#1

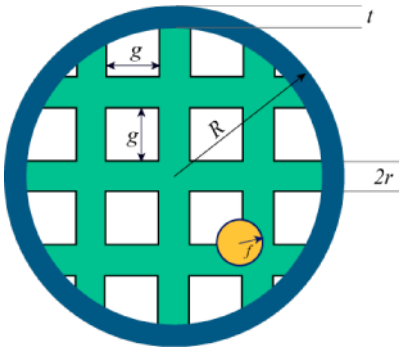
Problem

What are your chances of hitting a fly with a tennis racquet?

To start with, ignore the racquet's handle. Assume the racquet is a perfect ring, of outer radius **R** and thickness **t** (so the inner radius of the ring is **R**−**t**).

The ring is covered with horizontal and vertical strings. Each string is a cylinder of radius **r**. Each string is a chord of the ring (a straight line connecting two points of the circle). There is a gap of length **g** between neighbouring strings. The strings are symmetric with respect to the center of the racquet i.e. there is a pair of strings whose centers meet at the center of the ring.

The fly is a sphere of radius **f**. Assume that the racquet is moving in a straight line perpendicular to the plane of the ring. Assume also that the fly's center is inside the outer radius of the racquet and is equally likely to be anywhere within that radius. Any overlap between the fly and the racquet (the ring or a string) counts as a hit.



Input

One line containing an integer **N**, the number of test cases in the input file.

The next **N** lines will each contain the numbers **f**, **R**, **t**, **r** and **g** separated by exactly one space. Also the numbers will have exactly 6 digits after the decimal point.

Output

N lines, each of the form "Case #**k**: **P**", where **k** is the number of the test case and **P** is the probability of hitting the fly with a piece of the racquet.

Answers with a relative or absolute error of at most 10^{-6} will be considered correct.

Limits

Time limit: 60 seconds per test set.

Memory limit: 1GB.

f, **R**, **t**, **r** and **g** will be positive and smaller or equal to 10000.

t < **R**

f < **R**

r < **R**

Small dataset (Test set 1 - Visible)

1 ≤ **N** ≤ 30

The total number of strings will be at most 60 (so at most 30 in each direction).

Large dataset (Test set 2 - Hidden)

1 ≤ **N** ≤ 100

The total number of strings will be at most 2000 (so at most 1000 in each direction).

Sample

Input

```
5
0.250000 1.000000 0.100000 0.010000 0.500000
0.250000 1.000000 0.100000 0.010000 0.900000
0.000010 10000.000000 0.000010 0.000010 1000.000000
0.400000 10000.000000 0.000010 0.000010 700.000000
1.000000 100.000000 1.000000 1.000000 10.000000
```

Output

```
Case #1: 1.000000
Case #2: 0.910015
Case #3: 0.000000
Case #4: 0.002371
Case #5: 0.573972
```

#2

Problem

The urban legend goes that if you go to the Google homepage and search for "Google", the universe will implode. We have a secret to share... It is true! Please don't try it, or tell anyone. All right, maybe not. We are just kidding.

The same is not true for a universe far far away. In that universe, if you search on any search engine for that search engine's name, the universe does implode!

To combat this, people came up with an interesting solution. All queries are pooled together. They are passed to a central system that decides which query goes to which search engine. The central system sends a series of queries to one search engine, and can switch to another at any time. Queries must be processed in the order they're received. The central system must never send a query to a

search engine whose name matches the query. In order to reduce costs, the number of switches should be minimized.

Your task is to tell us how many times the central system will have to switch between search engines, assuming that we program it optimally.

Input

The first line of the input file contains the number of cases, **N**. **N** test cases follow.

Each case starts with the number **S** -- the number of search engines. The next **S** lines each contain the name of a search engine. Each search engine name is no more than one hundred characters long and contains only uppercase letters, lowercase letters, spaces, and numbers. There will not be two search engines with the same name.

The following line contains a number **Q** -- the number of incoming queries. The next **Q** lines will each contain a query. Each query will be the name of a search engine in the case.

Output

For each input case, you should output:

`Case #X: Y`

where X is the number of the test case and Y is the number of search engine switches. Do not count the initial choice of a search engine as a switch.

Limits

Time limit: 30 seconds per test set.

Memory limit: 1GB.

$0 < \mathbf{N} \leq 20$

Small dataset (Test set 1 - Visible)

$2 \leq \mathbf{S} \leq 10$

$0 \leq \mathbf{Q} \leq 100$

Large dataset (Test set 2 - Hidden)

$2 \leq \mathbf{S} \leq 100$

$0 \leq \mathbf{Q} \leq 1000$

Sample

Input	Output
2 5 Yeehaw NSM	Case #1: 1 Case #2: 0

```
Dont Ask
B9
Googol
10
Yeehaw
Yeehaw
Googol
B9
Googol
NSM
B9
NSM
Dont Ask
Googol
5
Yeehaw
NSM
Dont Ask
B9
Googol
7
Googol
Dont Ask
NSM
NSM
Yeehaw
Yeehaw
Googol
```

In the first case, one possible solution is to start by using Dont Ask, and switch to NSM after query number 8.

For the second case, you can use B9, and not need to make any switches.

#3

Problem

A train line has two stations on it, A and B. Trains can take trips from A to B or from B to A multiple times during a day. When a train arrives at B from A (or arrives at A from B), it needs a certain amount of time before it is ready to take the return journey - this is the *turnaround time*. For example, if a train arrives at 12:00 and the turnaround time is 0 minutes, it can leave immediately, at 12:00.

A train timetable specifies departure and arrival time of all trips between A and B. The train company needs to know how many trains have to start the day at A and B in order to make the timetable work: whenever a train is supposed to leave A or B, there must actually be one there ready to go. There are passing sections on the track, so trains don't necessarily arrive in the same order that they leave. Trains may not travel on trips that do not appear on the schedule.

Input

The first line of input gives the number of cases, N. N test cases follow.

Each case contains a number of lines. The first line is the turnaround time, **T**, in minutes. The next

line has two numbers on it, **NA** and **NB**. **NA** is the number of trips from A to B, and **NB** is the number of trips from B to A. Then there are **NA** lines giving the details of the trips from A to B. Each line contains two fields, giving the HH:MM departure and arrival time for that trip. The departure time for each trip will be earlier than the arrival time. All arrivals and departures occur on the same day. The trips may appear in any order - they are not necessarily sorted by time. The hour and minute values are both two digits, zero-padded, and are on a 24-hour clock (00:00 through 23:59).

After these **NA** lines, there are **NB** lines giving the departure and arrival times for the trips from B to A.

Output

For each test case, output one line containing "Case #**x**: " followed by the number of trains that must start at A and the number of trains that must start at B.

Limits

Time limit: 30 seconds per test set.

Memory limit: 1GB.

Small dataset (Test set 1 - Visible)

$1 \leq \mathbf{N} \leq 20$

$0 \leq \mathbf{NA}, \mathbf{NB} \leq 20$

$0 \leq \mathbf{T} \leq 5$

Large dataset (Test set 2 - Hidden)

$1 \leq \mathbf{N} \leq 100$

$0 \leq \mathbf{NA}, \mathbf{NB} \leq 100$

$0 \leq \mathbf{T} \leq 60$

Sample

Input Output

```
2
5
3 2
09:00 12:00
10:00 13:00
11:00 12:30
12:02 15:00
09:00 10:30
2
2 0
09:00 09:01
```

12:00 12:02

#4

Problem

You are given two vectors $v_1 = (x_1, x_2, ..., x_n)$ and $v_2 = (y_1, y_2, ..., y_n)$. The scalar product of these vectors is a single number, calculated as $x_1y_1 + x_2y_2 + ... + x_ny_n$.

Suppose you are allowed to permute the coordinates of each vector as you wish. Choose two permutations such that the scalar product of your two new vectors is the smallest possible, and output that minimum scalar product.

Input

The first line of the input file contains integer number T - the number of test cases. For each test case, the first line contains integer number n. The next two lines contain n integers each, giving the coordinates of v_1 and v_2 respectively.

Output

For each test case, output a line

`Case #X: Y`

where X is the test case number, starting from 1, and Y is the minimum scalar product of all permutations of the two given vectors.

Limits

Time limit: 30 seconds per test set.

Memory limit: 1GB.

Small dataset (Test set 1 - Visible)

T = 1000

$1 \le \mathbf{n} \le 8$

$-1000 \le \mathbf{x}_i, \mathbf{y}_i \le 1000$

Large dataset (Test set 2 - Hidden)

T = 10

$100 \le \mathbf{n} \le 800$

$-100000 \le \mathbf{x}_i, \mathbf{y}_i \le 100000$

Sample

Input	Output
2 3	

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

#5

Problem

You own a milkshake shop. There are N different flavors that you can prepare, and each flavor can be prepared "malted" or "unmalted". So, you can make 2N different types of milkshakes.

Each of your customers has a set of milkshake types that they like, and they will be satisfied if you have at least one of those types prepared. At most one of the types a customer likes will be a "malted" flavor.

You want to make N batches of milkshakes, so that:

- There is exactly one batch for each flavor of milkshake, and it is either malted or unmalted.
- For each customer, you make at least one milkshake type that they like.
- The minimum possible number of batches are malted.

Find whether it is possible to satisfy all your customers given these constraints, and if it is, what milkshake types you should make.

If it is possible to satisfy all your customers, there will be only one answer which minimizes the number of malted batches.

Input

One line containing an integer **C**, the number of test cases in the input file.

For each test case, there will be:

- One line containing the integer **N**, the number of milkshake flavors.
- One line containing the integer **M**, the number of customers.
- **M** lines, one for each customer, each containing:
- An integer **T** ≥ 1, the number of milkshake types the customer likes, followed by
- **T** pairs of integers "**X Y**", one for each type the customer likes, where **X** is the milkshake flavor between **1** and **N** inclusive, and **Y** is either 0 to indicate unmalted, or 1 to indicated malted. Note that:
- No pair will occur more than once for a single customer.
- Each customer will have at least one flavor that they like (T ≥ 1).
- Each customer will like at most one malted flavor. (At most one pair for each customer has Y = 1).

All of these numbers are separated by single spaces.

Output

C lines, one for each test case in the order they occur in the input file, each containing the string "Case #**X**: " where **X** is the number of the test case, starting from 1, followed by:

- The string "**IMPOSSIBLE**", if the customers' preferences cannot be satisfied; **OR**
- **N** space-separated integers, one for each flavor from **1** to **N**, which are 0 if the corresponding flavor should be prepared unmalted, and 1 if it should be malted.

Limits

Time limit: 30 seconds per test set.

Memory limit: 1GB.

Small dataset (Test set 1 - Visible)

C = 100

$1 \leq N \leq 10$

$1 \leq M \leq 100$

Large dataset (Test set 2 - Hidden)

C = 5

$1 \leq N \leq 2000$

$1 \leq M \leq 2000$

The sum of all the **T** values for the customers in a test case will not exceed 3000.

Sample

Input Output

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

In the first case, you must make flavor #1 malted, to satisfy the first customer. Every other flavor can be unmalted. The second customer is satisfied by getting flavor #2 unmalted, and the third customer is satisfied by getting flavor #5 unmalted.

In the second case, there is only one flavor. One of your customers wants it malted and one wants it unmalted. You cannot satisfy them both

#6

Problem

In this problem, you have to find the last three digits before the decimal point for the number $(3 + \sqrt{5})^{\mathbf{n}}$.

For example, when **n** = 5, $(3 + \sqrt{5})^5 = 3935.73982...$ The answer is 935.

For **n** = 2, $(3 + \sqrt{5})^2 = 27.4164079...$ The answer is 027.

Input

The first line of input gives the number of cases, **T**. **T** test cases follow, each on a separate line.

Each test case contains one positive integer **n**.

Output

For each input case, you should output:

`Case #X: Y`

where X is the number of the test case and Y is the last three integer digits of the number $(3 + \sqrt{5})^{\mathbf{n}}$. In case that number has fewer than three integer digits, add leading zeros so that your output contains exactly three digits.

Limits

Time limit: 30 seconds per test set.

Memory limit: 1GB.

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

Small dataset (Test set 1 - Visible)

$2 \leq \mathbf{n} \leq 30$

Large dataset (Test set 2 - Hidden)

$2 \leq \mathbf{n} \leq 2000000000$

Sample

Input Output

```
2
5
2     Case #1: 935
      Case #2: 027
```

#7

Problem

Some pranksters have watched too much Discovery Channel and now they want to build a crop triangle during the night. They want to build it inside a large crop that looks like an evenly spaced grid from above. There are some trees planted on the field. Each tree is situated on an intersection of two grid lines (a grid point). The pranksters want the vertices of their crop triangle to be located at these trees. Also, for their crop triangle to be more interesting they want the *center* of that triangle to be located at some grid point as well. We remind you that if a triangle has the vertices (x_1, y_1), (x_2, y_2) and (x_3, y_3), then the center for this triangle will have the coordinates $((x_1 + x_2 + x_3) / 3, (y_1 + y_2 + y_3) / 3)$.

You will be given a set of points with integer coordinates giving the location of all the trees on the grid. You are asked to compute how many triangles you can form with **distinct** vertexes in this set of points so that their center is a grid point as well (i.e. the center has integer coordinates).

If a triangle has area 0 we will still consider it a valid triangle.

Input

The first line of input gives the number of cases, **N**. **N** test cases follow. Each test case consists of one line containing the integers **n**, **A**, **B**, **C**, **D**, $\mathbf{x_0}$, $\mathbf{y_0}$ and **M** separated by exactly one space. **n** will be the number of trees in the input set. Using the numbers **n**, **A**, **B**, **C**, **D**, $\mathbf{x_0}$, $\mathbf{y_0}$ and **M** the following pseudocode will print the coordinates of the trees in the input set. *mod* indicates the remainder operation.

The parameters will be chosen such that the input set of trees will not have duplicates.

```
X = x0, Y = y0
print X, Y
for i = 1 to n-1
  X = (A * X + B) mod M
  Y = (C * Y + D) mod M
  print X, Y
```

Output

For each test case, output one line containing "Case #**X**: " where **X** is the test case number (starting from 1). This should be followed by an integer indicating the number of triangles which can be located at 3 distinct trees and has a center that is a grid point.

Limits

Time limit: 30 seconds per test set.

Memory limit: 1GB.

$1 \leq \mathbf{N} \leq 10$,

$0 \le \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \mathbf{x_0}, \mathbf{y_0} \le 10^9$,

$1 \le \mathbf{M} \le 10^9$.

Small dataset (Test set 1 - Visible)

$3 \le \mathbf{n} \le 100$.

Large dataset (Test set 2 - Hidden)

$3 \le \mathbf{n} \le 100000$.

Sample

Input	Output
2 4 10 7 1 2 0 1 20 6 2 0 2 1 1 2 11	Case #1: 1 Case #2: 2

In the first test case, the 4 trees in the generated input set are (0, 1), (7, 3), (17, 5), (17, 7).

#8

Problem

You start with a sequence of consecutive integers. You want to group them into sets.
You are given the interval, and an integer **P**. Initially, each number in the interval is in its own set.
Then you consider each pair of integers in the interval. If the two integers share a prime factor which is at least **P**, then you merge the two sets to which the two integers belong.
How many different sets there will be at the end of this process?

Input

One line containing an integer **C**, the number of test cases in the input file.
For each test case, there will be one line containing three single-space-separated integers **A**, **B**, and **P**. **A** and **B** are the first and last integers in the interval, and **P** is the number as described above.

Output

For each test case, output one line containing the string "Case #X: Y" where X is the number of the test case, starting from 1, and Y is the number of sets.

Limits

Time limit: 30 seconds per test set.
Memory limit: 1GB.

Small dataset (Test set 1 - Visible)

$1 \le \mathbf{C} \le 10$

$1 \le \mathbf{A} \le \mathbf{B} \le 1000$

$2 \le \mathbf{P} \le \mathbf{B}$

Large dataset (Test set 2 - Hidden)

$1 \le \mathbf{C} \le 100$

$1 \le \mathbf{A} \le \mathbf{B} \le 10^{12}$

$\mathbf{B} \le \mathbf{A} + 1000000$

$2 \le \mathbf{P} \le \mathbf{B}$

Sample

Input	Output
2 10 20 5 10 20 3	Case #1: 9 Case #2:

#9

Problem

Mousetrap is a simple card game for one player. It is played with a shuffled deck of cards numbered 1 through **K**, face down. You play by revealing the top card of the deck and then putting it on the bottom of the deck, keeping count of how many cards you have revealed. If you reveal a card whose number matches the current count, remove it from the deck and reset the count. If the count ever reaches **K**+1, you have lost. If the deck runs out of cards, you win.

Suppose you have a deck of 5 cards, in the order 2, 5, 3, 1, 4. You will reveal the 2 on count 1, the 5 on count 2, then the 3 on count 3. Since the value matches the count, you remove the 3 from the deck, and reset the count. You now have 4 cards left in the order 1, 4, 2, 5. You then reveal the 1 on count 1, and remove it as well (you're doing great so far!). Continuing in this way you will remove the 2, then the 4, and then finally the 5 for victory.

You would like to set up a deck of cards in such a way that you will win the game and remove the cards in increasing order. We'll call a deck organized in this way "perfect." For example, with 4 cards you can organize the deck as 1, 4, 2, 3, and you will win by removing the cards in the order 1, 2, 3, 4.

Input

The first line of input gives the number of cases, **T**. Each test case starts with a line containing **K**, the number of cards in a deck. The next line starts with an integer **n**, which is followed by **n** integers (**d**$_1$,**d**$_2$, ...), indices into the deck.

Output

For each test case, output one line containing "Case #**x**: " followed by **n** integers (**k**$_1$,**k**$_2$, ...), where **k**$_i$ is the value of the card at index **d**$_i$ of a perfect deck of size **K**. The numbers in the output should be separated by spaces, and there must be at least one space following the colon in each "Case #**x**:" line.

Limits

Memory limit: 1GB.

Small dataset (Test set 1 - Visible)

Time limit: 60 seconds.

T = 100,

$1 \le \mathbf{K} \le 5000$,

$1 \le \mathbf{n} \le 100$,

$1 \le \mathbf{d}_i \le \mathbf{K}$.

Large dataset (Test set 2 - Hidden)

Time limit: 180 seconds.

T = 10,

$1 \le \mathbf{K} \le 1000000$,

$1 \le \mathbf{n} \le 100$,

$1 \le \mathbf{d}_i \le \mathbf{K}$.

Sample

Input	Output
2 5 5 1 2 3 4 5 15 4 3 4 7 10	Case #1: 1 3 2 5 4 Case #2: 2 8 13 4

#10

The story

Professor Loony, a dear friend of mine, stormed into my office. His face was red and he looked very angry. The first thing that came out of his mouth was "Damn those phone manufacturers. I was trying to send a text message, and it took me more than ten minutes to type a one-line message." I tried to calm him down. "But what is wrong? Why did it take you so long?" He continued, "Don't you see?! Their placement of the letters is so messed up? Why is 's' the 4th letter on its key? and 'e'? Why is it not the first letter on its key? I have to press '7' FOUR times to type an 's'? This is lunacy!"
"Calm down, my friend," I said, "This scheme has been in use for so long, even before text messaging was invented. They had to keep it that way."
"That's not an excuse," his face growing redder and redder. "It is time to change all this. It was a stupid idea to start with. And while we are at it, how come they only put letters on 8 keys? Why not use all 12? And why do they have to be consecutive?"
"Umm... I... don't... know," I replied.
"Ok, that's it. Those people are clearly incompetent. I am sure someone can come up with a better scheme."
He was one of *those* people, I could see. People who complain about the problem, but never actually try to solve it.
In this problem, you are required to come up with the best letter placement of keys to minimize the number of key presses required to type a message. You will be given the number of keys, the maximum number of letters we can put on every key, the total number of letters in the alphabet, and the frequency of every letter in the message. Letters can be placed anywhere on the keys and in any order. Each letter can only appear on one key. Also, the alphabet can have more than 26 letters (it is not English).
For reference, the current phone keypad looks like this

```
key 2: abc
key 3: def
key 4: ghi
key 5: jkl
key 6: mno
key 7: pqrs
key 8: tuv
key 9: wxyz
```

The first press of a key types the first letter. Each subsequent press advances to the next letter. For example, to type the word "snow", you need to press "7" four times, followed by "6" twice, followed by "6" three times, followed by "9" once. The total number of key presses is 10.

Input

The first line in the input file contains the number of test cases **N**. This is followed by **N** cases. Each case consists of two lines. On the first line we have the maximum number of letters to place on a key (**P**), the number of keys available (**K**) and the number of letters in our alphabet (**L**) all separated by single spaces. The second line has **L** non-negative integers. Each number represents the frequency of a certain letter. The first number is how many times the first letter is used, the second number is how many times the second letter is used, and so on.

Output

For each case, you should output the following

```
Case #x: [minimum number of keypad presses]
```

indicating the number of keypad presses to type the message for the optimal layout.

Limits

Time limit: 30 seconds per test set.

Memory limit: 1GB.

P * **K** ≥ **L**

0 ≤ The frequency of each letter ≤ 1000000

Small dataset (Test set 1 - Visible)

1 ≤ **N** ≤ 10

1 ≤ **P** ≤ 10

1 ≤ **K** ≤ 12

1 ≤ **L** ≤ 100

Large dataset (Test set 2 - Hidden)

1 ≤ **N** ≤ 100

1 ≤ **P** ≤ 1 000

1 ≤ **K** ≤ 1 000

1 ≤ **L** ≤ 1 000

Sample

Input

```
2
3 2 6
8 2 5 2 4 9
3 9 26
1 1 1 100 100 1 1 1 1 1 1 1 1 1 1 1 10 11 11 11 11 1 1 1 100
```

#11

Problem

Once upon a time in a strange situation, people called a number *ugly* if it was divisible by any of the one-digit primes (2, 3, 5 or 7). Thus, 14 is ugly, but 13 is fine. 39 is ugly, but 121 is not. Note that 0 is ugly. Also note that negative numbers can also be ugly; -14 and -39 are examples of such numbers.

One day on your free time, you are gazing at a string of digits, something like:

```
123456
```

You are amused by how many possibilities there are if you are allowed to insert *plus* or *minus* signs between the digits. For example you can make

```
1 + 234 - 5 + 6 = 236
```

which is ugly. Or

```
123 + 4 - 56 = 71
```

which is not ugly.

It is easy to count the number of different ways you can play with the digits: Between each two adjacent digits you may choose put a plus sign, a minus sign, or nothing. Therefore, if you start with D digits there are 3^{D-1} expressions you can make.

Note that it is fine to have leading zeros for a number. If the string is "01023", then "01023", "0+1-02+3" and "01-023" are legal expressions.

Your task is simple: Among the 3^{D-1} expressions, count how many of them evaluate to an ugly number.

Input

The first line of the input file contains the number of cases, **N**. Each test case will be a single line containing a non-empty string of decimal digits.

Output

For each test case, you should output a line

```
Case #X: Y
```

where **X** is the case number, starting from 1, and **Y** is the number of expressions that evaluate to an ugly number.

Limits

Time limit: 30 seconds per test set.

Memory limit: 1GB.

$0 \leq \mathbf{N} \leq 100$.

The string in each test case will be non-empty and will contain only characters '0' through '9'.

Small dataset (Test set 1 - Visible)

Each string is no more than 13 characters long.

Large dataset (Test set 2 - Hidden)

Each string is no more than 40 characters long.

Sample

Input Output

```
4
1
9
011
12345
```

```
Case #1: 0
Case #2: 1
Case #3: 6
Case #4: 64
```

#12

Problem

You were driving along a highway when you got caught by the road police for speeding. It turns out that they've been following you, and they were amazed by the fact that you were accelerating the whole time without using the brakes! And now you desperately need an excuse to explain that. You've decided that it would be reasonable to say "all the speed limit signs I saw were in increasing order, that's why I've been accelerating". The police officer laughs in reply, and tells you all the signs that are placed along the segment of highway you drove, and says that's unlikely that you were so lucky just to see some part of these signs that were in increasing order.

Now you need to estimate that likelihood, or, in other words, find out how many different subsequences of the given sequence are strictly increasing. The empty subsequence does not count since that would imply you didn't look at any speed limits signs at all!

For example, (1, 2, 5) is an increasing subsequence of (1, 4, 2, 3, 5, 5), and we count it twice because there are two ways to select (1, 2, 5) from the list.

Input

The first line of input gives the number of cases, **N**. **N** test cases follow. The first line of each case contains **n**, **m**, **X**, **Y** and **Z** each separated by a space. **n** will be the length of the sequence of speed limits. **m** will be the length of the generating array A. The next **m** lines will contain the **m** elements of A, one integer per line (from A[0] to A[**m**-1]).

Using A, **X**, **Y** and **Z**, the following pseudocode will *print* the speed limit sequence in order. mod indicates the remainder operation.

```
for i = 0 to n-1
  print A[i mod m]
  A[i mod m] = (X * A[i mod m] + Y * (i + 1)) mod Z
```

Note: The way that the input is generated has nothing to do with the intended solution and exists solely to keep the size of the input files low.

Output

For each test case you should output one line containing "Case #**T**: **S**" (quotes for clarity)
where **T** is the number of the test case and **S** is the number of non-empty increasing subsequences mod 1 000 000 007.

Limits

Time limit: 60 seconds per test set.

Memory limit: 1GB.

$1 \leq \mathbf{N} \leq 20$

$1 \leq \mathbf{m} \leq 100$

$0 \leq \mathbf{X} \leq 10^9$

$0 \leq \mathbf{Y} \leq 10^9$

$1 \leq \mathbf{Z} \leq 10^9$

$0 \leq \mathbf{A[i]} < \mathbf{Z}$

Small dataset (Test set 1 - Visible)

$1 \leq \mathbf{m} \leq \mathbf{n} \leq 1000$

Large dataset (Test set 2 - Hidden)

$1 \leq \mathbf{m} \leq \mathbf{n} \leq 500000$

Sample

Input	Output
2 5 5 0 0 5 1 2 1 2 3 6 2 2 1000000000 6 1 2	Case #1: 15 Case #2: 13

The sequence of speed limit signs for case 2 should be 1, 2, 0, 0, 0, 4.

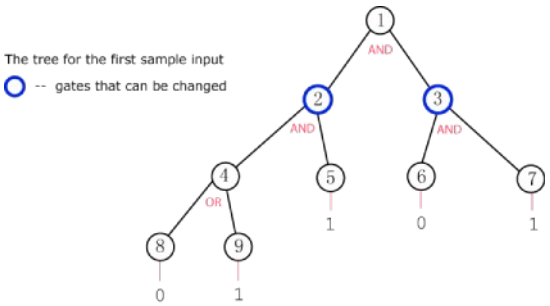
#13

Problem

For this problem we will consider a type of binary tree that we will call a boolean tree. In this tree, every row is completely filled, except possibly the last (deepest) row, and the nodes in the last row are as far to the left as possible. Additionally, every node in the tree will either have 0 or 2 children. What makes a boolean tree special is that each node has a boolean value associated with it, 1 or 0. In addition, each interior node has either an "AND" or an "OR" gate associated with it. The value of an "AND" gate node is given by the logical AND of its two children's values. The value of an "OR" gate likewise is given by the logical OR of its two children's values. The value of all of the leaf nodes will be given as input so that the value of all nodes can be calculated up the tree.
The root of the tree is of particular interest to us. We would really like for the root to have the value **V**, either 1 or 0. Unfortunately, this may not be the value the root actually has. Luckily for us, we can cheat and change the type of gate for some of the nodes; we can change an AND gate to an OR gate or an OR gate to an AND gate.
Given a description of a boolean tree and what gates can be changed, find the minimum number of gates that need to be changed to make the value of the root node **V**. If this is impossible, output "IMPOSSIBLE" (quotes for clarity).

Input

The first line of the input file contains the number of cases, **N**. **N** test cases follow.
Each case begins with **M** and **V**. **M** represents the number of nodes in the tree and will be odd to ensure all nodes have 0 or 2 children. **V** is the desired value for the root node, 0 or 1.
M lines follow describing each of the tree's nodes. The X_{th} line will describe node X, starting with node 1 on the first line.
The first (**M**−1)/2 lines describe the interior nodes. Each line contains **G** and **C**, each being either 0 or 1. If **G** is 1 then the gate for this node is an AND gate, otherwise it is an OR gate. If **C** is 1 then the gate for this node is changeable, otherwise it is not. Interior node X has nodes 2X and 2X+1 as children.
The next (**M**+1)/2 lines describe the leaf nodes. Each line contains one value **I**, 0 or 1, the value of the leaf node.
To help visualize, here is a picture of the tree in the first sample input.



Output

For each test case, you should output:

`Case #X: Y`

where **X** is the number of the test case and **Y** is the minimum number of gates that must be changed to make the output of the root node **V**, or "IMPOSSIBLE" (quotes for clarity) if this is impossible.

Limits

Time limit: 30 seconds per test set.

Memory limit: 1GB.

$1 < \mathbf{N} \leq 20$

Small dataset (Test set 1 - Visible)

$2 < \mathbf{M} < 30$

Large dataset (Test set 2 - Hidden)

$2 < \mathbf{M} < 10000$

Sample

Input Output

```
2
9 1
1 0
1 1
1 1
0 0
1
0
1
0
1
5 0
1 1
0 0
1
1
0
```

```
Case #1: 1
Case #2: IMPOSSIBLE
```

In case 1, we can change the gate on node 3 to an OR gate to achieve the desired result at the root.
In case 2, only the root can be changed but changing it to an OR gate does not help.

#14

Problem

Ten-year-old Tangor has just discovered how to compute the area of a triangle. Being a bright boy, he is amazed by how many different ways one can compute the area. He also convinced himself that, if all the points of the triangle have integer coordinates, then the triangle's area is always either an integer or half of an integer! Isn't that nice?
But today Tangor is trying to go in the opposite direction. Instead of taking a triangle and computing its area, he is taking an integer **A** and trying to draw a triangle of area **A**/2. He restricts himself to using only the integer points on his graph paper for the triangle's vertices.
More precisely, the sheet of graph paper is divided into an **N** by **M** grid of square cells. The triangle's vertices may only be placed in the corners of those cells. If you imagine a coordinate system on the paper, then these points are of the form (**x**, **y**), where **x** and **y** are integers such that 0 ≤ **x** ≤ **N** and 0 ≤ **y** ≤ **M**.
Given the integer **A**, help Tangor find three integer points on the sheet of graph paper such that the area of the triangle formed by those points is exactly **A**/2, if that is possible. In case there is more than one way to do this, any solution will make him happy.

Input

One line containing an integer **C**, the number of test cases in the input file.
The next **C** lines will each contain three integers **N**, **M**, and **A**, as described above.

Output

For each test case, output one line. If there is no way to satisfy the condition, output

```
Case #k: IMPOSSIBLE
```

where k is the case number, starting from 1. Otherwise, output

```
Case #k: x1 y1 x2 y2 x3 y3
```

where k is the case number and (x_1, y_1), (x_2, y_2), (x_3, y_3) are any three integer points on the graph paper that form a triangle of area **A**/2.

Limits

Time limit: 30 seconds per test set.
Memory limit: 1GB.

$0 \le \mathbf{C} \le 1000$

$1 \le \mathbf{A} \le 10^8$

Small dataset (Test set 1 - Visible)

$1 \le \mathbf{N} \le 50$

$1 \le \mathbf{M} \le 50$

Large dataset (Test set 2 - Hidden)

$1 \le \mathbf{N} \le 10000$

$1 \le \mathbf{M} \le 10000$

Sample

Input Output

```
3
1 1 1
1 2 64
10 10 1
```

```
Case #1: 0 0 0 1 1 1
Case #2: IMPOSSIBLE
Case #3: 1 1 2 3 5 8
```

#15

Problem

Near the planet Mars, in a faraway galaxy eerily similar to our own, there is a fight to the death between the imperial forces and the rebels. The rebel army has **N** ships which we will consider as points (x_i, y_i, z_i). Each ship has a receiver with power $\mathbf{p_i}$. The rebel army needs to be able to send messages from the central cruiser to all the ships, but they are tight on finances, so they cannot afford a strong transmitter.

If the cruiser is placed at (x, y, z), and one of the other ships is at (x_i, y_i, z_i) and has a receiver of power $\mathbf{p_i}$, then the power of the cruiser's transmitter needs to be at least:

$$(|x_i - x| + |y_i - y| + |z_i - z|) / p_i$$

Your task is to find the position for the cruiser that minimizes the power required for its transmitter, and to output that power.

Input

The first line of input gives the number of cases, **T**. **T** test cases follow.

Each test case contains on the first line the integer **N**, the number of ships in the test case.

N lines follow, each line containing four integer numbers $\mathbf{x_i}$, $\mathbf{y_i}$, $\mathbf{z_i}$ and $\mathbf{p_i}$, separated by single

spaces. These are the coordinates of the i-th ship, and the power of its receiver. There may be more than one ship at the same coordinates.

Output

For each input case, you should output:

```
Case #X: Y
```

where **X** is the number of the test case and **Y** is the minimal power that is enough to reach all the fleet's ships. Answers with a relative or absolute error of at most 10^{-6} will be considered correct.

Limits

Time limit: 30 seconds per test set.

Memory limit: 1GB.

$1 \le \mathbf{T} \le 10$

$0 \le \mathbf{x_i}, \mathbf{y_i}, \mathbf{z_i} \le 10^6$

$1 \le \mathbf{p_i} \le 10^6$

Small dataset (Test set 1 - Visible)

$1 \le \mathbf{N} \le 10$

Large dataset (Test set 2 - Hidden)

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

Sample

Input

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

Output

```
Case #1: 3.50000000
Case #2: 0.00000000
Case #3: 2.33333333
```

In the first test case, the four ships have coordinates (0, 0, 0), (1, 2, 0), (3, 4, 0), (2, 1, 0) and powers 1, 1, 1, 1 respectively. We can place a cruiser with the power 3.5 at the coordinates (1.5, 2, 0) which will be able to reach all the ships.

In the second case we can place the cruiser right on top of the ship, with transmitter power 0.

#16

Problem

You've invented a slight modification of the run-length encoding (RLE) compression algorithm, called PermRLE.

To compress a string, this algorithm chooses some permutation of integers between 1 and **k**, applies this permutation to the first **k** letters of the given string, then to the next block of **k** letters, and so on. The length of the string must be divisible by **k**. After permuting all blocks, the new string is compressed using RLE, which is described later.

To apply the given permutation *p* to a block of **k** letters means to place the *p[1]*-th of these letters in the first position, then *p[2]*-th of these letters in the second position, and so on. For example, applying the permutation {3,1,4,2} to the block "abcd" yields "cadb". Applying it to the longer string "abcdefghijkl" in blocks yields "cadbgehfkilj".

The permuted string is then compressed using run-length encoding. To simplify, we will consider the *compressed size* of the string to be the number of groups of consecutive equal letters. For example, the compressed size of "aabcaaaa" is 4; the first of the four groups is a group of two letters "a", then two groups "b" and "c" each containing only one letter, and finally a longer group of letters "a".

Obviously, the compressed size may depend on the chosen permutation. Since the goal of compression algorithms is to minimize the size of the compressed text, it is your job to choose the permutation that yields the smallest possible compressed size, and output that size.

Input

The first line of input gives the number of cases, **N**. **N** test cases follow.

The first line of each case will contain **k**. The second line will contain **S**, the string to be compressed.

Output

For each test case you should output one line containing "Case #**X**: **Y**" (quotes for clarity) where **X** is the number of the test case and **Y** is the minimum compressed size of **S**.

Limits

Time limit: 30 seconds per test set.

Memory limit: 1GB.

N = 20

S will contain only lowercase letters 'a' through 'z'

The length of **S** will be divisible by **k**

Small dataset (Test set 1 - Visible)

$2 \le \mathbf{k} \le 5$

$1 \le$ length of $\mathbf{S} \le 1000$

Large dataset (Test set 2 - Hidden)

$2 \le \mathbf{k} \le 16$

$1 \le$ length of $\mathbf{S} \le 50000$

Sample

Input	Output
2 4 abcabcabcabc 3 abcabcabcabc	Case #1: 7 Case #2: 12

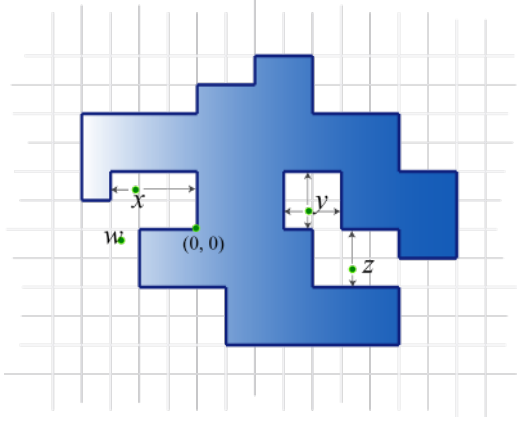
#17

Problem

Professor Polygonovich, an honest citizen of Flatland, likes to take random walks along integer points in the plane. He starts from the origin in the morning, facing north. There are three types of actions he makes:

- 'F': move forward one unit of length.
- 'L': turn left 90 degrees.
- 'R': turn right 90 degrees.

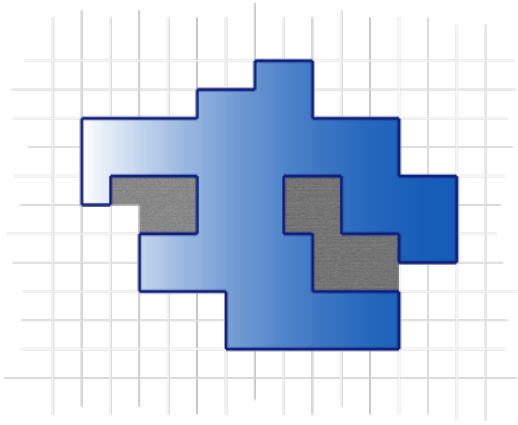
At the end of the day (yes, it is a long walk!), he returns to the origin. He never visits the same point twice except for the origin, so his path encloses a polygon. In the following picture the interior of the polygon is colored blue (ignore the points x, y, z, and w for now; they will be explained soon):



Notice that as long as Professor Polygonovich makes more than 4 turns, the polygon is not convex. So there are pockets in it.

Warning! To make your task more difficult, our definition of *pockets* might be different from what you may have heard before.

The gray area below indicates pockets of the polygon.



Formally, a point **p** is said to be in a pocket if it is not inside the polygon, and at least one of the following two conditions holds.

- There are boundary points directly both east and west of **p**; or
- There are boundary points directly both north and south of **p**.

Boundary points are the points traversed by Mr. Poligonovich on his walk (these include all points, not just those with integer coordinates).

Consider again the first picture from above. Point **x** satisfies the first condition; **y** satisfies both; **z** satisfies the second one. All three points are in pockets. The point **w** is not in a pocket.

Given Polygonovich's walk, your job is to find the total area of the pockets.

Input

The first line of input gives the number of cases, **N**. **N** test cases follow.

Each test case has the description of one walk of Professor Polygonovich. It starts with an integer **L**. Following are **L** "**S T**" pairs, where **S** is a string consisting of 'L', 'R', and 'F' characters, and **T** is an integer indicating how many times **S** is repeated.

In other words, the input for one test case looks like this:

S_1 T_1 S_2 T_2 ... S_L T_L

The actions taken are the concatenation of T_1 copies of S_1, followed by T_2 copies of S_2, and so on.

The "**S T**" pairs for a single test case may not all be on the same line, but the strings **S** will not be split across multiple lines. The second example below demonstrates this.

Output

For each test case, output one line containing "Case #**X**: **Y**", where **X** is the 1-based case number,

and **Y** is the total area of all pockets.

Limits

Time limit: 60 seconds per test set.

Memory limit: 1GB.

$1 \leq \mathbf{N} \leq 100$

$1 \leq \mathbf{T}$ (bounded from above by constraints in the problem statement, "Small dataset" and "Large dataset" sections)

The path, when concatenated from the input strings, will not have two consecutive direction changes (that is, there will be no 'LL', 'RR', 'LR', nor 'RL' in the concatenated path). There will be at least one 'F' in the path.

The path described will not intersect itself, except at the end, and it will end back at the origin.

Small dataset (Test set 1 - Visible)

$1 \leq \mathbf{L} \leq 100$

The length of each string **S** will be between 1 and 16, inclusive.

The professor will not visit any point with a coordinate bigger than 100 in absolute value.

Large dataset (Test set 2 - Hidden)

$1 \leq \mathbf{L} \leq 1000$

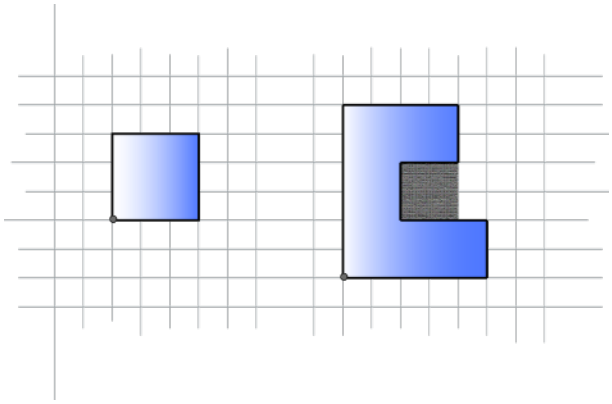
The length of each string **S** will be between 1 and 32, inclusive.

The professor will not visit any point with a coordinate bigger than 3000 in absolute value.

Sample

Input	Output
2 1 FFFR 4 9 F 6 R 1 F 4 RFF 2 LFF 1 LFFFR 1 F 2 R 1 F 5	Case #1: 0 Case #2: 4

The following picture illustrates the two sample test



cases.

#18

Problem

In the game of chess, there is a piece called the knight. A knight is special -- instead of moving in a straight line like other pieces, it jumps in an "L" shape. Specifically, a knight can jump from square (r1, c1) to (r2, c2) if and only if $(r1 - r2)^2 + (c1 - c2)^2 = 5$.

In this problem, one of our knights is going to undertake a chivalrous quest of moving from the top-left corner (the (1, 1) square) to the bottom-right corner (the (**H**, **W**) square) on a gigantic board. The chessboard is of height **H** and width **W**.

Here are some restrictions you need to know.

- The knight is so straightforward and ardent that he is only willing to move towards the right *and* the bottom. In other words, in each step he only moves to a square with a bigger row number and a bigger column number. Note that, this might mean that there is no way to achieve his goal, for example, on a 3 by 10 board.
- There are **R** squares on the chessboard that contain rocks with evil power. Your knight may not land on any of such squares, although flying over them during a jump is allowed.

Your task is to find the number of unique ways for the knight to move from the top-left corner to the bottom-right corner, under the above restrictions. It should be clear that sometimes the answer is huge. You are asked to output the remainder of the answer when divided by 10007, a prime number.

Input

Input begins with a line containing a single integer, **N**. **N** test cases follow.

The first line of each test case contains 3 integers, **H**, **W**, and **R**. The next **R** lines each contain 2 integers each, **r** and **c**, the row and column numbers of one rock. You may assume that (1, 1) and (**H**, **W**) never contain rocks and that no two rocks are at the same position.

Output

For each test case, output a single line of output, prefixed by "Case #**X**: ", where **X** is the 1-based case number, followed by a single integer indicating the number of ways of reaching the goal, modulo 10007.

Limits

Time limit: 30 seconds per test set.

Memory limit: 1GB.

$1 \le \mathbf{N} \le 100$

$0 \le \mathbf{R} \le 10$

Small dataset (Test set 1 - Visible)

$1 \le \mathbf{W} \le 100$

$1 \le \mathbf{H} \le 100$

$1 \le \mathbf{r} \le H$

$1 \le \mathbf{c} \le W$

Large dataset (Test set 2 - Hidden)

$1 \le \mathbf{W} \le 10^8$

$1 \le \mathbf{H} \le 10^8$

$1 \le \mathbf{r} \le H$

$1 \le \mathbf{c} \le W$

Sample

Input Output

```
5
1 1 0
4 4 1
2 1
3 3 0
7 10 2
1 2
7 1
4 4 1
3 2
```

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

Problem

Portal™ is a first-person puzzle/platform game developed and published by Valve Software. The idea of the game was to create two portals on walls and then jump through one portal and come out the other. This problem has a similar idea but it does not assume you have played Portal.

For this problem you find yourself in a **R** by **C** grid. Additionally there is a delicious cake somewhere else in the grid. You're very hungry and wish to arrive at the cake with as few moves as possible. You can move north, south, east or west to an empty cell. Additionally, you have the ability to create portals on walls.

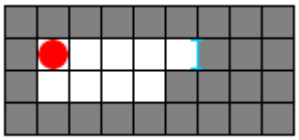
To help you get to the cake you have a portal gun which can shoot two types of portals, a yellow portal and a blue portal. A portal is created by shooting your portal gun either north, south, east or west. This emits a ball of energy that creates a portal on the first wall it hits. Note that for this problem shooting the portal gun does not count as a move. If you fire your portal gun at the cake, the energy ball will go right through it.

After creating a yellow portal and a blue portal, you can move through the yellow portal to arrive at the blue portal or vice versa. Using these portals you may be able to reach the cake even faster! You can only use portals after you create both a yellow and a blue portal.

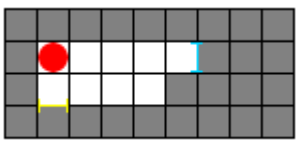
Consider the following grid:

Gray cells represent walls, white cells represent empty cells, and the red circle indicates your position.

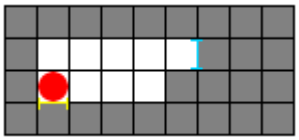
Suppose you shoot a blue portal east. The portal is created on the first wall it hits, resulting in:



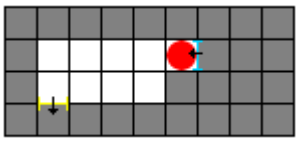
Now suppose you shoot a yellow portal south:



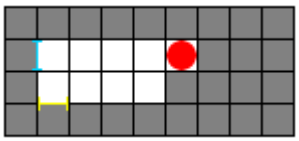
Next you move south once:



Now comes the interesting part. If you move south one more time you go through the yellow portal to the blue portal:



There can only be one yellow portal and one blue portal at any time. For example if you attempt to create a blue portal to the west the other blue portal will disappear:



A portal disappears only when another portal of the same color is fired.

Note that the portals are created on one side of the wall. If a wall has a portal on its east side you must move into the wall from the east to go through the portal. Otherwise you'll be moving into a wall, which is improbable.

Finally, you may not put two portals on top of each other. If you try to fire a portal at a side of a wall that already has a portal, the second portal will fail to form.

Given the maze, your initial position, and the cake's position, you want to find the minimum number of moves needed to reach the cake if it is possible. Remember that shooting the portal gun does not count as a move.

Input

The first line of input gives the number of cases, **N**. **N** test cases follow.

The first line of each test case will contain the integers **R** and **C** separated by a space. **R** lines follow containing **C** characters each, representing the map:

- `.` indicates an empty cell;
- `#` indicates a wall;
- `O` indicates your starting position; and
- `X` indicates the cake's position.

There will be exactly one `O` and one `X` character per case.

Cells outside of the grid are all walls and you may use them to create portals.

Output

For each test case you should output one line containing "Case #**X**: **Y**" (quotes for clarity) where **X** is the number of the test case and **Y** is the minimum number of moves needed to reach the cake or "THE CAKE IS A LIE" (quotes for clarity) if the cake cannot be reached.

Limits

Time limit: 30 seconds per test set.

Memory limit: 1GB.

Small dataset (Test set 1 - Visible)

N = 200

1 ≤ **R**, **C** ≤ 8

Large dataset (Test set 2 - Hidden)

N = 50

1 ≤ **R**, **C** ≤ 15

Sample

Input

```
3
4 7
.O..##.
.#.....
.#.####
.#...X.
5 5
O....
.....
.....
.....
....X
1 3
O#X
```

Output

```
Case #1: 4
Case #2: 2
Case #3: THE CAKE IS A LIE
```

Here is the sequence of moves for the first case (note that shooting the portal gun does not count as a move):

1.Move one step east.
2.Shoot a blue portal north.
3.Shoot a yellow portal south.
4.Move one step north through the blue portal.
5.Shoot a blue portal east.
6.Move one step south through the yellow portal.
7.Move one step west.

8.Eat your delicious and moist cake.

#20

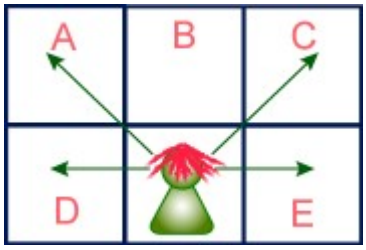
Problem

A local high school is going to hold a final exam in a big classroom. However, some students in this school are always trying to see each other's answer sheet during exams!

The classroom can be regarded as a rectangle of **M** rows by **N** columns of unit squares, where each unit square represents a seat.

The school principal decided to set the following rule to prevent cheating:

Assume a student is able to see his left, right, upper-left, and upper-right neighbors' answer sheets. The assignment of seats must guarantee that nobody's answer sheet can be seen by any other student.



As in this picture, it will not be a good idea to seat anyone in A, C, D, or E because the boy in the back row would be able to see their answer sheets. However, if there is a girl sitting in B, he will not be able to see her answer sheet.

Some seats in the classroom are broken, and we cannot put a student in a broken seat.

The principal asked you to answer the following question: What is the maximum number of students that can be placed in the classroom so that no one can cheat?

Input

The first line of input gives the number of cases, **C**. **C** test cases follow. Each case consists of two parts.

The first part is a single line with two integers **M** and **N**: The height and width of the rectangular classroom.

The second part will be exactly **M** lines, with exactly **N** characters in each of these lines. Each character is either a '.' (the seat is not broken) or 'x' (the seat is broken, lowercase x).

Output

For each test case, output one line containing "Case #**X**: **Y**", where **X** is the case number, starting

from 1, and **Y** is the maximum possible number of students that can take the exam in the classroom.

Limits

Time limit: 30 seconds per test set.

Memory limit: 1GB.

C = 20

Small dataset (Test set 1 - Visible)

$1 \leq \mathbf{M} \leq 10$

$1 \leq \mathbf{N} \leq 10$

Large dataset (Test set 2 - Hidden)

$1 \leq \mathbf{M} \leq 80$

$1 \leq \mathbf{N} \leq 80$

Sample

Input

```
4
2 3
...
...
2 3
x.x
xxx
2 3
x.x
x.x
10 10
....x.....
..........
..........
..x.......
..........
x...x.x...
.........x
...x......
........x.
.x...x....
```

Output

```
Case #1: 4
Case #2: 1
Case #3: 2
Case #4: 46
```

#21

Problem

You are studying animals in a forest, and are trying to determine which animals are birds and which are not.

You do this by taking two measurements of each animal – their height and their weight. For an animal to be a bird, its height needs to be within some range, and its weight needs to be within another range, but you're not sure what the height and weight ranges are. You also know that *every* animal that satisfies these ranges is a bird.

You have taken some of the animals you have measured and shown them to biologists, and they have told you which are birds and which are not. This has given you some information on what the height and weight ranges for a bird must be. For the remaining animals, your program should determine if they are definitely birds, definitely not birds, or if you don't know from the information you have.

Input

One line containing an integer **C**, the number of test cases in the input.

Then for each of the **C** test cases:

- One line containing an integer **N**, the number of animals you have shown to the biologists.
- **N** lines, one for each of these animals, each of the format "**H W X**", where **H** is the height of the animal, **W** is the weight of the animal, and **X** is either the string "BIRD" or "NOT BIRD". All numbers are positive integers.
- One line containing an integer **M**, the number of animals you have not shown to the biologists.
- **M** lines, one for each of these animals, each of the format "**H W**", where **H** is the height of the animal and **W** is the weight of the animal. All numbers are positive integers.

Output

For each of the **C** test cases:

- One line containing the string "Case #**X**:" where **X** is the number of the test case, starting from 1.
- M lines, each containing one of "BIRD", "NOT BIRD", or "UNKNOWN" (quotes are just for clarity and should not be part of the output).

Limits

Time limit: 30 seconds per test set.

Memory limit: 1GB.

$1 \leq \mathbf{C} \leq 10$

1 ≤ all heights and weights ≤ 1000000

Small dataset (Test set 1 - Visible)

$1 \leq \mathbf{N} \leq 10$

$1 \le \mathbf{M} \le 10$

Large dataset (Test set 2 - Hidden)

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

$1 \le \mathbf{M} \le 1000$

Sample

Input	Output
3 5 1000 1000 BIRD 2000 1000 BIRD 2000 2000 BIRD 1000 2000 BIRD 1500 2010 NOT BIRD 3 1500 1500 900 900 1400 2020 3 500 700 NOT BIRD 501 700 BIRD 502 700 NOT BIRD 2 501 600 502 501 1 100 100 NOT BIRD 3 107 93 86 70 110 115	Case #1: BIRD UNKNOWN NOT BIRD Case #2: UNKNOWN NOT BIRD Case #3: UNKNOWN UNKNOWN UNKNOWN

Case 1:

The animal "1500 1500" must be within the ranges for birds, since we know that the ranges for height and weight each include 1000 and 2000.

The animal "900 900" may or may not be a bird; we don't know if the ranges for height and weight include 900.

The animal "1400 2020" is within the height range for birds, but if 2020 was in the weight range, then the animal "1500 2010", which we know is not a bird, would also have to be within the weight range.

Case 2:

In this case we know that birds must have a height of 501. But we don't know what the weight range for a bird is, other than that it includes weight 700.

Case 3:

In this case, we know that anything with height 100 and weight 100 is not a bird, but we just don't

know what birds are.
#22

Problem

Oh no! The delicate political balance of the world has finally collapsed, and everybody has declared war on everybody else. You warned whoever would listen that this would happen, but did they pay attention? Ha! Now the only thing you can hope for is to survive as long as possible.

Fortunately (sort of), everyone's industrial centers have already been nuked, so the only method of attack available to each nation is to hurl wave after wave of conscripted soldiers at each other. This limits each nation to attacking only its immediate neighbors. The world is a R-by-C grid with R rows, numbered from 1 in the far North to R in the far South, and C columns, numbered from 1 in the far West to C in the far East. Each nation occupies one square of the grid, which means that each nation can reach at most 4 other adjacent nations.

Every nation starts with a specific strength value, known to everyone. They have no concept of advanced strategy, so at the beginning of each day, they will simply choose their strongest neighbor (breaking ties first by Northernmost nation, then by Westernmost) and attack them with an army. The army will have a power equal to the current strength S of the nation; by the end of the day, it will have depleted that neighbor's strength by S. A nation whose strength reaches 0 is destroyed. Note that all nations attack at the same time; an army's power is the same regardless of whether its nation is attacked that day.

Your nation is located at (c, r), in row r and column c. Fortunately, your nation is listening to your advice, so you don't have to follow this crazy strategy. You may choose to attack any of your neighbors on a given day (or do nothing at all). You can't attack multiple neighbors, however, or attack with an army of less than full power.

Determine the maximum number days you can survive.

Input

The first line of input gives the number of cases, **T**. **T** test cases follow. The first line of each test case contains four integers, **C**, **R**, **c**, and **r**. The next **R** lines each contain **C** integers, giving the starting strength $\mathbf{S_{ci,ri}}$ of the nation in column **ci** and row **ri**. It may be 0, indicating that the nation has already been destroyed. Your nation's starting strength will not be 0.

Output

For each test case, output one line containing "Case #**A**: " followed by:

- "**B** day(s)", where **B** is the most days you can hope to survive.
- "forever", if you can outlast all your neighbors.

Limits

Time limit: 30 seconds per test set.

Memory limit: 1GB.

$1 \le T \le 100$

$1 \le c \le C$

$1 \le r \le R$

Small dataset (Test set 1 - Visible)

$1 \le C \le 5$

$1 \le R \le 5$

$0 \le S_{ci,ri} \le 10$

Large dataset (Test set 2 - Hidden)

$1 \le C \le 50$

$1 \le R \le 50$

$0 \le S_{ci,ri} \le 1000$

Sample

Input	Output
2 3 3 2 2 2 3 2 1 7 1 2 1 2 4 3 2 1 1 2 2 0 10 8 5 10 10 2 9 10	Case #1: forever Case #2: 3 day(s)

#23

Problem

You have been invited to the popular TV show "Would you like to be a millionaire?". Of course you would!

The rules of the show are simple:

•Before the game starts, the host spins a wheel of fortune to determine **P**, the probability of winning each bet.

•You start out with some money: **X** dollars.

•There are **M** rounds of betting. In each round, you can bet any part of your current money, including none of it or all of it. The amount is not limited to whole dollars or whole cents.
If you win the bet, your total amount of money increases by the amount you bet. Otherwise, your amount of money decreases by the amount you bet.
•After all the rounds of betting are done, you get to keep your winnings (this time the amount is rounded down to whole dollars) only if you have accumulated $1000000 or more. Otherwise you get nothing.

Given **M**, **P** and **X**, determine your probability of winning at least $1000000 if you play optimally (i.e. you play so that you maximize your chances of becoming a millionaire).

Input

The first line of input gives the number of cases, **N**.

Each of the following **N** lines has the format "**M P X**", where:

•**M** is an integer, the number of rounds of betting.

•**P** is a real number, the probability of winning each round.

•**X** is an integer, the starting number of dollars.

Output

For each test case, output one line containing "Case #**X**: **Y**", where:

•**X** is the test case number, beginning at 1.

•**Y** is the probability of becoming a millionaire, between 0 and 1.

Answers with a relative or absolute error of at most 10^{-6} will be considered correct.

Limits

Time limit: 30 seconds per test set.

Memory limit: 1GB.

$1 \leq \mathbf{N} \leq 100$

$0 \leq \mathbf{P} \leq 1.0$, there will be at most 6 digits after the decimal point.

$1 \leq \mathbf{X} \leq 1000000$

Small dataset (Test set 1 - Visible)

$1 \leq \mathbf{M} \leq 5$

Large dataset (Test set 2 - Hidden)

$1 \leq \mathbf{M} \leq 15$

Sample

Input	Output
2 1 0.5 500000 3 0.75 600000	Case #1: 0.500000 Case #2: 0.843750

In the first case, the only way to reach $1000000 is to bet everything in the single round.

In the second case, you can play so that you can still reach $1000000 even if you lose a bet. Here's one way to do it:

- You have $600000 on the first round. Bet $150000.
- If you lose the first round, you have $450000 left. Bet $100000.
- If you lose the first round and win the second round, you have $550000 left. Bet $450000.
- If you win the first round, you have $750000 left. Bet $250000.
- If you win the first round and lose the second round, you have $500000 left. Bet $500000.

#24

Problem

You have pictures of two sculptures. The sculptures consist of several solid metal spheres, and some rubber pipes connecting pairs of spheres. The pipes in each sculpture are connected in such a way that for any pair of spheres, there is exactly one path following a series of pipes (without repeating any) between those two spheres. All the spheres have the same radius, and all the pipes have the same length.

You suspect that the smaller of the two sculptures was actually created by simply removing some spheres and pipes from the larger one. You want to write a program to test if this is possible.

The input will contain several test cases. One sculpture is described by numbering the spheres consecutively from 1, and listing the pairs of spheres which are connected by pipes. The numbering is chosen independently for each sculpture.

Input

- One line containing an integer **C**, the number of test cases in the input file.

For each test case, there will be:

- One line containing the integer **N**, the number of spheres in the large sculpture.

•**N**−1 lines, each containing a pair of space-separated integers, indicating that the two spheres with those numbers in the large sculpture are connected by a pipe.
•One line containing the integer **M**, the number of spheres in the small sculpture.
•**M**−1 lines, each containing a pair of space-separated integers, indicating that the two spheres with those numbers in the small sculpture are connected by a pipe.

Output

•**C** lines, one for each test case in the order they occur in the input file, containing "Case #**X**: YES" if the small sculpture in case **X** could have been created from the large sculpture in case **X**, or "Case #**X**: NO" if it could not. (**X** is the number of the test case, between 1 and **C**.)

Limits

Time limit: 30 seconds per test set.

Memory limit: 1GB.

Small dataset (Test set 1 - Visible)

$1 \leq \mathbf{C} \leq 100$

$2 \leq \mathbf{N} \leq 8$

$1 \leq \mathbf{M} < N$

Large dataset (Test set 2 - Hidden)

$1 \leq \mathbf{C} \leq 50$

$2 \leq \mathbf{N} \leq 100$

$1 \leq \mathbf{M} < N$

Sample

Input Output

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

```
Case #1: NO
Case #2: YES
```

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

In the first case, the large sculpture has five spheres connected in a line, and the small sculpture has one sphere that has three other spheres connected to it. There's no way the smaller sculpture could have been made by removing things from the larger one.
In the second case, the small sculpture is four spheres connected in a line. These can match the larger sculpture's spheres in the order 2-1-4-5.

#25

Problem

You are following a recipe to create your lunch.
The recipe is a mixture made by combining ingredients together in a bowl. Each ingredient will be either:

- Another mixture which you must make first in a separate bowl; or
- A basic ingredient you already have in your kitchen, which can be added directly.

To make a mixture, you need to have all its ingredients ready, take an empty bowl and mix the ingredients in it. It is not possible to make mixtures by adding ingredients to an already-existing mixture in a bowl.
For example, if you want to make CAKE (a mixture) out of CAKEMIX (a mixture) and lies (a basic ingredient), then you must first make CAKEMIX in its own bowl, then add the CAKEMIX and lies to a second bowl to make the CAKE.
Once you have used a mixture as an ingredient and emptied the bowl it was prepared in, you can re-use that bowl for another mixture. So the number of bowls you need to prepare the recipe will depend on the order in which you decide to make mixtures.
Determine the minimum number of bowls you will need.

Input

The first line will contain an integer **C**, the number of test cases in the input file.
For each test case, there will be:

- One line containing an integer **N**, the number of mixtures in the test case.
- **N** lines, one for each mixture, containing:

•One string giving the mixture name;
•An integer **M**, the number of ingredients in this mixture;
•**M** strings, giving the names of each of the ingredients of this mixture.

The tokens on one line will be separated by single spaces.

The first mixture in a test case is the recipe you are making.

The names of mixtures are strings of between 1 and 20 UPPERCASE letters.

The names of basic ingredients are strings of between 1 and 20 lowercase letters.

Each mixture is used in exactly one other mixture, except for the recipe, which is not used in any other mixture. Each ingredient will appear at most once in the ingredient list for a mixture. No mixture will (directly or indirectly) require itself as an ingredient.

Output

For each test case, output one line containing "Case #**X**: **Y**" where **X** is the number of the test case, starting from 1, and **Y** is the minimum number of mixing bowls required.

Limits

Time limit: 30 seconds per test set.

Memory limit: 1GB.

$1 \leq \mathbf{C} \leq 10$

$2 \leq \mathbf{M} \leq 10$

Small dataset (Test set 1 - Visible)

$1 \leq \mathbf{N} \leq 10$

Large dataset (Test set 2 - Hidden)

$1 \leq \mathbf{N} \leq 1000$

Sample

Input	Output
2 3 SOUP 3 STOCK salt water STOCK 2 chicken VEGETABLES VEGETABLES 2 celery onions 5 MILKSHAKE 4 milk icecream FLAVOR FRUIT FRUIT 2 banana berries FLAVOR 2 SPICES CHOCOLATE SPICES 2 nutmeg cinnamon CHOCOLATE 2 cocoa syrup	Case #1: 2 Case #2: 3

In the first case, to satisfy your craving for SOUP, you follow these steps:

1.Make VEGETABLES by mixing celery and onions in a bowl.
2.Make STOCK in a second bowl by mixing chicken and VEGETABLES from the first bowl. The first bowl becomes empty.
3.Make SOUP in the first bowl by mixing STOCK, salt and water.

In the second case, you have a choice of whether to make FLAVOR or FRUIT first before mixing them with milk and icecream to make MILKSHAKE.

If we make FRUIT first, we use four bowls:

1.Make FRUIT in a bowl by mixing banana and berries.
2.Make SPICES in a second bowl by mixing nutmeg and cinnamon, and CHOCOLATE in a third bowl by mixing cocoa and syrup. (In either order)
3.Make FLAVOR in a fourth bowl by mixing SPICES and CHOCOLATE.
4.Make MILKSHAKE in the second or third bowl by mixing FRUIT, FLAVOR, milk and icecream.

However if we make FRUIT after FLAVOR, we use three bowls:

1.Make SPICES in a bowl by mixing nutmeg and cinnamon, and CHOCOLATE in a second bowl by mixing cocoa and syrup. (In either order)
2.Make FLAVOR in a third bowl by mixing SPICES and CHOCOLATE.
3.Make FRUIT in the first bowl by mixing banana and berries.
4.Make MILKSHAKE in the second bowl by mixing FRUIT, FLAVOR, milk and icecream.

#26

Problem

Dave is taking a multiple choice test on the Internet. Dave possibly gets many opportunities to submit his answers to the test, but he passes only if he gets all the questions correct. He must answer every question on the test to make a submission. The only information he receives after he submits is whether he has passed.

For each question, he estimates the probability that each of 4 responses is correct, independent of his responses to other questions. Given a fixed number of submissions he can make, Dave wants to choose his responses so that he maximizes the probability of passing the test.

What is the probability that Dave will pass the test if he chooses his responses optimally?

Input

The first line of input gives the number of cases, **C**. **C** test cases follow.

Each test case starts with a line containing **M** and **Q**. Dave is allowed to make **M** submissions to solve the test. There are **Q** questions on the test. **Q** lines follow, each containing 4 probabilities of

correctness. There will be at most 6 digits after the decimal point. The probabilities for each line are non-negative and sum to 1.

Output

For each test case, output one line containing "Case #**X**: **Y**" where **X** is the number of the test case (starting from 1), and **Y** is the probability of success.
Answers with a relative or absolute error of at most 10^{-6} will be considered correct.

Limits

Memory limit: 1GB.

$1 \leq \mathbf{C} \leq 100$

Small dataset (Test set 1 - Visible)

Time limit: 60 seconds.

$1 \leq \mathbf{Q} \leq 6$

$1 \leq \mathbf{M} \leq 1000$

Large dataset (Test set 2 - Hidden)

Time limit: 180 seconds.

$1 \leq \mathbf{Q} \leq 30$

$1 \leq \mathbf{M} \leq 10000$

Sample

Input	Output
3 10 2 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 64 3 0.3 0.4 0.0 0.3 1.0 0.0 0.0 0.0 0.2 0.2 0.2 0.4 3 2 0.5 0.17 0.17 0.16 0.5 0.25 0.25 0.0	Case #1: 0.625 Case #2: 1.0 Case #3: 0.5

#27

Problem

You are trying to compute the next number in a sequence S_n generated by a secret code. You know

that the code was generated according to the following procedure.

First, for each k between 0 and 29, choose a number C_k between 0 and 10006 (inclusive).

Then, for each integer n between 0 and 1 000 000 000 (inclusive):

- Write n in binary.
- Take the numbers C_k for every bit k that is set in the binary representation of n. For example, when n=5, bits 0 and 2 are set, so C_0 and C_2 are taken.
- Add these C_k together, divide by 10007, and output the remainder as S_n.

You will be given a series of consecutive values of sequence S, but you don't know at which point in the sequence your numbers begin (although you do know that there is at least one more number in the sequence), and you don't know what values of C_k were chosen when the sequence was generated.

Find the next number in the sequence, or output UNKNOWN if this cannot be determined from the input data.

Input

The first line will contain an integer **T**, the number of test cases in the input file.

For each test case, there will be:

- One line containing the integer **N**, the number of elements of sequence S that you have.
- One line containing **N** single-space-separated integers between 0 and 10006, the known elements of the sequence.

Output

For each test case, output one line containing "Case #**X**: **Y**" where **X** is the number of the test case, starting from 1, and **Y** is the next number in the sequence, or the string UNKNOWN if the next number cannot be determined.

Limits

Time limit: 30 seconds per test set.

Memory limit: 1GB.

$1 \leq \mathbf{T} \leq 20$

Small dataset (Test set 1 - Visible)

$1 \leq \mathbf{N} \leq 5$

Large dataset (Test set 2 - Hidden)

$1 \leq \mathbf{N} \leq 1000$

Sample

Input	Output
3 7 1 2 3 4 5 6 7 4 1 10 11 200 4 1000 1520 7520 7521	Case #1: UNKNOWN Case #2: 201 Case #3: 3514

In the first case, C_0, C_1 and C_2 might have been 1, 2 and 4, and the values of S_n we have starting at n=1. If this is correct, we don't know C_3, so the next number in the sequence could be anything! Therefore the answer is unknown.
In the second case, we cannot know all the values of C_k or even what n is, but we can prove that in any sequence, if 1, 10, 11, 200 occur in order, then the next value will always be 201.

#28

Problem

Alice and Bob want to play a game. The game is played on a chessboard with **R** rows and **C** columns, for a total of **RC** squares. Some of these squares are burned.
A king will be placed on an unburned square of the board, and Alice and Bob will make successive moves with the king.
In a move, the player must move the king to any of its 8 neighboring squares, with the following two conditions:

- The destination square must not be burned
- The king must never have been in the destination square before.

If a player can't make a move, he or she loses the game. Alice will move first; you need to determine who will win, assuming both players play optimally.

Input

The first line of input gives the number of cases, **N**.
N test cases follow. The first line of each case will contain two integers, **R** and **C**. The next **R** lines will contain strings of length **C**, representing the **C** squares of each row. Each string will contain only the characters '.', '#' and 'K':

- '#' means the square is burned;
- '.' means the square is unburned, and unoccupied; and

•'K' means the king is in that cell at the beginning of the game.
There will be only one 'K' character in each test case.

Output

For each test case, output one line containing "Case #**X**: " (where **X** is the case number, starting from 1) followed by A if Alice wins, or B if Bob wins.

Limits

Memory limit: 1GB.

$1 \leq \mathbf{N} \leq 100$

Small dataset (Test set 1 - Visible)

Time limit: 30 seconds.

$1 \leq \mathbf{R}, \mathbf{C} \leq 4$

Large dataset (Test set 2 - Hidden)

Time limit: 180 seconds.

$1 \leq \mathbf{R}, \mathbf{C} \leq 15$

Sample

Input Output

```
2
2 2
K.
.#
4 2  Case #1: B
K#   Case #2: A
.#
.#
.#
```

#29

Problem

You need to hire some people to paint a fence. The fence is composed of 10000 contiguous sections, numbered from 1 to 10000.

You get some offers from painters to help paint the fence. Each painter offers to paint a contiguous subset of fence sections in a particular color. You need to accept a set of the offers, such that:

•Each section of the fence is painted.
•At most 3 colors are used to paint the fence.

If it is possible to satisfy these two requirements, find the minimum number of offers that you must accept.

Input

•One line containing an integer **T**, the number of test cases in the input file.
For each test case, there will be:

•One line containing an integer **N**, the number of offers.
•**N** lines, one for each offer, each containing "**C A B**" where **C** is the color, which is an uppercase string of up to 10 letters, **A** is the first section and **B** is the last section to be painted. 1 ≤ **A** ≤ **B** ≤ 10000.

Output

•**T** lines, one for each test case in the order they occur in the input file, each containing the string "Case #**X**: **Y**", where **X** is the case number, and **Y** is the number of offers that need to be accepted, or "Case #**X**: IMPOSSIBLE" if there is no acceptable set of offers.

Limits

Time limit: 30 seconds per test set.

Memory limit: 1GB.

1 ≤ **T** ≤ 50

Small dataset (Test set 1 - Visible)

1 ≤ **N** ≤ 10

Large dataset (Test set 2 - Hidden)

1 ≤ **N** ≤ 300

Sample

Sample Input

save_alt

content_copy

```
5
2
BLUE 1 5000
RED 5001 10000
3
BLUE 1 6000
RED 2000 8000
WHITE 7000 10000
```

```
4
BLUE 1 3000
RED 2000 5000
ORANGE 4000 8000
GREEN 7000 10000
2
BLUE 1 4000
RED 4002 10000
3
BLUE 1 6000
RED 4000 10000
ORANGE 3000 8000
```

Sample Output

save_alt

content_copy

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

In the first test case, accepting both offers will exactly paint the whole fence, 5000 sections each, with no overlap.

In the second case, the painters will overlap, which is acceptable.

In the third case, accepting all four offers would cover the whole fence, but it would use 4 different colours, so this is not acceptable.

In the fourth case, section 4001 cannot be painted.

In the fifth case, we can accept just the first and second offer and successfully paint the fence.

#30

Problem

You are given two triangle-shaped pictures. The second picture is a possibly translated, rotated and scaled version of the first. The two triangles are placed on the table, with the second one placed completely inside (possibly touching the boundary of) the first one. The second triangle is always scaled by a factor that is strictly between 0 and 1.

You need to process the picture, and for that you need a point in the picture which overlaps with the same point of the scaled picture. If there is more than one solution, you can return any of them. If there are no solutions, print "No Solution" (without the quotes) for that test case.

Input

The first line of input gives the number of cases, **N**. Then for each test case, there will be two lines, each containing six space-separated integers -- the coordinates of one of the triangles -- in the format "x_1 y_1 x_2 y_2 x_3 y_3". The point (x_1, y_1) in the first triangle corresponds to the same corner of the picture as (x_1, y_1) in the second triangle, and similarly for (x_2, y_2) and (x_3, y_3).

Output

For each test case, output one line containing "Case #**x**: " followed two real numbers representing the coordinates of the overlapping point separated by one space character, or the string "No Solution". Answers with a relative or absolute error of at most 10^{-5} will be considered correct.

Limits

Time limit: 30 seconds per test set.

Memory limit: 1GB.

$1 \leq \mathbf{N} \leq 10$.

The coordinates of the points will be integer numbers between -10 000 and 10 000. The three points in each triangle will not be collinear.

Small dataset (Test set 1 - Visible)

All tests will contain isosceles right-angle triangles. (i.e., the triangle's angles will be 45 degrees, 45 degrees, and 90 degrees.)

Large dataset (Test set 2 - Hidden)

The triangles can have any shape.

Sample

Input	Output
2 0 0 0 2 2 0 0 0 0 1 1 0 10 0 0 10 0 0 3 3 1 1 3 1	Case #1: 0.000000 0.000000 Case #2: 2.692308 1.538462

#31

Problem

In graph theory, a *tree* is a connected, undirected simple graph with no cycles. A tree with **n** nodes always has **n** - 1 edges.

A *path* in a tree is a sequence of distinct edges which are connected (each pair of consecutive edges in the path share a vertex).

Consider a tree with **n** vertices and **n**-1 edges. You can color each edge in one of **k** colors.

An assignment of colors to edges is a *rainbow coloring* if in every path of 2 or 3 edges, the colors of the edges are different. (i.e., every two consecutive edges have different colors, and every three consecutive edges have different colors).

Given a tree and the number of colors **k**, find the number of rainbow colorings modulo 1000000009.

Input

The first line of input gives the number of test cases, **C**. Then for each of the **C** cases, there will be:

•One line containing two integers in the format "**n k**". **n** is the number of nodes in the tree, and **k** is the number of colors available.

•**n** - 1 lines, one for each edge, containing two integers "**x y**", indicating that the edge is between node **x** and node **y**. Nodes are numbered from 1 to **n**.

Output

For each test case, output one line. That line should contain "Case #**X**: **Y**", where **X** is 1-based number of the case, and **Y** is the answer for that test case.

Limits

Time limit: 30 seconds per test set.

Memory limit: 1GB.

$1 \leq \mathbf{k} \leq 1000000000$

All the node numbers are between 1 and **n**, inclusive.

Small dataset (Test set 1 - Visible)

$1 \leq \mathbf{C} \leq 100$

$2 \leq \mathbf{n} \leq 20$

Large dataset (Test set 2 - Hidden)

$1 \leq \mathbf{C} \leq 40$

$2 \leq \mathbf{n} \leq 500$

Sample

Input Output

```
2
4 10
1 2
1 3
1 4  Case #1: 720
5 3  Case #2: 6
1 2
2 3
3 4
4 5
```

In the first case, the tree has four nodes. There are edges from one node to each of the other three. Each pair of these edges are adjacent, so for there to be a rainbow coloring, all the edges must have different colors. There are therefore 10 x 9 x 8 = 720 rainbow colorings.
In the second case, the tree itself is a path of 4 edges, and there are 3 colors. The first three edges must all have different colors, so there are 3 x 2 x 1 colorings for these, and then there is only one choice for the fourth edge, so there are 6 rainbow colorings.

#32

Problem

In the First City of Mars there are **N** bus stops, all aligned in a straight line of length **N**-1 km. The mayor likes to keeps things simple, so he gave the bus stops numbers from 1 to **N**, and separated adjacent stops by exactly 1 km.
There are also **K** buses in the city. The mayor has to plan the bus schedule and he would like to know in how many ways that can be done. This number can be very large. Luckily there are a few constraints:

- In the beginning of the day all the buses are in the first **K** bus stops (one bus per stop)
- Buses only move from the left to the right (1 is the leftmost bus stop)
- At the end of the day all the buses must be in the last **K** bus stops (one bus per stop)
- In each bus station exactly one bus has to stop
- For the same bus the distance between any two consecutive stops is at most **P** km

Help the mayor evaluate the number of schedules. However try not to give him very bad news (a lot of schedules) so just output the real number modulo 30031.

Input

The first line in the input file is the number of cases **T**.

Each of the next **T** lines contains 3 integers separated by one space: **N**, **K** and **P**.

Output

For each case output the number of ways to plan the bus schedules (modulo 30031) in the format "Case #**t**: [number of ways modulo 30031]" where **t** is the number of the test case, starting from 1.

Limits

Time limit: 30 seconds per test set.

Memory limit: 1GB.

$1 < \mathbf{T} \leq 30$

$1 < \mathbf{P} \leq 10$

$\mathbf{K} < \mathbf{N}$

$1 < \mathbf{K} \leq \mathbf{P}$

Small dataset (Test set 1 - Visible)

$1 < \mathbf{N} < 1000$

Large dataset (Test set 2 - Hidden)

$1 < \mathbf{N} < 10^9$

Sample

Input	Output
3 10 3 3 5 2 3 40 4 8	Case #1: 1 Case #2: 3 Case #3: 7380

Let's name the buses: A, B, C...

For the first case there is only one possible way of planning the schedule: A → 1, 4, 7, 10. B → 2, 5, 8. C → 3, 6, 9.

For the second case the possible ways of planning are:

(A → 1,3,5. B → 2,4),

(A → 1,3,4. B → 2,5),

(A → 1,4. B → 2,3,5).

#33

Problem

You are holding a party. In preparation, you are making a drink by mixing together three different types of fruit juice: Apple, Banana, and Carrot. Let's name the juices A, B and C.

You want to decide what fraction of the drink should be made from each type of juice, in such a way that the maximum possible number of people attending the party like it.

Each person has a minimum fraction of each of the 3 juices they would like to have in the drink. They will only like the drink if the fraction of each of the 3 juices in the drink is greater or equal to their minimum fraction for that juice.

Determine the maximum number of people that you can satisfy.

Input

•One line containing an integer **T**, the number of test cases in the input file.

For each test case, there will be:

•One line containing the integer **N**, the number of people going to the party.

•**N** lines, one for each person, each containing three space-separated numbers "A B C", indicating the minimum fraction of each juice that would like in the drink. A, B and C are integers between 0 and 10000 inclusive, indicating the fraction in parts-per-ten-thousand. $A + B + C \leq 10000$.

Output

•**T** lines, one for each test case in the order they occur in the input file, each containing the string "Case #X: Y" where X is the number of the test case, starting from 1, and Y is the maximum number of people who will like your drink.

Limits

Time limit: 30 seconds per test set.

Memory limit: 1GB.

$1 \leq \mathbf{T} \leq 12$

Small dataset (Test set 1 - Visible)

$1 \leq \mathbf{N} \leq 10$

Large dataset (Test set 2 - Hidden)

$1 \leq \mathbf{N} \leq 5000$

Sample

Input Output

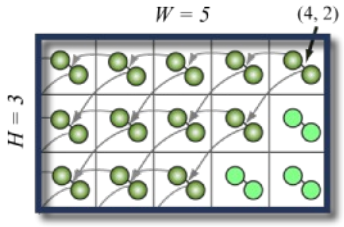
```
3
3
10000 0 0
0 10000 0
0 0 10000
3
5000 0 0        Case #1: 1
0 2000 0        Case #2: 2
0 0 4000        Case #3: 5
5
0 1250 0
3000 0 3000
1000 1000 1000
2000 1000 2000
1000 3000 2000
```

In the first case, for each juice, we have one person that wants the drink to be made entirely out of that juice! Clearly we can only satisfy one of them.
In the second case, we can satisfy any two of the three preferences.
In the third case, all five people will like the drink if we make it using equal thirds of each juice.

#34

Problem

A large room is filled with mousetraps, arranged in a grid. Each mousetrap is loaded with two ping-pong balls, carefully placed so that when the mousetrap goes off they will be flung, land on other mousetraps and set them off. The walls of the room are sticky, so any balls that hit the walls of the room are effectively absorbed.
Every mousetrap that gets hit sends the two ping-pong balls in the same way: their movement is determined by a X and Y displacement relative to the launching mousetrap. You then decide to launch a single ping-pong ball into the room. It hits a mousetrap, setting it off, and launching its two balls. These two balls then set off two more mousetraps, and now four balls fly off... When the dust settles, many of the mousetraps have been set off, but some have been missed by all the flying balls. You need to calculate how many mousetraps will be set off.
As an example (see the first sample test case), the picture below illustrates a room with width 5, height 3. The two directions for the ping-pong balls in each room are (-1, 0) and (-1, -1), respectively. The first ball you launch hits the mousetrap at the position (4, 2). In the end, 12 mousetraps are triggered.



Input

The first line of input gives the number of cases, **C**. **C** test cases follow. Each case contains four lines. The first line is the size of the grid of mousetraps (equal to the size of the room), given as its width **W** and height **H**. The next two lines give the destinations of the two ping-pong balls, as an X and Y displacement. For example, if the two lines were `0 1` and `1 1`, then triggering a mousetrap would launch two balls; one would hit the mousetrap just up from the triggered mousetrap, and the other would hit the mousetrap that is up and to the right of the triggered mousetrap. The final line has two integers specifying, respectively, the column and row of the mousetrap set off by the original ping-pong ball (where 0 0 would be the bottom left mousetrap).

Output

For each test case, output one line containing "Case #**A**: **B**", where **A** is 1-based number of the case and **B** is the number of mousetraps that are triggered (including the first one).

Limits

Time limit: 30 seconds per test set.

Memory limit: 1GB.

$1 \leq \mathbf{C} \leq 100$

$-20 \leq$ any displacement ≤ 20

Neither vector will have zero length.

Small dataset (Test set 1 - Visible)

$2 \leq \mathbf{W}, \mathbf{H} \leq 100$

Large dataset (Test set 2 - Hidden)

$2 \leq \mathbf{W}, \mathbf{H} \leq 1000000$

Sample

Input	Output
3 5 3 -1 0	Case #1: 12 Case #2: 820 Case #3: 5

```
-1 -1
4 2
50 50
0 1
1 1
10 10
6 2
2 0
3 0
0 0
```

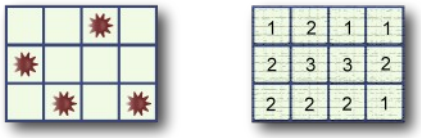
#35

Problem

MineLayer is a MineSweeper-like puzzle game played on an **R** by **C** grid. Each square in the grid either has one mine or no mines at all. A MineLayer puzzle consists of a grid of numbers, each of which indicates the total number of mines in all adjacent squares and in the square underneath. The numbers will thus range from zero to nine.

The objective of MineLayer is to figure out a layout of the mines in the grid that matches the given clues.

Below is a typical 3 by 4 grid. The original layout is on the left, and the puzzle on the right.



Since there may be many solutions, your task is to write a program that outputs the maximum possible number of mines in the middle row. The number of rows will always be odd, and there will always be at least one solution to the puzzle.

Input

The first line of input gives the number of cases, **N**. **N** test cases follow.

The first line of each case contains two space-separated numbers: **R**, the number of rows, and **C**, the number of columns. **R** is always an odd integer. Each of the next **R** lines contains **C** space-separated numbers that denote the clues of that row.

Output

For each test case, output one line containing "Case #X: Y", where X is the 1-based case number,

and Y is the maximum possible number of mines in the middle row of a grid that satisfies the given constraints.

Limits

Time limit: 30 seconds per test set.

Memory limit: 1GB.

$1 \leq \mathbf{N} \leq 50$.

Each puzzle is guaranteed to have at least one solution.

Small dataset (Test set 1 - Visible)

$\mathbf{R} = 3$ or $\mathbf{R} = 5$.

$3 \leq \mathbf{C} \leq 5$.

Large dataset (Test set 2 - Hidden)

R is an odd number between 3 and 49, inclusive.

$3 \leq \mathbf{C} \leq 49$.

Sample

Input

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

Output

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

#37

Problem

The king wants bridges built and he wants them built as quickly as possible. The king owns an **N** by **M** grid of land with each cell separated from its adjacent cells by a river running between them and he wants you to figure out how many man-hours of work it will take to build enough bridges to connect every island. Some cells are actually lakes and need not have a bridge built to them.

Some of the islands are forests where trees are abundant. Located in the top left corner is the *base*

camp, which is always a forest.

A bridge can only be built between two islands if they are vertically or horizontally adjacent, and one of the islands is accessible from the base camp through the bridges that are already built.

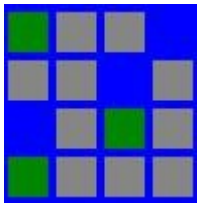
The number of man-hours it takes to build a bridge is the number of bridges the builders have to cross to get from the nearest forest to the island you're building to, including the bridge being built.

Builders can only walk between two islands if there is already a bridge between them.

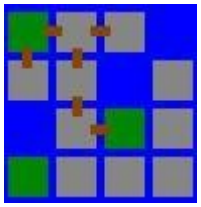
The king has already ensured that there is at least one way to connect all the islands.

Write a program that, given a map of the islands, will output the minimum number of man-hours required to connect all islands.

Consider this example. A green tile indicates a forest, gray indicates an empty island, and blue indicates water.

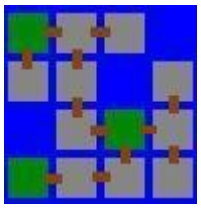


One optimal solution starts out by building the following bridges from the base camp forest.



This has a cost of 1 + 2 + 1 + 2 + 3 + 4 = 13

Now since the forest at row 3, column 3 is connected to base camp, we can build bridges from there. One optimal solution connects the rest of the islands with bridges built from this forest.



This has a cost of 2 + 1 + 2 + 1 + 2 + 3 = 11. This brings the total cost to 24 which is the optimal solution.

Input

The first line of the input contains an integer **T**, the number of test cases. **T** test cases follow. Each test case will begin with **N**, the number of rows, and **M**, the number of columns, on one line separated by a space. **N** rows follow that contain exactly **M** characters each. A 'T' indicates an

island with a forest, a '#' indicates an island, and a '.' indicates water.

Output

A single line containing "Case #X: Y", where **X** is the 1-based case number, and **Y** is the minimum number of man-hours needed to connect all islands.

Limits

Time limit: 30 seconds per test set.

Memory limit: 1GB.

$1 \leq \mathbf{T} \leq 50$

$2 \leq \mathbf{N} \leq 30$

$2 \leq \mathbf{M} \leq 30$

The top left cell will always be a 'T'

It will be possible to connect all islands through bridges

Small dataset (Test set 1 - Visible)

There will be at most 2 forests in the grid including the base camp.

Large dataset (Test set 2 - Hidden)

There will be no limit on the number of forests in the grid.

Sample

Input Output

```
3
2 2
T.
T#
4 4
T##.
##.#  Case #1: 2
.#T#  Case #2: 24
####  Case #3: 49
5 5
T#T.#
..#.#
#.###
###.#
T###T
```

#38

Problem

The year 2008 will be known as a year of change and transition, the start of a new era: we're talking, of course, about the new Google Code Jam format. The introduction of this contest has jammed so many great programming competitions together in a single year that people have started calling it *The Year of Code Jam*.

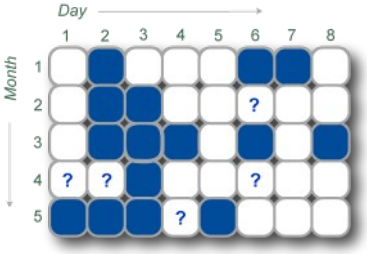
Sphinny, a passionate contestant, is looking at her calendar of the year and discovering that a great number of programming contests has been scheduled. She has marked every day of the year on the calendar in one of the three ways:

- White: She will not participate in a contest on this day. Either no contests are scheduled, or she has more important things to do (surely there are other good things in life!).
- Blue: She will definitely participate in a contest on this day.
- Question mark: There is a contest scheduled, but she has not decided yet whether she will participate.

Note: To simplify the problem, we'll assume that there is no concept of qualification: you don't have to participate in one contest to be eligible for another.

Being in a world that is somewhat different from ours, Sphinny's calendar has some features we must mention: It has **N** months, and each month has exactly **M** days.

The picture below depicts a calendar with 5 months, 8 days in each month, 15 blue days, and 5 question marks.



Looking at her beautiful calendar, Sphinny has decided that each day has up to 4 **neighbors** in the year: The previous day in the same month, the next day in the same month, the same day in the previous month, and the same day in the next month.

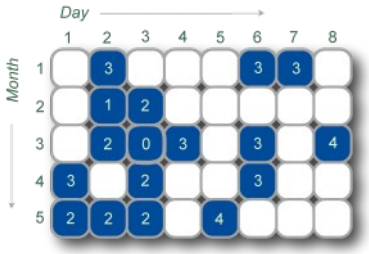
Sphinny wants to maximize her happiness from these contests, and she estimates the effect of the contests on her happiness as a summation of values for all the blue days. For each blue day, the value is computed as follows:

- The initial value is 4.
- For each blue neighbour the day has, decrease the value by 1.

You may think that Sphinny likes the contests, but participating on two consecutive days makes her a little tired. And for aesthetic reasons, participating on the same day in two consecutive months is also not so great.

Sphinny wants to plan her year now, and decide for every day with a question mark whether it should be white or blue. Her goal is simply to maximize the happiness value.

The following picture shows a solution for the example above. By changing two question marks to blue days, and the other three to white days, she can achieve a happiness value of 42.



Input

The first line in the input file contains the number of cases **T**. This is followed by **T** cases in the following format.

The first line is of the form "**N M**", where **N** and **M** are two numbers giving the number of months and the number of days per month.

The next **N** lines each contain a string of length **M**. The **j**-th character in the **i**-th string is one of {'#', '.', '?'}, which gives the status of the j-th day in the i-th month. '#' indicates a blue day, '.' indicates a white day, and '?' indicates a day with a question mark.

Output

For each input case, you should output a line in the format:

```
Case #X: Y
```

where **X** is the 1-based case number, and **Y** is the maximum happiness value.

Limits

Memory limit: 1GB.

1 ≤ **T** ≤ 100.

Small dataset (Test set 1 - Visible)

Time limit: 30 seconds.

1 ≤ **M**, **N** ≤ 15.

Large dataset (Test set 2 - Hidden)

Time limit: 120 seconds.

1 ≤ **M**, **N** ≤ 50.

Sample

Input	Output
2 3 3 .?. .?.	Case #1: 8 Case #2: 42

```
.#.
5 8
.#...##.
.##..?..
.###.#.#
??#..?..
###?#...
```

Note that the second sample is our example in the pictures above.