Problem A. Teaching Assistant

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	512 megabytes

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.

- $1 \le T \le 100$
- The length of S is even

Scoring

Small dataset (5pt):

- $2 \le |S| \le 50$

Large dataset (10pt):

- $2 \le |S| \le 20\,000$
- The sum of lengths of all S in the dataset is at most $150\,000$

Example

standard input	standard output

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

Note

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.

Problem B. Forest University

Input file:	`standard input`
Output file:	`standard output`
Time limit:	60 seconds
Memory limit:	512 megabytes

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.

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:

- the number N of courses.
- 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.
- N uppercase English letters (without whitespace in between), with the i-th character giving the first letter of the i-th course's name.
- the number M of cool words.
- M cool words, each of which consists only of uppercase English letters.

- $1 \le T \le 100$
- $1 \le N \le 100$
- $1 \le M \le 5$
- 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

Output

For each test case, output one line containing "`Case #x:` y_1 y_2 `...` y_M", 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.

Scoring

Small dataset (25pt)

Example

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

Note

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:

- take course 1, then course 2, then course 3 (string: "`BAA`")
- take course 1, then course 3, then course 2 (string: "`BAA`")
- 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

Input file:	`standard input`
Output file:	`standard output`
Time limit:	60 seconds
Memory limit:	512 megabytes

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 (x_i, y_i, z_i) in space, and it will move a total distance of (V_{xi}, V_{yi}, 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 (x_i, y_i, z_i) position of the i-th asteroid in space, and the distance (V_{xi}, V_{yi}, V_{zi}) it moves in a single second.

- $1 \le T \le 20$
- $2 \le N \le 1\,000$
- $1 \le S \le 100$
- $-500 \le x_i, y_i, z_i \le 500$

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.

Scoring

Small dataset (8pt):

- $V_{xi} = 0$, $V_{yi} = 0$, $V_{zi} = 0$

Large dataset (17pt):

- $-500 \leq V_{xi}, V_{yi}, V_{zi} \leq 500$

Example

standard input	standard output
`3`	`Case #1: 1.73205080753998`
`3 7`	`Case #2: 1.99999999999992`
`0 0 0 0 0 0`	`Case #3: 3.99999999998743`
`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`	

Note

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++

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	512 megabytes

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.)

- $\underline{\texttt{?0}}\texttt{1?}$, which will print `01`. (Remember that the register is initialized to 0.)
- $\underline{\texttt{?}}\texttt{1}\underline{\texttt{0}}\texttt{?}$, which will print `00`.
- $\underline{\texttt{?}}\texttt{1?}\underline{\texttt{0}}$, which will print `01`.
- $\texttt{1}\underline{\texttt{?0}}\texttt{?}$, which will print `10`.
- $\texttt{1}\underline{\texttt{?}}\texttt{?}\underline{\texttt{0}}$, which will print `11`.
- $\texttt{1?}\underline{\texttt{?0}}$, which will print `11`.

Note that the output string always consists of 0s and 1s, 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 $B = \texttt{11}$ and $G = \{\texttt{10}, \texttt{00}\}$, 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 0s and/or 1s.

- $1 \le T \le 100$
- $1 \le N \le 100$
- $1 \le L \le 50$
- All strings in G are different

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.

Scoring

Small dataset (7pt):

- B consists entirely of 1s

Large dataset (28pt):

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

Example

standard input	standard 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

Note

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.