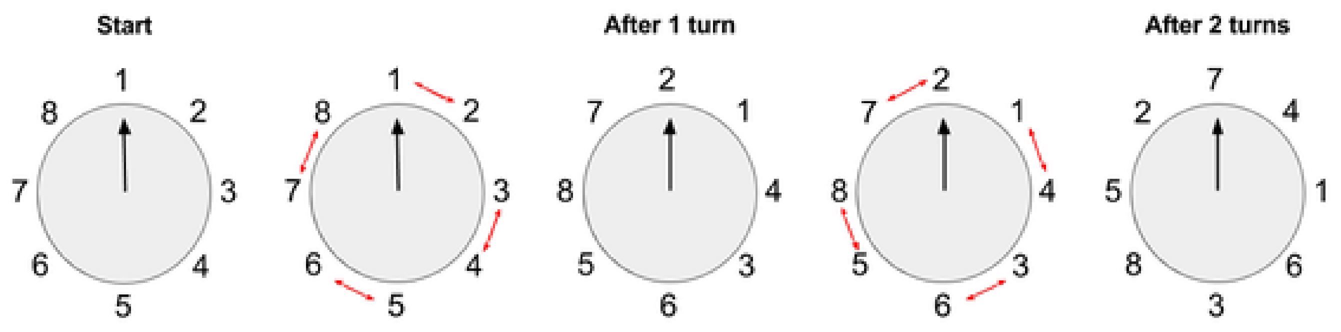
Problem

The owner of a prestigious ballroom has painted a beautiful circular clock on the dance floor, and a group of **D** dancers numbered 1 through **D** are about to literally "dance around the clock". They are standing in a circle, with dancer 1 at the 12:00 position of the circle and the other dancers going clockwise around the circle in increasing numerical order. The number of dancers is even.

The dance will go on for **N** turns. On the i-th turn (counting starting from 1), the following will happen:

- If i is odd, then the dancer currently at the 12:00 position will swap positions with the next dancer in clockwise order. Then, going past those two, the next pair of dancers in clockwise order will swap positions, and so on, all the way around the ring clockwise, until all dancers have participated in exactly one swap.
- If i is even, then the dancer currently at the 12:00 position will swap positions with the next dancer in counterclockwise order. Then, going past those two, the next pair of dancers in counterclockwise order will swap positions, and so on, all the way around the ring counterclockwise, until all dancers have participated in a swap.

For example, this diagram shows the initial state and two turns of a dance with eight people.



Which two dancers will be next to dancer number **K** when the dance is over?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each consists of one line with three integers **D**, **K**, and **N**: the total number of dancers, the number of one of the dancers, and the number of turns the dance will go on for.

Output

For each test case, output one line containing `Case #x: y z`, where:

- x is the test case number (starting from 1).
- y is the number of the dancer who will be standing to dancer number **K**'s left (that is, one step away in clockwise order) when the dance is over.
- z is the number of the dancer who will be standing to dancer number **K**'s right (that is, one step away in counterclockwise order) when the dance is over.

Limits

1 ≤ **T** ≤ 100.
Time limit: 20 seconds per test set.
Memory limit: 1GB.
D is even.
1 ≤ **K** ≤ **D**.

Small dataset (Test set 1 - Visible)

4 ≤ **D** ≤ 10.
1 ≤ **N** ≤ 10.

Large dataset (Test set 2 - Hidden)

4 ≤ **D** ≤ 10^8.
1 ≤ **N** ≤ 10^8.

Sample

Input Output

```
3       Case #1: 6 4
8 3 1 Case #2: 1 7
8 4 2 Case #3: 2 4
4 1 8
```

For Cases #1 and #2, refer to the illustration above. In Case #1, after 1 turn, dancer 6 is to dancer 3's left, and dancer 4 is to dancer 3's right. In Case #2, after 2 turns, dancer 1 is to dancer 4's left, and dancer 7 is to dancer 4's right. Remember that you're looking from the dancer's perspective; it may help to think in terms of clockwise and counterclockwise instead of left and right.

In Case #3, after eight turns, the arrangement looks the same as the initial arrangement, with dancer 2 to dancer 1's left, and dancer 4 to dancer 1's right

Polynesiaglot (5pts, 10pts, 10pts)

Problem

Ursula is a big fan of constructing artificial languages. Today, she is starting to work on a language inspired by real Polynesian languages. The only rules she has established are:

- All words consist of letters. Letters are either consonants or vowels.
- Any consonant in a word must be immediately followed by a vowel.

For example, in a language in which *a* is the only vowel and *h* is the only consonant, *a*, *aa*, *aha*, *aaha*, and *haha* are valid words, whereas *h*, *ahh*, *ahah*, and *ahha* are not. Note that the rule about consonants disallows ending a word in a consonant as well as following a consonant with another consonant.

If Ursula's new language has **C** different consonants and **V** different vowels available to use, then how many different valid words of length **L** are there in her language? Since the output can be a really big number, we only ask you to output the remainder of dividing the result by the prime 10^9+7 (1000000007).

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each consists of one line with three integers **C**, **V**, and **L**.

Output

For each test case, output one line containing `Case #x: y`, where `x` is the test case number (starting from 1) and `y` is the number of different valid words of length **L** in the language, modulo the prime 10^9+7 (1000000007).

Limits

Time limit: 20 seconds per test set.
Memory limit: 1GB.

Small dataset 1 (Test set 1 - Visible)

T = 15.
C = 1.
V = 1.
1 ≤ **L** ≤ 15.

Small dataset 2 (Test set 2 - Visible)

1 ≤ **T** ≤ 100.
1 ≤ **C** ≤ 50.
1 ≤ **V** ≤ 50.
1 ≤ **L** ≤ 15.

Large dataset (Test set 3 - Hidden)

1 ≤ **T** ≤ 100.
1 ≤ **C** ≤ 50.
1 ≤ **V** ≤ 50.
1 ≤ **L** ≤ 500.

Sample

Input	Output
`2` `1 1 4` `1 2 2`	`Case #1: 5` `Case #2: 6`

In Case #1, suppose that the only vowel is *a* and the only consonant is *h*. Then the possible valid words of length 4 are: *aaaa*, *aaha*, *ahaa*, *haaa*, *haha*.

In Case #2 (which would not appear in Small dataset 1), suppose that the two vowels are *a* and *e* and the only consonant is *h*. Then the possible valid words of length 2 are: *aa*, *ae*, *ea*, *ee*, *ha*, *he*.

Password Security (10pts, 20pts)

Problem

You just bought your young nephew Andrey a complete set of 26 English wooden alphabet letters from A to Z. Because the letters come in a long, linear package, they appear to spell out a 26-letter message.

You use **N** different passwords to log into your various online accounts, and you are concerned that this message might coincidentally include one or more of them. Can you find any arrangement of the 26 letters, such that no password appears in the message as a continuous substring?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each consists of one line with an integer **N**, and then another line with **N** different strings of uppercase English letters $\mathbf{P_1}$, $\mathbf{P_2}$, ..., $\mathbf{P_N}$, which are the passwords.

Output

For each test case, output one line containing `Case #x: y`, where `x` is the test case number (starting from 1) and `y` is a permutation of the entire uppercase English alphabet that contains no password as a continuous substring, or the word `IMPOSSIBLE` if there is no such permutation.

Limits

1 ≤ **T** ≤ 100.
Time limit: 20 seconds per test set.
Memory limit: 1GB.
1 ≤ length of $\mathbf{P_i}$ ≤ 26, for all i. (Each password is between 1 and 26 letters long.)
$\mathbf{P_i} \neq \mathbf{P_j}$ for all i ≠ j. (All passwords are different.)

Small dataset 1 (Test set 1 - Visible)

N = 1.

Small dataset 2 (Test set 2 - Visible)

1 ≤ **N** ≤ 50.

Sample

Input

```
7
1
ABCDEFGHIJKLMNOPQRSTUVWXYZ
1
X
1
QQ
5
XYZ GCJ OMG LMAO JK
3
AB YZ NM
6
C PYTHON GO PERL RUBY JS
2
SUBDERMATOGLYPHIC UNCOPYRIGHTABLES
```

Output

```
Case #1: QWERTYUIOPASDFGHJKLZXCVBNM
Case #2: IMPOSSIBLE
Case #3: ABCDEFGHIJKLMNOPQRSTUVWXYZ
Case #4: ABCDEFGHIKLMNOPQRSTUVWXYJZ
Case #5: ZYXWVUTSRQPOMNLKJIHGFEDCBA
Case #6: IMPOSSIBLE
Case #7: THEQUICKBROWNFXJMPSVLAZYDG
```

For each of the non-`IMPOSSIBLE` cases, the sample output shows only one possible answer. There are many valid answers for these inputs.

Note that only sample cases #1, #2, and #3 would appear in Small dataset 1. Any of the sample cases could appear in Small dataset 2.