Problem A. Counting Sheep

Small input 7 points	Solve A-small
Large input 8 points	Solve A-large

Problem

Bleatrix Trotter the sheep has devised a strategy that helps her fall asleep faster. First, she picks a number **N**. Then she starts naming **N**, 2 × **N**, 3 × **N**, and so on. Whenever she names a number, she thinks about all of the digits in that number. She keeps track of which digits (0, 1, 2, 3, 4, 5, 6, 7, 8, and 9) she has seen at least once so far as part of any number she has named. Once she has seen each of the ten digits at least once, she will fall asleep.

Bleatrix must start with **N** and must always name (i + 1) × **N** directly after i × **N**. For example, suppose that Bleatrix picks **N** = 1692. She would count as follows:

- **N** = 1692. Now she has seen the digits 1, 2, 6, and 9.
- 2**N** = 3384. Now she has seen the digits 1, 2, 3, 4, 6, 8, and 9.
- 3**N** = 5076. Now she has seen all ten digits, and falls asleep.

What is the last number that she will name before falling asleep? If she will count forever, print `INSOMNIA` instead.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each consists of one line with a single integer **N**, the number Bleatrix has chosen.

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 last number that Bleatrix will name before falling asleep, according to the rules described in the statement.

Limits

1 ≤ **T** ≤ 100.

Small dataset

0 ≤ **N** ≤ 200.

Large dataset

0 ≤ **N** ≤ 10^6.

Sample

```
Input     Output

5         Case #1: INSOMNIA
0         Case #2: 10
1         Case #3: 90
2         Case #4: 110
11        Case #5: 5076
1692
```

In Case #1, since 2 × 0 = 0, 3 × 0 = 0, and so on, Bleatrix will never see any digit other than 0, and so she will count forever and never fall asleep. Poor sheep!

In Case #2, Bleatrix will name 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. The 0 will be the last digit needed, and so she will fall asleep after 10.

In Case #3, Bleatrix will name 2, 4, 6... and so on. She will not see the digit 9 in any number until 90, at which point she will fall asleep. By that point, she will have already seen the digits 0, 1, 2, 3, 4, 5, 6, 7, and 8, which will have appeared for the first time in the numbers 10, 10, 2, 30, 4, 50, 6, 70, and 8, respectively.

In Case #4, Bleatrix will name 11, 22, 33, 44, 55, 66, 77, 88, 99, 110 and then fall asleep.

Case #5 is the one described in the problem statement. Note that it would only show up in the Large dataset, and not in the Small dataset.

Problem B. Revenge of the Pancakes

Small input 10 points	Solve B-small
Large input 10 points	Solve B-large

Problem

The Infinite House of Pancakes has just introduced a new kind of pancake! It has a happy face made of chocolate chips on one side (the "happy side"), and nothing on the other side (the "blank side").

You are the head waiter on duty, and the kitchen has just given you a stack of pancakes to serve to a customer. Like any good pancake server, you have X-ray pancake vision, and you can see whether each pancake in the stack has the happy side up or the blank side up. You think the customer will be happiest if every pancake is happy side up when you serve them.

You know the following maneuver: carefully lift up some number of pancakes (possibly all of them) from the top of the stack, flip that entire group over, and then put the group back down on top of any pancakes that you did not lift up. When flipping a group of pancakes, you flip the entire group in one motion; you do *not* individually flip each pancake. Formally: if we number the pancakes 1, 2, ..., *N* from top to bottom, you choose the top *i* pancakes to flip. Then, after the flip, the stack is *i*, *i*-1, ..., 2, 1, *i*+1, *i*+2, ..., *N*. Pancakes 1, 2, ..., *i* now have the opposite side up, whereas pancakes *i*+1, *i*+2, ..., *N* have the same side up that they had up before.

For example, let's denote the happy side as + and the blank side as -. Suppose that the stack, starting from the top, is `--+-`. One valid way to execute the maneuver would be to pick up the top three, flip the entire group, and put them back down on the remaining fourth pancake (which would stay where it is and remain unchanged). The new state of the stack would then be `-++-`. The other valid ways would be to pick up and flip the top one, the top two, or all four. It would not be valid to choose and flip the middle two or the bottom one, for example; you can only take some number off the top.

You will not serve the customer until every pancake is happy side up, but you don't want the pancakes to get cold, so you have to act fast! What is the smallest number of times you will need to execute the maneuver to get all the pancakes happy side up, if you make optimal choices?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each consists of one line with a string **S**, each character of which is either + (which represents a pancake that is initially happy side up) or - (which represents a pancake that is initially blank side up). The string, when read left to right, represents the stack when viewed from top to bottom.

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 number of times you will need to execute the maneuver to get all the pancakes happy side up.

Limits

1 ≤ **T** ≤ 100.
Every character in **S** is either + or -.

Small dataset

1 ≤ length of **S** ≤ 10.

Large dataset

1 ≤ length of **S** ≤ 100.

Sample

Input	Output
5 - -+ +- +++	Case #1: 1 Case #2: 1 Case #3: 2 Case #4: 0 Case #5: 3

```
--+-
```

In Case #1, you only need to execute the maneuver once, flipping the first (and only) pancake.

In Case #2, you only need to execute the maneuver once, flipping only the first pancake.

In Case #3, you must execute the maneuver twice. One optimal solution is to flip only the first pancake, changing the stack to `--`, and then flip both pancakes, changing the stack to `++`. Notice that you cannot just flip the bottom pancake individually to get a one-move solution; every time you execute the maneuver, you must select a stack starting from the top.

In Case #4, all of the pancakes are already happy side up, so there is no need to do anything.

In Case #5, one valid solution is to first flip the entire stack of pancakes to get `+-++`, then flip the top pancake to get `--++`, then finally flip the top two pancakes to get `++++`.

Problem C. Coin Jam

Small input 10 points	Solve C-small
Large input 20 points	Solve C-large

Problem

A *jamcoin* is a string of **N** ≥ 2 digits with the following properties:

- Every digit is either `0` or 1.
- The first digit is 1 and the last digit is 1.
- If you interpret the string in any base between 2 and 10, inclusive, the resulting number is not prime.

Not every string of `0`s and 1s is a jamcoin. For example, `101` is not a jamcoin; its interpretation in base 2 is 5, which is prime. But the string `1001` is a jamcoin: in bases 2 through 10, its interpretation is 9, 28, 65, 126, 217, 344, 513, 730, and 1001, respectively, and none of those is prime.

We hear that there may be communities that use jamcoins as a form of currency. When sending someone a jamcoin, it is polite to prove that the jamcoin is legitimate by including a nontrivial divisor of that jamcoin's interpretation in each base from 2 to 10. (A nontrivial divisor for a positive integer *K* is some positive integer other than 1 or *K* that evenly divides *K*.) For convenience, these divisors must be expressed in base 10.

For example, for the jamcoin `1001` mentioned above, a possible set of nontrivial divisors for the base 2 through 10 interpretations of the jamcoin would be: 3, 7, 5, 6, 31, 8, 27, 5, and 77, respectively.

Can you produce **J** different jamcoins of length **N**, along with proof that they are legitimate?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow; each consists of one line with two integers **N** and **J**.

Output

For each test case, output **J**+1 lines. The first line must consist of only `Case #x:`, where x is the test case number (starting from 1). Each of the last **J** lines must consist of a jamcoin of length **N** followed by nine integers. The i-th of those nine integers (counting starting from 1) must be a nontrivial divisor of the jamcoin when the jamcoin is interpreted in base i+1.

All of these jamcoins must be different. You cannot submit the same jamcoin in two different lines, even if you use a different set of divisors each time.

Limits

T = 1. (There will be only one test case.)
It is guaranteed that at least **J** distinct jamcoins of length **N** exist.

Small dataset

N = 16.
J = 50.

Large dataset

N = 32.
J = 500.

Note that, unusually for a Code Jam problem, you already know the exact contents of each input file. For example, the Small dataset's input file will always be exactly these two lines:

```
1
16 50
```

So, you can consider doing some computation before actually downloading an input file and starting the clock.

Sample

Input	Output

```
1       Case #1:
6 3     100011 5 13 147 31 43 1121 73 77 629
        111111 21 26 105 1302 217 1032 513 13286 10101
        111001 3 88 5 1938 7 208 3 20 11
```

In this sample case, we have used very small values of **N** and **J** for ease of explanation. Note that this sample case would not appear in either the Small or Large datasets.

This is only one of multiple valid solutions. Other sets of jamcoins could have been used, and there are many other possible sets of nontrivial base 10 divisors. Some notes:

- `110111` could not have been included in the output because, for example, it is 337 if interpreted in base 3 (1*243 + 1*81 + 0*27 + 1*9 + 1*3 + 1*1), and 337 is prime.
- `010101` could not have been included in the output even though `10101` is a jamcoin, because jamcoins begin with 1.
- `101010` could not have been included in the output, because jamcoins end with 1.
- `110011` is another jamcoin that could have also been used in the output, but could not have been added to the end of this output, since the output must contain exactly **J** examples.
- For the first jamcoin in the sample output, the first number after `100011` could not have been either 1 or 35, because those are trivial divisors of 35 (`100011` in base 2).

Problem D. Fractiles

Small input 10 points	Solve D-small
Large input 25 points	Solve D-large

Problem

Long ago, the Fractal civilization created artwork consisting of linear rows of tiles. They had two types of tile that they could use: gold (G) and lead (L).

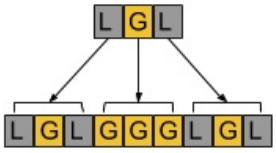
Each piece of Fractal artwork is based on two parameters: an original sequence of **K** tiles, and a complexity **C**. For a given original sequence, the artwork with complexity 1 is just that original sequence, and the artwork with complexity $X+1$ consists of the artwork with complexity X, transformed as follows:

- replace each L tile in the complexity X artwork with another copy of the original sequence
- replace each G tile in the complexity X artwork with **K** G tiles

For example, for an original sequence of LGL, the pieces of artwork with complexity 1 through 3 are:

- **C** = 1: LGL (which is just the original sequence)
- **C** = 2: LGLGGGLGL
- **C** = 3: LGLGGGLGLGGGGGGGGLGLGGGLGL

Here's an illustration of how the artwork with complexity 2 is generated from the artwork with complexity 1:



You have just discovered a piece of Fractal artwork, but the tiles are too dirty for you to tell what they are made of. Because you are an expert archaeologist familiar with the local Fractal culture, you know the values of **K** and **C** for the artwork, but you do not know the original sequence. Since gold is exciting, you would like to know whether there is at least one G tile in the artwork. Your budget allows you to hire **S** graduate students, each of whom can clean one tile of your choice (out of the $\mathbf{K}^{\mathbf{C}}$ tiles in the artwork) to see whether the tile is G or L.

Is it possible for you to choose a set of no more than **S** specific tiles to clean, such that *no matter what* the original pattern was, you will be able to know for sure whether at least one G tile is present in the artwork? If so, which tiles should you clean?

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: **K**, **C**, and **S**.

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 `IMPOSSIBLE` if no set of tiles will answer your question, or a list of between 1 and **S** positive integers, which are the positions of the tiles that will answer your question. The tile positions are numbered from 1 for the leftmost tile to $\mathbf{K}^{\mathbf{C}}$ for the rightmost tile. Your chosen positions may be in any order, but they must all be different.

If there are multiple valid sets of tiles, you may output any of them. Remember that once you submit a Small and it is accepted, you will not be able to download and submit another Small input. See the FAQ for a more thorough explanation. **This reminder won't appear in problems in later rounds.**

Limits

$1 \le \mathbf{T} \le 100$.
$1 \le \mathbf{K} \le 100$.
$1 \le \mathbf{C} \le 100$.
$\mathbf{K}^{\mathbf{C}} \le 10^{18}$.

Small dataset

S = **K**.

Large dataset

1 ≤ **S** ≤ **K**.

Sample

```
Input     Output

5         Case #1: 2
2 3 2     Case #2: 1
1 1 1     Case #3: IMPOSSIBLE
2 1 1     Case #4: 1 2
2 1 2     Case #5: 2 6
3 2 3
```

Note: for some of these sample cases, other valid solutions exist.

In sample case #1, there are four possible original sequences: GG, GL, LG, and LL. They would produce the following artwork, respectively:

- Original sequence GG: GGGGGGGG
- Original sequence GL: GGGGGGGL
- Original sequence LG: LGGGGGGG
- Original sequence LL: LLLLLLLL

One valid solution is to just look at tile #2. If tile #2 turns out to be G, then you will know for sure the artwork contains at least one G. (You will not know whether the original sequence is GG, GL, or LG, but that doesn't matter.) If tile #2 turns out to be L, then you will know that the original sequence must be LL, so there are no Gs in the artwork. So 2 is a valid solution.

On the other hand, it would not be valid to just look at tile #1. If it turns out to be L, you will only know that the original sequence could have been either LG or LL. If the original sequence is LG, there is at least one G in the artwork, but if the original sequence is LL, there are no Gs. So 1 would not be a valid solution.

Note that 1 2 is also a valid solution, because tile #2 already provides all the information you need. 1 2 3 is not a valid solution, because it uses too many tiles.

In sample case #2, the artwork must consist of only one tile: either G or L. Looking at that tile will trivially tell you whether or not the artwork has a G in it.

In sample case #3, which would not appear in the Small dataset, the artwork must be either GG, GL, LG, or LL. You can only look at one tile, and neither of them on its own is enough to answer the question. If you see L for tile #1, you will not know whether the artwork is LG or LL, so you will not know whether any Gs are present. If you see L for tile #2, you will not know whether the artwork is GL or LL, so you will not know whether any Gs are present.

Sample case #4 is like sample case #3, but with access to one more tile. Now you can just look at the entire artwork.

In sample case #5, there are eight possible original sequences, and they would produce the following artwork:

- Original sequence GGG: GGGGGGGGG
- Original sequence GGL: GGGGGGGGL
- Original sequence GLG: GGGGLGGGG
- Original sequence GLL: GGGGLLGLL
- Original sequence LGG: LGGGGGGGG
- Original sequence LGL: LGLGGGLGL
- Original sequence LLG: LLGLLGGGG
- Original sequence LLL: LLLLLLLLL

One valid solution is to look at tiles #2 and #6. If they both turn out to be Ls, the artwork must be all Ls. Otherwise, there must at least one G. Note that 1 2 would not be a valid solution, because even if those tiles both turn out to be Ls, that does not rule out an original sequence of LLG. 6 2 would be a valid solution, since the order of the positions in your solution does not matter.


Powered by
Google Cloud Platform

Problem A. The Last Word

Small input 9 points	Solve A-small
Large input 11 points	Solve A-large

Problem

On the game show *The Last Word*, the host begins a round by showing the contestant a string **S** of uppercase English letters. The contestant has a whiteboard which is initially blank. The host will then present the contestant with the letters of **S**, one by one, in the order in which they appear in **S**. When the host presents the first letter, the contestant writes it on the whiteboard; this counts as the first *word* in the game (even though it is only one letter long). After that, each time the host presents a letter, the contestant must write it at the beginning or the end of the word on the whiteboard before the host moves on to the next letter (or to the end of the game, if there are no more letters).

For example, for **S** = `CAB`, after writing the word `C` on the whiteboard, the contestant could make one of the following four sets of choices:

- put the `A` before `C` to form `AC`, then put the `B` before `AC` to form `BAC`
- put the `A` before `C` to form `AC`, then put the `B` after `AC` to form `ACB`
- put the `A` after `C` to form `CA`, then put the `B` before `CA` to form `BCA`
- put the `A` after `C` to form `CA`, then put the `B` after `CA` to form `CAB`

The word is called the *last word* when the contestant finishes writing all of the letters from **S**, under the given rules. The contestant wins the game if their last word is the last of an alphabetically sorted list of all of the possible last words that could have been produced. For the example above, the winning last word is `CAB` (which happens to be the same as the original word). For a game with **S** = `JAM`, the winning last word is `MJA`.

You are the next contestant on this show, and the host has just showed you the string **S**. What's the winning last word that you should produce?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each consists of one line with a string **S**.

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 winning last word, as described in the statement.

Limits

1 ≤ **T** ≤ 100.

Small dataset

1 ≤ length of **S** ≤ 15.

Large dataset

1 ≤ length of **S** ≤ 1000.

Sample

Input	Output
7 CAB JAM CODE ABAAB CABCBBABC ABCABCABC ZXCASDQWE	Case #1: CAB Case #2: MJA Case #3: OCDE Case #4: BBAAA Case #5: CCCABBBAB Case #6: CCCBAABAB Case #7: ZXCASDQWE

Problem B. Rank and File

Small input 14 points	Solve B-small
Large input 21 points	Solve B-large

Problem

When Sergeant Argus's army assembles for drilling, they stand in the shape of an **N** by **N** square grid, with exactly one soldier in each cell. Each soldier has a certain height.

Argus believes that it is important to keep an eye on all of his soldiers at all times. Since he likes to look at the grid from the upper left, he requires that:

- Within every row of the grid, the soldiers' heights must be in strictly increasing order, from left to right.
- Within every column of the grid, the soldiers' heights must be in strictly increasing order, from top to bottom.

Although no two soldiers in the same row or column may have the same height, it is possible for multiple soldiers in the grid to have the same height.

Since soldiers sometimes train separately with their row or their column, Argus has asked you to make a report consisting of 2*__N__ lists of the soldiers' heights: one representing each row (in left-to-right order) and column (in top-to-bottom order). As you surveyed the soldiers, you only had small pieces of paper to write on, so you wrote each list on a separate piece of paper. However, on your way back to your office, you were startled by a loud bugle blast and you dropped all of the pieces of paper, and the wind blew one away before you could recover it! The other pieces of paper are now in no particular order, and you can't even remember which lists represent rows and which represent columns, since you didn't write that down.

You know that Argus will make you do hundreds of push-ups if you give him an incomplete report. Can you figure out what the missing list is?

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**, followed by 2*__N__-1 lines of **N** integers each, representing the lists you have, as described in the statement. It is guaranteed that these lists represent all but one of the rows and columns from a valid grid, as described in the statement.

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 in strictly increasing order, representing the missing list.

Limits

1 ≤ **T** ≤ 50.
1 ≤ all heights ≤ 2500.
The integers on each line will be in strictly increasing order.
It is guaranteed that a unique valid answer exists.

Small dataset

2 ≤ **N** ≤ 10.

Large dataset

2 ≤ **N** ≤ 50.

Sample

Input	Output
1 3 1 2 3 2 3 5 3 5 6 2 3 4 1 2 3	Case #1: 3 4 6

In the sample case, the arrangement must be either this:

```
1 2 3
2 3 4
3 5 6
```

or this:

```
1 2 3
2 3 5
3 4 6
```

In either case, the missing list is `3 4 6`.

Problem C. BFFs

Small input 16 points	Solve C-small
Large input 29 points	Solve C-large

Problem

You are a teacher at the brand new Little Coders kindergarten. You have **N** kids in your class, and each one has a different student ID number from 1 through **N**. Every kid in your class has a single best friend forever (BFF), and you know who that BFF is for each kid. BFFs are not necessarily reciprocal -- that is, B being A's BFF does not imply that A is B's BFF.

Your lesson plan for tomorrow includes an activity in which the participants must sit in a circle. You want to make the activity as successful as possible by building the largest possible circle of kids such that each kid in the circle is sitting directly next to their BFF, either to the left or to the right. Any kids not in the circle will watch the activity without participating.

What is the greatest number of kids that can be in the circle?

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 of a test case contains a single integer **N**, the total number of kids in the class. The second line of a test case contains **N** integers $\mathbf{F_1}$, $\mathbf{F_2}$, ..., $\mathbf{F_N}$, where $\mathbf{F_i}$ is the student ID number of the BFF of the kid with student ID 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 maximum number of kids in the group that can be arranged in a circle such that each kid in the circle is sitting next to his or her BFF.

Limits

$1 \leq \mathbf{T} \leq 100$.
$1 \leq \mathbf{F_i} \leq \mathbf{N}$, for all i.
$\mathbf{F_i} \neq$ i, for all i. (No kid is their own BFF.)

Small dataset

$3 \leq \mathbf{N} \leq 10$.

Large dataset

$3 \leq \mathbf{N} \leq 1000$.

Sample

Input	Output
4 4 2 3 4 1 4 3 3 4 1 4 3 3 4 3 10 7 8 10 10 9 2 9 6 3 3	Case #1: 4 Case #2: 3 Case #3: 3 Case #4: 6

In sample case #4, the largest possible circle seats the following kids in the following order: 7 9 3 10 4 1. (Any reflection or rotation of this circle would also work.) Note that the kid with student ID 1 is next to the kid with student ID 7, as required, because the list represents a circle.

Powered by

Google Cloud Platform

Problem A. Getting the Digits

Small input 11 points	Solve A-small
Large input 12 points	Solve A-large

Problem

You just made a new friend at an international puzzle conference, and you asked for a way to keep in touch. You found the following note slipped under your hotel room door the next day:

"Salutations, new friend! I have replaced every digit of my phone number with its spelled-out uppercase English representation ("ZERO", "ONE", "TWO", "THREE", "FOUR", "FIVE", "SIX", "SEVEN", "EIGHT", "NINE" for the digits 0 through 9, in that order), and then reordered all of those letters in some way to produce a string **S**. It's up to you to use **S** to figure out how many digits are in my phone number and what those digits are, but I will tell you that my phone number consists of those digits in nondecreasing order. Give me a call... if you can!"

You would to like to call your friend to tell him that this is an obnoxious way to give someone a phone number, but you need the phone number to do that! What is it?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each consists of one line with a string **S** of uppercase English letters.

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 string of digits: the phone number.

Limits

$1 \leq \mathbf{T} \leq 100$.
A unique answer is guaranteed to exist.

Small dataset

3 ≤ length of **S** ≤ 20.

Large dataset

3 ≤ length of **S** ≤ 2000.

Sample

Input	Output
4 OZONETOWER WEIGHFOXTOURIST OURNEONFOE ETHER	Case #1: 012 Case #2: 2468 Case #3: 114 Case #4: 3

Problem B. Close Match

Small input 10 points	Solve B-small
Large input 23 points	Solve B-large

Problem

You are attending the most important game in sports history. The Oceania Coders are playing the Eurasia Jammers in the Centrifugal Bumble-Puppy world finals. Unfortunately, you were sleep deprived from all the anticipation, so you fell asleep during the game!

The scoreboard is currently displaying both scores, perhaps with one or more leading zeroes (because the scoreboard displays a fixed number of digits). While you were asleep, some of the lights on the scoreboard were damaged by strong ball hits, so one or more of the digits in one or both scores are not being displayed.

You think close games are more exciting, and you would like to imagine that the scores are as close as possible. Can you fill in all of the missing digits in a way that minimizes the absolute difference between the scores? If there is more than one way to attain the minimum absolute difference, choose the way that minimizes the Coders' score. If there is more than one way to attain the minimum absolute difference while also minimizing the Coders' score, choose the way that minimizes the Jammers' score.

Input

The first line of the input gives the number of test cases, **T**. **T** cases follow. Each case consists of one line with two non-empty strings **C** and **J** of the same length, composed only of decimal digits and question marks, representing the score as you see it for the Coders and the Jammers, respectively. There will be at least one question mark in each test case.

Output

For each test case, output one line containing `Case #x: c j`, where x is the test case number (starting from 1), c is **C** with the question marks replaced by digits, and `j` is **J** with the question marks replaced by digits, such that the absolute difference between the integers represented by `c` and `j` is minimized. If there are multiple solutions with the same absolute difference, use the one in which `c` is minimized; if there are multiple solutions with the same absolute difference and the same value of `c`, use the one in which `j` is minimized.

Limits

$1 \leq \mathbf{T} \leq 200$.
C and **J** have the same length.

Small dataset

1 ≤ the length of **C** and **J** ≤ 3.

Large dataset

1 ≤ the length of **C** and **J** ≤ 18.

Sample

Input	Output
4 1? 2? ??2? ??3 ? ? ?5 ?0	Case #1: 19 20 Case #2: 023 023 Case #3: 0 0 Case #4: 05 00

In sample case #4, note that the answer cannot be `15 10`; that minimizes the absolute difference, but does not minimize the Coders' score. Nor can the answer be `05 10`; that minimizes the absolute difference and the Coders' score, but does not minimize the Jammers' score.

Problem C. Technobabble

Small input 14 points	Solve C-small
Large input 30 points	Solve C-large

Problem

Every year, your professor posts a blank sign-up sheet for a prestigious scientific research conference on her door. If a student wants to give a lecture at the conference, they choose a two-word topic that is not already on the sheet and write it on the sheet. Once the deadline has passed, the professor has one of her grad students put the topics in a random order, to avoid biasing for or against students who signed up earlier. Then she presents the topics to you for review.

Since the snacks at the conference are excellent, some students try to fake their way into the conference. They choose the first word of some topic already on the sheet and the second word of some topic already on the sheet, and combine them (putting the first word first, and the second word second) to create a new "topic" (as long as it isn't already on the sheet). Since your professor is open-minded, sometimes this strategy actually works!

The fakers are completely unoriginal and can't come up with any new first or second words on their own; they must use existing ones from the sheet. Moreover, they won't try to use an existing first word as their own second word (unless the word also already exists on the sheet as a second word), or vice versa.

You have a list of all **N** of the submitted topics, in some arbitrary order; you don't know the order in which they were actually written on the sheet. What's the largest number of them that could have been faked?

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**, followed by **N** lines, each of which represents a different topic and has two strings of uppercase English letters: the two words of the topic, in order.

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 an integer: the largest number of topics that could have possibly been faked.

Limits

$1 \le \mathbf{T} \le 100$.
$1 \le$ length of each word ≤ 20.
No topic is repeated within a case.

Small dataset

$1 \le \mathbf{N} \le 16$.

Large dataset

$1 \le \mathbf{N} \le 1000$.

Sample

Input	Output
3 3 HYDROCARBON COMBUSTION QUAIL BEHAVIOR QUAIL COMBUSTION 3 CODE JAM SPACE JAM PEARL JAM 2 INTERGALACTIC PLANETARY PLANETARY INTERGALACTIC	Case #1: 1 Case #2: 0 Case #3: 0

In sample case #1, one possibility is that the topics were added to the sheet in this order:

`QUAIL BEHAVIOR` (real)
`HYDROCARBON COMBUSTION` (real)
`QUAIL COMBUSTION` (fake)

There is no scenario in which more than one of the topics can be fake.

In sample case #2, all of the topics must be real. Whatever order they were written in, at no point would it have been possible to use existing words to create a new topic that was not already on the list.

In sample case #3, neither topic can be fake. For example, if `INTERGALACTIC PLANETARY` had been the first and only topic written on the sheet, a faker could only have used `INTERGALACTIC` as the first word of a new topic and could only have used `PLANETARY` as the second word of a new topic... but the only topic that the faker could have formed would have been `INTERGALACTIC PLANETARY`, which would have been off limits since it was already on the sheet. So `PLANETARY INTERGALACTIC` must have also been a real topic.

Round 1C 2016

A. Senate Evacuation
B. Slides!
C. Fashion Police

Contest Analysis
Questions asked

Problem A. Senate Evacuation

Small input 8 points	Solve A-small
Large input 10 points	Solve A-large

Problem

A small fire started in the senate room, and it needs to be evacuated!

There are some senators in the senate room, each of whom belongs to of one of **N** political parties. Those parties are named after the first **N** (uppercase) letters of the English alphabet.

The emergency door is wide enough for up to two senators, so in each step of the evacuation, you may choose to remove either one or two senators from the room.

The senate rules indicate the senators in the room may vote on any bill at any time, even in the middle of an evacuation! So, the senators must be evacuated in a way that ensures that no party ever has an absolute majority. That is, it can never be the case after any evacuation step that more than half of the senators in the senate room belong to the same party.

Can you construct an evacuation plan? The senate is counting on you!

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 parties. The second line contains **N** integers, $\mathbf{P_1}$, $\mathbf{P_2}$, ..., $\mathbf{P_N}$, where $\mathbf{P_i}$ represents the number of senators of the party named after the i-th letter of the alphabet.

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 evacuation plan. The plan must be a space-separated list of instructions, in the order in which they are to be carried out, where each instruction is either one or two characters, representing the parties of the senators to evacuate in each step.

It is guaranteed that at least one valid evacuation plan will exist. If multiple evacuation plans are valid, you may output any of them.

Limits

$1 \leq \mathbf{T} \leq 50$.
No party will have an absolute majority before the start of the evacuation.
$1 \leq \mathbf{P_i} \leq 1000$, for all i.

Small dataset

$2 \leq \mathbf{N} \leq 3$.
sum of all $\mathbf{P_i} \leq 9$.

Large dataset

$2 \leq \mathbf{N} \leq 26$.
sum of all $\mathbf{P_i} \leq 1000$.

Sample

Input	Output
4 2 2 2 3 3 2 2 3 1 1 2 3 2 3 1	Case #1: AB BA Case #2: AA BC C BA Case #3: C C AB Case #4: BA BB CA

The sample output displays one set of answers to the sample cases. Other answers may be possible.

In Case #1, there are two senators from each of the parties A and B. If we remove one from each party every time, the perfect balance is maintained until evacuation is complete.

Case #2 proceeds as follows:

Initially in the room: 3 A, 2 B, 2 C.
Evacuate AA. Still in the room: 1 A, 2 B, 2 C.
Evacuate BC. Still in the room: 1 A, 1 B, 1 C.
Evacuate C. Still in the room: 1 A, 1 B.
Evacuate AB. Evacuation complete!

Note that it would not be valid to begin the evacuation with BC, which would leave 3 A, 1 B, and 1 C in the room; party A would have an absolute majority (3 out of 5 = 60%).

For Case #3, note that `CC AB` would also be a valid answer, and `C C AB` is also valid even though it requires three evacuation steps instead of two.

Problem B. Slides!

Small input
13 points
Solve B-small

Large input
21 points
Solve B-large

Problem

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

The CEO wants to build a set of slides between buildings that she can use to travel from her office in building 1 to her favorite cafe in building **B**. Slides, of course, are one-way only, but the buildings are tall and have elevators, so a slide can start in any building and end in any other building, and can go in either direction. Specifically, for any two buildings x and y, you can build either zero or one slides from x to y, and you can build either zero or one slides from y to x. The exception is that no slides are allowed to originate in building **B**, since once the CEO reaches that building, there is no need for her to do any more sliding.

In honor of Gooli becoming exactly **M** milliseconds old, the design must ensure that the CEO has *exactly* **M** different ways to travel from building 1 to building **B** using the new slides. A way is a sequence of buildings that starts with building 1, ends with building **B**, and has the property that for each pair of consecutive buildings x and y in the sequence, a slide exists from x to y. Note that the CEO is *not* requiring that any building be reachable from any other building via slides.

Can you come up with any set of one or more slides that satisfies the CEO's requirements, or determine that it is impossible?

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow; each consists of one line with two integers **B** and **M**, 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 word `POSSIBLE` or `IMPOSSIBLE`, depending on whether the CEO's requirements can be fulfilled or not. If it is possible, output an additional **B** lines containing **B** characters each, representing a matrix describing a valid way to build slides according to the requirements. The j-th character of the i-th of these lines (with both i and j counting starting from 1) should be 1 if a slide should be built going from building i to building j, and `0` otherwise. The i-th character of the i-th line should always be `0`, and every character of the last line should be `0`.

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

Limits

$1 \le \mathbf{T} \le 100$.

Small dataset

$2 \le \mathbf{B} \le 6$.
$1 \le \mathbf{M} \le 20$.

Large dataset

$2 \le \mathbf{B} \le 50$.
$1 \le \mathbf{M} \le 10^{18}$.

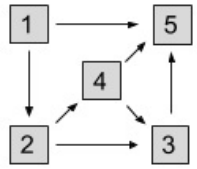
Sample

Input	Output
3 5 4 2 1 4 20	Case #1: POSSIBLE 01001 00110 00001 00101 00000 Case #2: POSSIBLE 01 00

```
        Case #3: IMPOSSIBLE
```

The sample outputs show one possible way to fulfill the specifications for each case. Other valid answers may exist.

Here is an illustration of the sample answer for Case #1:



The four ways to get from building 1 to building 5 are:

- 1 to 5
- 1 to 2 to 3 to 5
- 1 to 2 to 4 to 5
- 1 to 2 to 4 to 3 to 5

In Case #3, building slides from 1 to 2, 2 to 3, 3 to 1, and 1 to 4 would create infinitely many ways for the CEO to reach building 4 (she could go directly to 4, or go around the loop once and then go to 4, or go around the loop twice...), but the CEO requested *exactly* 20 ways.

Round 1C 2016

A. Senate Evacuation
B. Slides!
C. Fashion Police

Contest Analysis
Questions asked

Problem C. Fashion Police

Small input 14 points	Solve C-small
Large input 34 points	Solve C-large

Problem

You are so excited about the 2016 Code Jam World Finals that you just moved to New York. You have brought along **J** different jackets (numbered 1 through **J**), **P** different pairs of pants (numbered 1 through **P**), and **S** different shirts (numbered 1 through **S**). You have at least as many shirts as pairs of pants, and at least as many pairs of pants as jackets. (**J** ≤ **P** ≤ **S**.)

Every day, you will pick one jacket, one pair of pants, and one shirt to wear as an *outfit*. You wash all of your garments every night so all of your garments are available to use each day.

In New York, the Fashion Police officers are always watching and keeping track of what everyone wears every day. If they find out that you have worn the exact same outfit twice, you will immediately be taken to the Fashion Jail on 5th Avenue for a mandatory makeover; you definitely want to avoid that! You will also immediately be taken to Fashion Jail if they find out that you have worn the same two-garment *combination* more than **K** times in total. A combination consists of a particular jacket worn with a particular pair of pants, a particular jacket worn with a particular shirt, or a particular pair of pants worn with a particular shirt. For example, in the set of outfits (jacket 1, pants 2, shirt 3) and (jacket 1, pants 1, shirt 3), the combination (jacket 1, shirt 3) appears twice, whereas the combination (pants 1, shirt 3) only appears once.

You will wear one outfit per day. Can you figure out the largest possible number of days you can avoid being taken to Fashion Jail and produce a list of outfits to use each day?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow; each consists of one line with four integers **J**, **P**, **S**, 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 an integer: the maximum number of days you will be able to avoid being taken to Fashion Jail. Then output y more lines, each of which consists of three integers: the numbers of the jacket, pants, and shirt (in that order) for one day's outfit. The list of outfits can be in any order, but the outfits must not cause you to go to Fashion Jail as described in the statement above.

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

Limits

1 ≤ **T** ≤ 100.
1 ≤ **J** ≤ **P** ≤ **S**.
1 ≤ **K** ≤ 10.

Small dataset

S ≤ 3.

Large dataset

S ≤ 10.

Sample

Input	Output
4 1 1 1 10 1 2 3 2 1 1 3 2 1 2 3 1	Case #1: 1 1 1 1 Case #2: 4 1 1 2 1 2 3 1 2 1 1 1 1 Case #3: 2 1 1 2

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

The sample output displays one set of answers to the sample cases. Other answers may be possible.

In Case #1, even though the Fashion Police officers have set a lenient **K** value of 10, there is only one possible outfit that you can form, so you can only avoid Fashion Jail for one day.

In Case #2, adding any other outfit would cause you to go to Fashion Jail:

- Adding 1 1 3 would use the combination (jacket 1, pants 1) more than 2 times.
- Adding 1 2 2 would use the combination (jacket 1, pants 2) more than 2 times.

In this case, any set of 5 outfits would include at least one fashion violation.

Note that the numbers of the jacket, pants, and shirt within an individual outfit do not have to be in nondecreasing order in the same way that **J**, **P**, and **S** do.

In Case #3, you have only one jacket + pants combination which you must keep reusing, so no matter which shirts you wear, you cannot form more than **K** = 2 different outfits.

In Case #4, another possible maximally large set of outfits is:
1 2 2
1 1 1

Problem A. Rather Perplexing Showdown

Small input 4 points	Solve A-small
Large input 14 points	Solve A-large

Problem

You've been asked to organize a Rock-Paper-Scissors tournament. The tournament will have a single-elimination format and will run for **N** rounds; $2^{\mathbf{N}}$ players will participate.

Initially, the players will be lined up from left to right in some order that you specify. In each round, the first and second players in the lineup (starting from the left) will play a match against each other, and the third and fourth players in the lineup (if they exist) will play a match against each other, and so on; all of these matches will occur simultaneously. The winners of these matches will remain in the lineup, in the same relative order, and the losers will leave the lineup and go home. Then a new round will begin. This will continue until only one player remains in the lineup; that player will be declared the winner.

In each Rock-Paper-Scissors match, each of the two players secretly chooses one of *Rock*, *Paper*, or *Scissors*, and then they compare their choices. Rock beats Scissors, Scissors beats Paper, and Paper beats Rock. If one player's choice beats the other players's choice, then that player wins and the match is over. However, if the players make the same choice, then it is a tie, and they must choose again and keep playing until there is a winner.

You know that the players this year are stubborn and not very strategic. Each one has a *preferred move* and will only play that move in every match, regardless of what the opponent does. Because of this, if two players with the same move go up against each other, they will keep tying and their match will go on forever! If this happens, the tournament will never end and you will be a laughingstock.

This year, there are **R** players who prefer Rock, **P** players who prefer Paper, and **S** players who prefer Scissors. Knowing this, you want to create a lineup that guarantees that the tournament will go to completion and produce a single winner — that is, no match will ever be a tie. Your boss has asked you to produce a list of all such lineups (written in left to right order, with R, P, and S standing for players who prefer Rock, Paper, and Scissors, respectively), and then put that list in alphabetical order.

You know that the boss will lazily pick the first lineup on the list; what will that be? Or do you have to tell your boss that it is `IMPOSSIBLE` to prevent a tie?

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow; each represents one test case. Each test case consists of four integers: **N**, **R**, **P**, and **S**, as described in the statement 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 either `IMPOSSIBLE` or a string of length $2^{\mathbf{N}}$ representing the alphabetically earliest starting lineup that solves the problem. Every character in a lineup must be R, P, or S, and there must be **R** Rs, **P** Ps, and **S** Ss.

Limits

$\mathbf{R} + \mathbf{P} + \mathbf{S} = 2^{\mathbf{N}}$.
$0 \leq \mathbf{R} \leq 2^{\mathbf{N}}$.
$0 \leq \mathbf{P} \leq 2^{\mathbf{N}}$.
$0 \leq \mathbf{S} \leq 2^{\mathbf{N}}$.

Small dataset

$1 \leq \mathbf{T} \leq 25$.
$1 \leq \mathbf{N} \leq 3$.

Large dataset

$1 \leq \mathbf{T} \leq 75$.
$1 \leq \mathbf{N} \leq 12$.

Sample

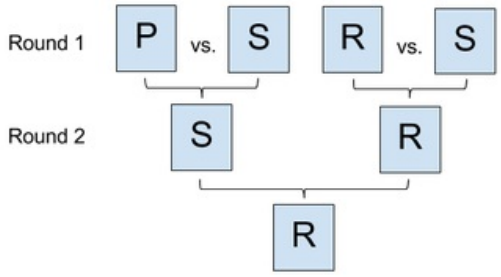
```
Input       Output

4           Case #1: PR
1 1 1 0     Case #2: IMPOSSIBLE
1 2 0 0     Case #3: PSRS
2 1 1 2     Case #4: IMPOSSIBLE
2 2 0 2
```

In sample case #1, there are only two players and the tournament will consist of one round. It doesn't matter what order the two line up in; the Paper-using player will defeat the Rock-using player. You will give your boss the alphabetically ordered list PR, RP, and the first element is PR.

In sample case #2, the only two players both play Rock, so a tie is unavoidable.

In sample case #3, there are four players and the tournament will go on for two rounds. In the first round, the first player (Paper) will lose to the second player (Scissors), and the third player (Rock) will defeat the fourth player (Scissors). The second round lineup will be PR, and the first remaining player (Paper) will defeat the other remaining player (Rock), so the tournament will end with a winner and no ties.

Here is an illustration of the tournament for sample case #3:



Other lineups such as PSSR will appear on the list you give to your boss, but PSRS is alphabetically first.

In sample case #4, the only way to organize the first round such that there are no ties is to create two matches with one Rock player and one Scissors player. But both of those matches will have a Rock winner, and when these two winners go on to face each other, there will be a tie.

Problem B. Red Tape Committee

Small input 5 points	Solve B-small
Large input 17 points	Solve B-large

Problem

You are the head of the Department of Redundancy Reduction and Superfluity Shrinkage. Currently, the department cannot agree on whether there is too much "red tape" (inefficiency) in the department itself. They have asked you to form a Red Tape Committee to vote on the issue.

The department has **N** members. For each member, you know the probability $\mathbf{P_i}$ that that member will vote "Yes". If a member does not vote "Yes", they necessarily vote "No"; nobody abstains.

You must choose exactly **K** members to be on the committee. The department rules dictate that **K** must be an even number to allow for ties, which are seen as part of a healthy bureaucracy.

If you choose committee members to *maximize* the probability of a tie, what is that probability?

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 a test case consists of two integers **N** and **K**, the sizes of the department and the committee. The second line of a test case consists of **N** decimal values $\mathbf{P}_i$; each has exactly two decimal places of precision and represents the probability that the *i*-th department member will vote "Yes".

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 floating-point number: the maximum possible probability of a tie. 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

$1 \le \mathbf{T} \le 100$.
$2 \le \mathbf{K} \le \mathbf{N}$.
K is even.
$0.00 \le$ each $\mathbf{P_i} \le 1.00$.

Small dataset

$2 \le \mathbf{N} \le 16$.

Large dataset

$2 \le \mathbf{N} \le 200$.

Sample

Input	Output
3 2 2 0.50 0.50 4 2 0.00 0.00 1.00 1.00 3 2 0.75 1.00 0.50	Case #1: 0.5 Case #2: 1.0 Case #3: 0.5

In sample case #1, you must use the only two available department members to form the committee. That committee will tie only if the two committee members vote differently, which will happen half the time. (Without loss of generality, choose the vote of the first. Then the probability that the second will vote the other way is 0.5.)

In sample case #2, the best strategy is to pick one of the members with "Yes" probability 0.00 and one of the members with "Yes" probability 1.00. This guarantees a tie.

In sample case #3, suppose that we pick the two members with "Yes" probabilities of 0.50 and 0.75. A tie will happen if the first one votes "Yes" and the second one votes "No" (probability 0.5 * 0.25 = 0.125), or if the first one votes "No" and the second one votes "Yes" (probability 0.5 * 0.75 = 0.375). So the total probability of a tie is 0.125 + 0.375 = 0.5. Choosing the two members with "Yes" probabilities of 0.50 and 1.00 would also make the tie probability 0.5, since the 1.00 member will vote "Yes" and the 0.50 member must vote "No". Choosing the two members with "Yes" probabilities of 0.75 and 1.00 would make the tie probability only 0.25, since the 1.00 member will vote "Yes" and the 0.75 member must vote "No". So 0.5 is the best we can do.

Problem C. The Gardener of Seville

Small input 6 points	Solve C-small
Large input 23 points	Solve C-large

Problem

You are the Gardener of Seville, a minor character in an opera. The setting for the opera is a courtyard which is a rectangle of unit cells, with **R** rows and **C** columns. You have been asked to install a maze of hedges in the courtyard: every cell must contain a hedge that runs diagonally from one corner to another. For any cell, there are two possible kinds of hedge: lower left to upper right, which we represent with `/`, and upper left to lower right, which we represent with `\`. Wherever two hedges touch, they form a continuous wall.

Around the courtyard is an outer ring of unit cells, one cell wide, with the four corners missing. Each of these outer cells is the home of a courtier. The courtiers in these outer cells are numbered clockwise, starting with 1 for the leftmost of the cells in the top row, and ending with 2 * (**R** + **C**) for the topmost cell in the left column. For example, for **R** = 2, **C** = 2, the numbering in the outer ring looks like this. (Note that no hedges have been added yet.)

```
 12
8  3
7  4
 65
```

In this unusual opera, love is mutual and exclusive: each courtier loves exactly one other courtier, who reciprocally loves only them. Each courtier wants to be able to sneak through the hedge maze to his or her lover without encountering any other courtiers. That is, any two courtiers in love with each other must be connected by a path through the maze that is separated from every other path by hedge walls. It is fine if there are parts of the maze that are not part of any courtier's path, as long as all of the pairs of lovers are connected.

Given a list of who loves who, can you construct the hedge maze so that every pair of lovers is connected, or determine that this 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 with two integers **R** and **C**, followed by another line with a permutation of all of the integers from 1 to 2 * (**R** + **C**), inclusive. Each integer is the number of a courtier; the first and second courtiers in the list are in love and must be connected, the third and fourth courtiers in the list are in love and must be connected, and so on.

Output

For each test case, output one line containing *only* `Case #x:`, where x is the test case number (starting from 1). Then, if it is impossible to satisfy the conditions, output one more line with the text `IMPOSSIBLE`. Otherwise, output **R** more lines of **C** characters each, representing a hedge maze that satisfies the conditions, where every character is `/` or `\`. You may not leave any cells in the maze blank. If multiple mazes are possible, you may output any one of them.

Limits

Small dataset

1 ≤ **T** ≤ 100.
1 ≤ **R** * **C** ≤ 16.

Large dataset

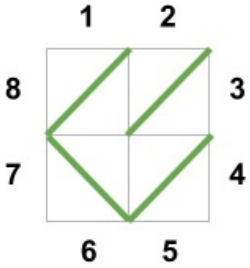
1 ≤ **T** ≤ 500.
1 ≤ **R** * **C** ≤ 100.

Sample

Input	Output
4 1 1 1 4 3 2 1 3	Case #1: / Case #2: //\

```
1 8 2 7 3 4 5 6   Case #3:
2 2               //
8 1 4 5 2 3 7 6   \/
1 1               Case #4:
1 3 2 4           IMPOSSIBLE
```

In Case #3, the following pairs of courtiers are lovers: (8, 1), (4, 5), (2, 3), (7, 6). Here is an illustration of our sample output:



For Case #3, note that this would also be a valid maze:

```
/\
\/
```

In Case #4, the courtyard consists of only one cell, so the courtiers living around it, starting from the top and reading clockwise, are 1, 2, 3, and 4. There are only two possible options to put in the one cell: / or \. The first of these choices would form paths from 1 to 4, and from 2 to 3. The second of these choices would form paths from 1 to 2, and from 3 to 4. However, neither of these helps our lovesick courtiers, since in this case, 1 loves 3 and 2 loves 4. So this case is IMPOSSIBLE, and the opera will be full of unhappy arias!

Problem D. Freeform Factory

Small input 6 points	Solve D-small
Large input 25 points	Solve D-large

Problem

You have just built a brand new factory. Your factory has **N** different machines, and each machine needs to be *operated* by exactly one worker for the factory to function well.

You have also hired **N** workers to operate those machines. Since you were in a hurry when you hired them, you did not check whether they actually know how to operate your machines. Now you have finally asked them, and so you have the information on whether the i-th worker can operate the j-th machine, for each i and j.

In a typical working day, the workers will arrive at the factory in a random order, which can be different each day. As each worker arrives, they will find all machines that they know how to operate and that do not already have an operator. They will choose one of those at random and operate it for the whole working day. If all machines they know how to operate already have an operator, they will not work that day. Your goal is to make sure that all machines are being operated each working day, regardless of what order the workers arrive in and which machines they choose.

For example, suppose there are two workers A and B, and two machines 1 and 2. Suppose that A knows how to operate 1 and 2, and B knows how to operate 1 but not 2. If worker B arrives first, he will pick machine 1, then when worker A arrives she will have to choose 2, and the factory will work well. However, if worker A arrives first, it might happen that she chooses to operate 1 on that day, and then when worker B arrives he does not have anything to do, leaving machine 2 without an operator, and causing your factory to waste a whole day!

As another example, suppose there are two workers A and B, and two machines 1 and 2, and that A knows how to operate 1 but not 2, and B does not know how to operate anything. Then, regardless of the order in which the workers arrive, the factory will not be able to function well.

Before you open your factory, in order to guarantee that the factory will constantly function well, you can teach your workers how to operate machines. It costs one dollar to give a single worker a lesson on how to operate a single machine. Each lesson involves only one worker and only one machine, but you can teach any number of lessons to any number of workers, and the same worker can receive multiple lessons. You cannot make a worker forget how to operate a machine if they already know how to operate it.

For example, both examples above can be fixed by teaching worker B to operate machine 2. In that case each machine is guaranteed to have an operator every day, regardless of which order the workers arrive in and which machines they choose to operate when they have more than one possibility.

What is the minimum amount of dollars you need to spend on training workers to make sure the factory functions well every day?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case starts with one line with an integer **N**, the number of workers (and machines). This line is followed by **N** lines with a string of **N** characters each. The j-th character on the i-th of those lines is `1` if the i-th worker knows how to operate the j-th machine, and `0` otherwise.

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 non-negative integer: the minimum amount of dollars you need to spend to make sure that all **N** machines will always have an operator.

Limits

$1 \leq \mathbf{T} \leq 100$.

Small dataset

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

Large dataset

$1 \leq \mathbf{N} \leq 25$.

Sample

Input	Output
5 2 11 10 2 10 00 3 000 000 000 1 1 3 000 110 000	Case #1: 1 Case #2: 1 Case #3: 3 Case #4: 0 Case #5: 3

Sample cases #1 and #2 are the ones described in the problem statement.

In sample case #3, nobody knows how to do anything! One optimal strategy is to teach worker A to operate machine 1, worker B to operate machine 2, and worker C to operate machine 3.

In sample case #4, no action is necessary. There is only one worker, and the worker already knows how to operate the one machine.

In sample case #5, worker B already knows how to operate machines 1 and 2. One optimal strategy is to teach worker A to operate machine 3, and make A the only worker who can operate that machine. But now we have to consider that B might operate either machine 1 or 2 upon arrival, so C needs to be able to operate the one not chosen by B. So C must be taught to operate both 1 and 2.

Problem A. Teaching Assistant

Small input 5 points	Solve A-small
Large input 10 points	Solve A-large

Problem

You are taking a programming course which is graded using problem sets of different types. The course goes for a positive even number of days. You start the course with no problem sets. On each day of the course, you *must* do *exactly one* of the following:

- Request a "Coding" problem set.
- Request a "Jamming" problem set.
- Submit a problem set for grading. You must have at least one problem set to choose this option. If you have multiple problem sets, you must submit the one among those that was *requested most recently*, regardless of its type.

All problem sets are different. There is no requirement on how many sets of each type must be submitted. Once you submit a set, you no longer have that set. Any problem sets that you have not submitted before the end of the course get you no points.

The problem sets are requested from and submitted to an artificially-intelligent teaching assistant. Strangely, the assistant has different moods — on each day it is in the mood for either "Coding" or "Jamming".

- When you request a problem set:
  - If the requested topic matches the assistant's mood, it assigns a problem set worth a maximum of 10 points.
  - If the requested topic does not match its mood, it assigns a problem set worth a maximum of 5 points.
- When you submit a problem set:
  - If the topic of the submitted set matches the assistant's mood that day, it gives you the maximum number of points for that set.
  - If the topic of the submitted set does not match its mood that day, it gives you 5 points fewer than the maximum number of points for that set.

For example:

- If you request a "Coding" problem set on a day in which the assistant is in a "Coding" mood, and submit it on a day in which it is in a "Jamming" mood, you will earn 5 points: the problem set is worth a maximum of 10, but the assistant gives 5 points fewer than that.
- If you request a "Jamming" problem set on a day in which the assistant is in a "Coding" mood, and submit it on a day in which it is in a "Jamming" mood, you will earn 5 points: the set is worth a maximum of 5, and the assistant gives you the maximum number of points.

Thanks to some help from a senior colleague who understands the assistant very well, you know what sort of mood the assistant will be in on each day of the course. What is the maximum total score that you will be able to obtain?

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 a string **S** of C and/or J characters. The i-th character of **S** denotes the assistant's mood on the i-th day of the course. If it is C, it is in the mood for "Coding"; if it is J, it is in the mood for "Jamming".

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 maximum number of points you can obtain for that case.

Limits

$1 \leq \mathbf{T} \leq 100$.
The length of **S** is even.

Small dataset

$2 \leq$ the length of $\mathbf{S} \leq 50$.

Large dataset

2 ≤ the length of **S** ≤ 20000.
The sum of lengths of all **S** in the dataset is at most 150000.

Sample

```
Input      Output

5          Case #1: 20
CCJJ       Case #2: 10
CJCJ       Case #3: 20
CJJC       Case #4: 15
CJJJ       Case #5: 30
CCCCCC
```

This strategy is optimal for sample case #1:
Day 1: Request a "Coding" problem set (call it C1).
Day 2: Submit C1.
Day 3: Request a "Jamming" problem set (call it J1).
Day 4: Submit J1.

The following strategy is optimal for sample cases #2, #3, and #4: request C1, request J1, submit J1, submit C1.

For case #2, for example, note that you could *not* request C1, request J1, and then submit C1. Only the most recently requested problem set can be submitted.

In sample case #5, you can alternate between requesting a "Coding" problem set on one day, and submitting it on the next day.

Round 3 2016

A. Teaching Assistant
B. Forest University
C. Rebel Against The Empire
D. Go++

Contest Analysis
Questions asked

Problem B. Forest University

Small input
25 points

Solve B-small

Problem

The Forest University offers its students **N** courses, which must all be taken to obtain the degree. The courses can only be taken one at a time - you must complete a course before starting another. Each course is either *basic*, which means one can take it without any prior knowledge, or *advanced*, in which case exactly one other course is its *prerequisite*.

A student must take the prerequisite for a course before taking the course, although they do not need to be taken immediately one after the other. A course might be the prerequisite for multiple other courses. There are no prerequisite cycles. Any sequence of the **N** courses that meets the rules for prerequisites is valid for obtaining the degree.

When you graduate, the university commemorates the sequence of courses you have taken by printing an abbreviated version of it on your graduation hat. More precisely, this abbreviated version is a string consisting of the first letter of the name of each course you have taken, in the order you have taken them. For example, if you have taken a Coding course and a Jamming course, in that order, your graduation hat will say CJ. It is considered trendy to have certain *cool words* as a substring of the string on one's graduation hat.

Consider all possible valid sequences in which the courses can be taken. For each cool word, you need to find the fraction of those sequences that have the cool word as a substring (at least once) of the string on the corresponding graduation hat. Note that we're interested in the fraction of possible course sequences, *not* the fraction of possible different graduation hat strings. (Since multiple courses may start with the same letter, there may be fewer possible strings than course sequences.)

Somewhat unusually for Code Jam, we are only looking for an approximate answer to this problem; pay careful attention to the output format.

Solving this problem

This problem has only 1 Small input and no Large input. You will be able to retry the input (with a time penalty).

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case consists of five lines, in this order, which contain the following:

1. the number **N** of courses.
2. **N** integers; the i-th of these integers gives the number of the prerequisite course for the i-th course, or 0 if the i-th course is basic. The courses are numbered from 1 to **N**.
3. **N** uppercase English letters (without whitespace in between), with the i-th character giving the first letter of the i-th course's name.
4. the number **M** of cool words.
5. **M** cool words, each of which consists only of uppercase English letters.

Output

For each test case, output one line containing `Case #x: y1 y2 ... yM`, where x is the test case number (starting from 1) and y_i is the fraction of valid course sequences that will have the i-th cool word as a substring of the string on the graduation hat.

y_i will be considered correct if it is within an absolute error of 0.03 of the correct answer. See the FAQ for an explanation of what that means, and what formats of real numbers we accept.

Limits

$1 \le \mathbf{T} \le 100$.
$1 \le \mathbf{N} \le 100$.
$1 \le \mathbf{M} \le 5$.
The length of each cool word is between 1 and 20.
Each cool word consists of uppercase English letters only.
There are no cycles formed by the prerequisites.

Sample

Input	Output

```
2            Case #1: 1.0 1.0 0.0 0.0
2            Case #2: 0.67 0.0 0.33
0 1
CJ
4
CJ C D JC
3
0 1 0
BAA
3
AA AAB ABA
```

The sample output displays one set of acceptable answers to the sample cases. Other answers are possible within the allowed precision.

In sample case #1, course 1 (C) is a basic course that is a prerequisite for the advanced course 2 (J). The only way to complete the courses is to take course 1 and then course 2. This creates the string CJ. So the cool words CJ, C, D, and JC are present as substrings in 1, 1, 0, and 0 out of 1 possible cases, respectively.

In sample case #2, the basic course 1 (B) is a prerequisite for the advanced course 2 (A), and course 3 (A) is another basic course. There are three possible ways of completing the courses:

1. take course 1, then course 2, then course 3 (string: BAA)
2. take course 1, then course 3, then course 2 (string: BAA)
3. take course 3, then course 1, then course 2 (string: ABA)

The cool words AA, AAB, and ABA are present as substrings in 2, 0, and 1 out of 3 possible cases, respectively.

Problem C. Rebel Against The Empire

Small input 8 points	Solve C-small
Large input 17 points	Solve C-large

Problem

You are a rebel against the evil Galactic Empire, and you are on the run!

You have sabotaged the Empire's Factory of Evil, and imperial security forces will be after you soon! The factory is located on asteroid 0 in a system of **N** numbered asteroids. Your getaway ship, the Century Quail, is located on asteroid 1, and if you can get there, you will be able to fly away safely.

Each asteroid is a single point in space with a velocity, and you move through space along with whichever asteroid you are currently on. Your Asteroid Jumper will allow you to instantaneously jump between any two asteroids in the system. Long jumps are scarier than short ones (and the vacuum of space is terrifying), so you want to minimize the maximum distance you need to jump. However, starting now, if you ever spend more than a continuous **S** seconds without jumping, the imperial security forces will catch you. That is, the interval from now until your first jump, and each interval between subsequent jumps, must be less than or equal to **S**. You may jump at *any* instant; it does not have to be after an integer number of seconds have elapsed. You escape the instant you jump to asteroid 1.

The i-th asteroid starts at position ($\mathbf{x_i}$, $\mathbf{y_i}$, $\mathbf{z_i}$) in space, and it will move a total distance of ($\mathbf{V_{xi}}$, $\mathbf{V_{yi}}$, $\mathbf{V_{zi}}$) each second. This movement is continuous throughout time; it does not update discretely each second. (It is also possible for an asteroid to be stationary.) Nothing happens if asteroids occupy the same point in space at the same time. You can only travel between two asteroids by jumping, even if they happen to occupy the same point at the instant of your jump.

In the escape plan that minimizes the maximum jump distance, what is that maximum jump distance?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. The first line of each test case contains two integers: **N** (the number of asteroids) and **S** (the limit on how long you can go without jumping). Next, there are **N** lines describing the asteroids. The i-th of these lines (counting starting from 0) contains six integers: the initial ($\mathbf{x_i}$, $\mathbf{y_i}$, $\mathbf{z_i}$) position of the i-th asteroid in space, and the distance ($\mathbf{V_{xi}}$, $\mathbf{V_{yi}}$, $\mathbf{V_{zi}}$) it moves in a single second.

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 floating-point number: the distance of the longest jump you will have to make in order to get away. y will be considered correct if it is within an absolute or relative error of 10^{-4} of the correct answer. See the FAQ for an explanation of what that means, and what formats of real numbers we accept.

Limits

$1 \le \mathbf{T} \le 20$.
$2 \le \mathbf{N} \le 1000$.
$1 \le \mathbf{S} \le 100$.
$-500 \le \mathbf{x_i} \le 500$.
$-500 \le \mathbf{y_i} \le 500$.
$-500 \le \mathbf{z_i} \le 500$.

Small dataset

$\mathbf{V_{xi}} = 0$.
$\mathbf{V_{yi}} = 0$.
$\mathbf{V_{zi}} = 0$.

Large dataset

$-500 \le \mathbf{V_{xi}} \le 500$.
$-500 \le \mathbf{V_{yi}} \le 500$.
$-500 \le \mathbf{V_{zi}} \le 500$.

Sample

Input	Output
3 3 7 0 0 0 0 0 0 1 2 2 0 0 0 1 1 1 0 0 0 5 10 0 0 0 0 0 0 35 0 0 -1 0 0 1 54 0 0 -2 0 2 -150 0 0 10 0 4 0 0 -1 0 0 3 1 -10 2 0 1 0 0 0 0 10 0 0 -1 -10 -2 0 1 0 0	Case #1: 1.7320508 Case #2: 2.0000000 Case #3: 4.0000000

Sample case #1 is the only sample case that could appear in the Small dataset. Any of the sample cases could appear in the Large dataset.

In sample case #1, we start on a stationary asteroid at (0, 0, 0), and our ship is on an asteroid at (1, 2, 2). There is another asteroid at (1, 1, 1). One option is to jump directly to our ship, which is a distance of 3 away. Another option is to jump to the other asteroid, which is a distance of sqrt(3) away, and then jump to the ship from there, which is a distance of sqrt(2) away. The maximum jump distance is 3 for the first option and sqrt(3) for the second option, so the second option is preferable.

Note that the value of **S** does not matter in the Small cases. Since all of the asteroids are stationary, there is no reason to wait around; we can make all jumps instantaneously.

In sample case #2, we start on a stationary asteroid at (0, 0, 0). We can wait there for 4 seconds for asteroid 4 to come very close, jump onto it, fly for 1 second on it, and then jump back at time 5 back to asteroid 0 (the distance between the two asteroids is 1 at this moment). There we wait 10 seconds, cutting it very close to being caught, and then jump to the speeding asteroid 3 at time 15. Two seconds later, asteroid 3 flies by asteroid 2, and we jump to asteroid 2. At time 27, we can jump from asteroid 2 to asteroid 0. There we patiently wait until time 35 when asteroid 1 reaches us, then we can jump onto it and escape. The longest jump we made was from asteroid 0 to asteroid 3 at time 15, and the distance we jumped was 2.

In sample case #3, the security forces are really active! You could, of course, wait one second and jump directly to asteroid 1, but a better choice - that allows you to make jumps no longer than 4 - is to jump back and forth between asteroids 0 and 2; while waiting for asteroid 1 to get close, and only then jump to it.

Problem D. Go++

Problem

The Go language was designed to have a simple API and to support multi-threading. The Code Jam team wants to push these goals to the limit, so we are proposing a new language called Go++.

The Go++ language uses one register, which stores one boolean value (0 or 1). This register is initialized to 0. The language has three instructions:

- `0`, which sets the register to 0.
- `1`, which sets the register to 1.
- `?`, which prints the current register value.

Simple, right? To support multi-threading, we allow two different Go++ programs to run simultaneously while sharing the one register. Each instruction executes atomically — that is, one instruction must completely finish before the next instruction can start. However, the two programs may be interleaved in any way that preserves the relative order within each program.

For example, here are the only six ways in which the two programs `1?` and `?0` could be executed together. (The underline on the second program is just to distinguish its instructions from the instructions in the first program.)

- `?01?`, which will print `01`. (Remember that the register is initialized to 0.)
- `?10?`, which will print `00`.
- `?1?0`, which will print `01`.
- `1?0?`, which will print `10`.
- `1??0`, which will print `11`.
- `1??0`, which will print `11`.

Note that the output string always consists of `0`s and `1`s, and never `?`s, since `?` is not a state the register can be in.

Usually, programmers write programs to produce a desired output, but your task will be to write two programs that *won't* produce an *undesired* output! Specifically, you will be given a "bad" string **B** of length **L**, and a set **G** of **N** "good" strings, all of length **L**. You must produce two Go++ programs (not necessarily of the same length), which, when run in the way described here, could produce *all* of the strings in **G**, but could *not* produce the string **B**. It is fine if the programs could also produce other strings that are not **B** and not in **G**. Note that there must be a combined total of exactly **L** `?` instructions in the two programs. The combined number of instructions in the two programs must not exceed 200.

For example, for **B** = `11` and **G** = { `10`, `00` }, the programs `?` and `10?1` would be one valid answer. They can produce every string in **G**, but they cannot produce **B**, no matter how they are interleaved. (They can also produce the string `01`, which is not **B** and is not in **G**, but that is fine.) However, the programs `1?` and `?0` would not be a valid answer, since (as we saw above) they can produce **B**. The programs `00` and `??` would not be a valid answer, since they cannot produce every string in **G**.

Can you produce two programs that satisfy the conditions, or determine that the task is `IMPOSSIBLE`?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow; each consists of three lines. The first line of each test case has two integers **N** and **L**: the number of strings in **G**, and the length of the **B** string and the strings in **G**. The second line has **N** different strings of length **L**: the strings in **G**. The third line has one string of length **L**: the bad string **B**. **B** and all of the strings in **G** are made up of only `0`s and/or `1`s.

Output

For each test case, output one line containing `Case #x: IMPOSSIBLE`, if no programs will satisfy the conditions; otherwise, output `Case #x: y z`, where x is the test case number (starting from 1) and y and z are your two programs that satisfy the conditions. The combined number of instructions in your programs must not exceed 200. Each program must contain at least one instruction. There must be a combined total of exactly **L** `?` instructions in the two programs. If there are multiple correct outputs, print any of them.

Limits

$1 \le \mathbf{T} \le 100$.
$1 \le \mathbf{N} \le 100$.
$1 \le \mathbf{L} \le 50$.
All strings in **G** are different.

Small dataset

B consists entirely of 1s.

Large dataset

B may be any string consisting of 0s and/or 1s.

Sample

Input	Output
3 2 2 10 00 11 3 2 11 10 00 01 4 2 00 01 10 11 11	Case #1: ? 10?1 Case #2: 1?? 0 Case #3: IMPOSSIBLE

The sample output displays one set of answers to the sample cases. Other answers may be possible.

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

Sample case #2 would not appear in the Small dataset.

Sample case #3 is obviously `IMPOSSIBLE` because **B** is in **G**.

Problem A. Integeregex

Small input 15 points	Solve A-small
Large input 15 points	Solve A-large

Problem

In this problem, a valid regular expression is one of the following. In the following descriptions, E_1, E_2, etc. denote (not necessarily different) valid regular expressions.

- A decimal digit: that is, one of `0 1 2 3 4 5 6 7 8 9`.
- Concatenation: E_1E_2.
- Disjunction: $(E_1|E_2|...|E_N)$, for at least two expressions. Note that the outer parentheses are required.
- Repetition: $(E_1)*$. Note that the outer parentheses are required.

For example, `7`, `23`, `(7)*`, `(45)*`, `(1|2|3)`, `((2)*|3)`, `(1|2|3)`, and `((0|1))*` are valid expressions. `(7)`, `4|5`, `4*`, `(1|)`, and `(0|1)*` are not.

We say that an expression E matches a string of digits D if and only if at least one of the following is true:

- $E = D$.
- $E = E_1E_2$ and there exist D_1 and D_2 such that $D = D_1D_2$ and E_i matches D_i.
- $E = (E_1|E_2|...|E_N)$ and at least one of the E_i matches D.
- $E = (E_1)*$ and there exist D_1, D_2, ..., D_N for some non-negative integer N such that $D = D_1D_2...D_N$ and E_1 matches each of the D_i. In particular, note that $(E_1)*$ matches the empty string.

For example, the expression `((1|2))*3` matches 3, 13, 123, and 2221123, among other strings. However, it does *not* match 1234, 3123, 12, or 33, among other strings.

Given a valid regular expression **R**, for how many integers between **A** and **B**, inclusive, does **R** match the integer's base 10 representation (with no leading zeroes)?

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 has two positive integers **A** and **B**: the inclusive limits of the integer range we are interested in. The second has a string **R** consisting only of characters in the set `0123456789()|*`, which is guaranteed to be a valid regular expression as described in the statement 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 number of integers in the inclusive range [**A**, **B**] that the the regular expression **R** matches.

Limits

$1 \le \mathbf{T} \le 100$.
$1 \le \mathbf{A} \le \mathbf{B} \le 10^{18}$.
$1 \le$ length of $\mathbf{R} \le 30$.

Small dataset

R contains no | characters.

Large dataset

No additional limits.

Sample

Input	Output
8 1 1000	Case #1: 4 Case #2: 1

```
(0)*1(0)*                   Case #3: 5
379009 379009               Case #4: 0
379009                      Case #5: 4
1 10000                     Case #6: 2
(12)*(34)*                  Case #7: 1000000000000000000
4 5                         Case #8: 6
45
1 100
((0|1))*
1 50
(01|23|45|67|23)
1 1000000000000000000
((0|1|2|3|4|5|6|7|8|9))*
1 1000
1(56|(((7|8))*9)*)
```

Note that sample cases 5 through 8 would not appear in the Small dataset.

In sample case 1, the matches in range are 1, 10, 100, and 1000.

In sample case 2, the match in range is 379009.

In sample case 3, the matches in range are 12, 34, 1212, 1234, and 3434.

In sample case 4, there are no matches in range.

In sample case 5, the matches in range are 1, 10, 11, and 100.

In sample case 6, the matches in range are 23 and 45.

In sample case 7, it is possible to form any number in the range.

In sample case 8, the matches in range are 1, 19, 156, 179, 189, and 199.

Problem B. Family Hotel

Small input 10 points	Solve B-small
Large input 20 points	Solve B-large

Family Hotel

You run a hotel with **N** rooms arranged along one long corridor, numbered from 1 to **N** along that corridor. Your guests are big families, and every family asks for exactly two adjacent rooms when they arrive. Two rooms are adjacent if their numbers differ by exactly 1.

At the start of the day today, your hotel was empty. You have been using the following simple strategy to assign rooms to your guests. As each family arrives, you consider all possible pairs of adjacent rooms that are both free, pick one of those pairs uniformly at random, and assign the two rooms in that pair to the family. New families constantly arrive, one family at a time, but once there are no more pairs of adjacent rooms that are both free, you turn on the NO VACANCY sign and you do not give out any more rooms.

Given a specific room number, what is the probability that it will be occupied at the time that you turn on the NO VACANCY sign?

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow. Each line contains two numbers: the number of rooms **N** and the room number **K** that we are interested in.

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 sought probability computed *modulo* 10^9+7, which is defined precisely as follows. Represent the probability that room **K** is occupied as an irreducible fraction p/q. The number y then must satisfy the modular equation $y \times q \equiv p \pmod{10^9+7}$, and be between 0 and 10^9+6, inclusive. It can be shown that under the constraints of this problem such a number y always exists and is uniquely determined.

Limits

$1 \le \mathbf{T} \le 100$.
$1 \le \mathbf{K} \le \mathbf{N}$.

Small dataset

$2 \le \mathbf{N} \le 10^4$.

Large dataset

$2 \le \mathbf{N} \le 10^7$.

Sample

Input	Output
4 3 1 3 2 4 1 4 2	Case #1: 500000004 Case #2: 1 Case #3: 666666672 Case #4: 1

In sample case #3, there are four rooms and we are looking for probability that the first room is occupied. When the first family arrives, there are 3 possible situations, each with probability 1/3: occupy rooms 1+2, 2+3 or 3+4. In the first situation, the first room is already occupied and will stay occupied. In the second situation, the first room is free and no more families can be accommodated, so it will stay free. Finally, in the third situation, the next arriving family will definitely get rooms 1+2, and thus the first room will be occupied. The probability that the first room is occupied is thus 2/3, and the answer is 666666672, since `(666666672 * 3) mod 1000000007 = 2 mod 1000000007`.

The probability for sample case #1 is 1/2, and for sample cases #2 and #4 it is 1.

Problem C. Gallery of Pillars

Small input 10 points	Solve C-small
Large input 30 points	Solve C-large

Problem

Your friend Cody-Jamal is working on his new artistic installment called "Gallery of Pillars". The installment is to be exhibited in a square gallery of **N** by **N** meters. The gallery is divided into $\mathbf{N}^2$ squares of 1 by 1 meter, forming an **N** by **N** matrix. The exact center of the southwest corner cell is called the *viewpoint*; a person viewing the artwork is supposed to stand there. Each other cell contains a cylindrical pillar. All pillars have two circular bases of radius **R**: one resting on the floor, in the center of its corresponding cell, and the other touching the gallery's ceiling. The observer will stand in the viewpoint, observe the $\mathbf{N}^2$ - 1 pillars, and marvel.

Cody-Jamal is currently scouting venues trying to see how large he can make the value of **N**. Also, he has not decided which material the pillars will be made of; it could be concrete, or carbon nanotubes, so the radius **R** of the base of each pillar could vary from 1 micrometer to almost half a meter. Notice that a radius of half a meter would make neighboring pillars touch.

You, as a trained mathematician, quickly observe that there could be pillars impossible to see from the viewpoint. Cody-Jamal asks your help in determining, for different combinations of **N** and **R**, the number of visible pillars. Formally, a pillar is visible if and only if there is a straight line segment that runs from the center of the southwest corner cell (the viewpoint) to any point on the pillar's boundary, and does not touch or intersect any other pillar.

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow. Each line describes a different test case with two integers **N** and **R**. **N** is the number of 1 meter square cells along either dimension of the gallery, and **R** is the radius of each pillar, in micrometers. Thus, **R** / 10^6 is the radius of each pillar in meters.

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 pillars in the installment that are visible from the viewpoint.

Limits

$1 \le \mathbf{T} \le 100$.
$1 \le \mathbf{R} < 10^6 / 2$.

Small dataset

$2 \le \mathbf{N} \le 300$.

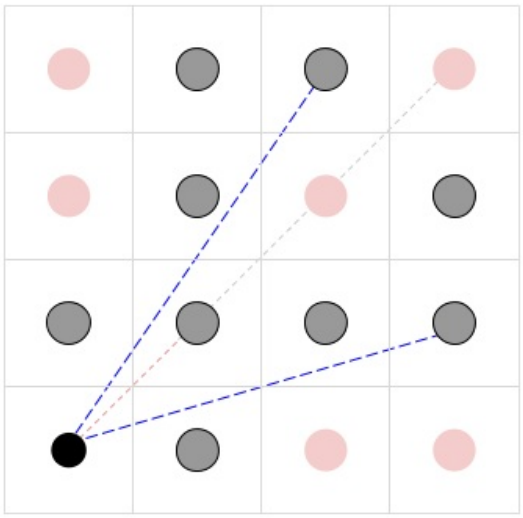
Large dataset

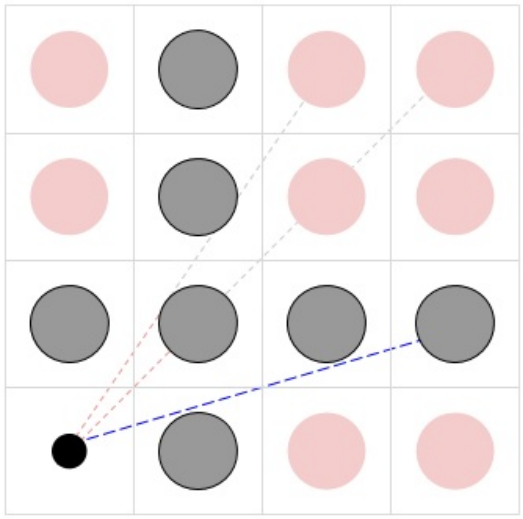
$2 \le \mathbf{N} \le 10^9$.

Sample

Input	Output
4 4 100000 4 300000 3 300000 100 499999	Case #1: 9 Case #2: 7 Case #3: 5 Case #4: 3

The pictures below illustrate the first two samples (not to scale). In the center of the black circle is the observer. The other circles are pillars, with the visible ones in gray and the not visible ones in red. The blue dotted lines represent some of the unblocked lines of sight; the red dotted lines represent blocked lines of sight (that turn gray at the point at which they are first blocked).





Problem D. Map Reduce

Small input 20 points	Solve D-small
Large input 30 points	Solve D-large

Problem

Ben the brilliant video game designer is trying to design maps for his upcoming augmented-reality mobile game. Recently, he has created a map which is represented as a matrix of **R** rows and **C** columns. The map consists of a bunch of . characters representing empty squares, a bunch of # characters representing impassable walls, a single start position represented by S and a single finish position represented by F. For example, the map could look like:

```
#############
#S..#..##...#
###.##..#.#F#
#...##.##.###
#.#.........#
#############
```

In Ben's game, a *path* is a sequence of steps (up, down, left or right) to go from one cell to another while not going through any impassable walls.

Ben considers a *good* map to have the following properties:

- There is a path between any two empty squares (including the start and finish positions).
- To preserve structural integrity, impassable walls must meet at edges and not just corners. For every 2×2 region in the map, if the region contains exactly two walls, then those walls are either in the same row or the same column. In other words, there is no 2×2 region where the walls are in one of these two configurations:

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

- The boundary consists of only impassable walls. A cell is considered part of the boundary if it is in the uppermost/lowermost rows or if it is in the leftmost/rightmost columns.

The distance of the shortest path is the minimum number of steps required to reach the finish position from the start position. For instance, the shortest path in the above example takes 17 steps.

Being such a clever mapmaker, Ben realized that he has created a map that is too hard for his friends to solve. He would like to reduce its difficulty by removing some of the impassable walls. In particular, he wants to know whether it is possible to remove zero or more impassable walls from his map such that the shortest path from start to finish takes *exactly* **D** steps, and that the resulting map is still *good*. Note that it is not enough to simply find *a* path with **D** steps; **D** must be the number of steps in the *shortest* path.

For example, if **D** = 15, we could remove the impassable wall directly below the finish position to get a good solution.

```
#############
#S..#..##...#
###.##..#.#F#
#...##.##.#.#
#.#.........#
#############
```

There is no solution if **D** = 5.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case starts with a line containing three space-separated integers **R**, **C** and **D**: the number of rows and columns in the map, and the desired number of steps in the shortest path from start to finish after possibly removing impassable walls. **R** lines follow, each consisting of **C** characters (either ., #, S or F) representing Ben's map.

It is guaranteed that the map is good, as described in the problem statement.

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 word `POSSIBLE` or `IMPOSSIBLE`, depending on whether the shortest path can be made equal to **D** by removing some number of walls such that the map is still good. If it is possible, output **R** more lines containing **C** characters each, representing the new map. In your output, replace the # characters for removed walls (if any) with `.` characters.

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

Limits

$1 \leq \mathbf{T} \leq 100$.
Each test case contains exactly one S and exactly one F.
The input file is at most 3MB in size.

Small dataset

$3 \leq \mathbf{R} \leq 40$.
$3 \leq \mathbf{C} \leq 40$.
$1 \leq \mathbf{D} \leq 1600$.

Large dataset

$3 \leq \mathbf{R} \leq 1000$.
$3 \leq \mathbf{C} \leq 1000$.
$1 \leq \mathbf{D} \leq 10^6$.

NOTE: The Large output breaks the usual cap on Code Jam output size, but you can upload it as normal.

Sample

Input	Output
3 6 13 15 ############# #S..#..##...# ###.##..#.#F# #...##.##.### #.#.........# ############# 5 8 3 ######## #S.....# ####...# #F.....# ######## 4 10 11 ########## #S#...#.F# #...#...## ##########	Case #1: POSSIBLE ############# #S..#..##...# ###.##..#.#F# #...##.##.#.# #.#.........# ############# Case #2: IMPOSSIBLE Case #3: POSSIBLE ########## #S#...#.F# #...#...## ##########

The sample output displays one set of answers to the sample cases. Other answers may be possible.

Sample case #1 is the example in the problem statement.

In sample case #2, it is possible to remove walls to make the distance of the shortest path either 2 or 4, for example. However, there is no way to make the distance of the shortest path exactly 3.

In sample case #3, the shortest path already takes 11 steps to begin with, so there is no need to reduce the difficulty of the map.

Problem E. Radioactive Islands

Problem

You are steering a boat from the coordinates (-10, **A**) to the coordinates (10, **B**). The coordinates are measured in kilometers, and your boat travels at a constant speed of 1 kilometer per hour. You have full control over the path the boat takes. We model the boat as a single point.

There are **N** islands in the area; we model them as single points. The i-th island is at the coordinates (0, $\mathbf{C_i}$).

The area is radioactive, and you constantly receive 1 microsievert per hour of radiation from the general environment, no matter where you are. Moreover, the islands themselves are radioactive, and you constantly receive additional radiation at a rate of $(D_i)^{-2}$ microsieverts per hour from the i-th island, where D_i is your current distance (in kilometers) from the i-th island. (Formally: let $D_i(t)$ be your distance from the i-th island as a function of time t, and X be the total time your journey takes. Then the total radiation received from the i-th island is the definite integral from 0 to X of $D_i(t)^{-2}$.) You can get as close to an island as you would like, as long as you do not match its exact coordinates.

Find the minimum total radiation dose that you can receive if you plot your course optimally.

Input

The first line of the input gives the number of test cases, **T**; **T** test cases follow. Each test cases consists of two lines. The first line of a test case consists of three values: an integer **N**, and two floating-point numbers **A** and **B**, as described in the statement above. The second line of a test case consists of **N** floating-point numbers $\mathbf{C_i}$; the i-th of these numbers gives the y coordinate of the i-th island.

All floating-point numbers are specified to exactly two decimal places.

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 radiation dose (in microsieverts) received while completing the journey.

y will be considered correct if it is within an absolute or relative error of 10^{-3} of the correct answer. See the FAQ for an explanation of what that means, and what formats of real numbers we accept.

Limits

-10.00 ≤ **A** ≤ 10.00.
-10.00 ≤ **B** ≤ 10.00.
-10.00 ≤ $\mathbf{C_i}$ ≤ 10.00, for all i.
$\mathbf{C_i} \neq \mathbf{C_j}$, for all i ≠ j.

Small dataset

1 ≤ **T** ≤ 20;
N = 1.

Large dataset

1 ≤ **T** ≤ 50;
1 ≤ **N** ≤ 2.

Sample

Input	Output
2 1 1.00 -2.00 0.00 2 0.00 0.00 3.00 -3.00	Case #1: 21.81 Case #2: 21.71

Here is a diagram of the optimal path for sample case #1. We have enlarged the island to make it more visible, but remember to treat it as a single point.